The ACB’s of VSAM

Chapter Prerequisite:

You should understand assembler Move Mode and Locate Mode I/O for QSAM files.

What is VSAM?

The acronym VSAM stands for *Virtual Storage Access Method*. Access methods are software systems that support file processing. IBM’s VSAM access methods are some of the most extensive and complex of any operating system. VSAM records have a unique format that is not recognized by other access methods. IBM uses the term *data set* as a synonym for *file* and *DASD* (direct access storage device) for devices that provide random access to record locations. VSAM supports four file types:

1) ESDS (Entry Sequence Data Sets)—This type of data set provides sequential processing of records.

2) RRDS (Relative Record Data Sets)—Allows sequential and random access by relative record number.

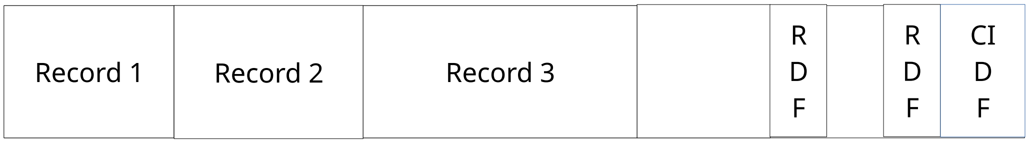
3) KSDS (Key-Sequenced Data Sets)—Allows sequential, skip-sequential, and random processing by key.

4) LDS (Linear Data Sets)—This type of data set is the only byte-stream data set in traditional z/OS files, although IBM z machines run other operating systems like z/OS UNIX which support byte-stream methods. This format is rarely used for application programs.

KSDS data sets are by far the most heavily used of the four types. This is followed by ESDS, RRDS, and LDS. QSAM, the Queued-Sequential Access Method, a non-VSAM data type, is the most heavily used for sequential files. ESDS provides sequential processing for VSAM data sets, but is slower than QSAM processing. For that reason, most IBM shops use QSAM for sequential file processing and VSAM KSDS processing for random access to data sets.

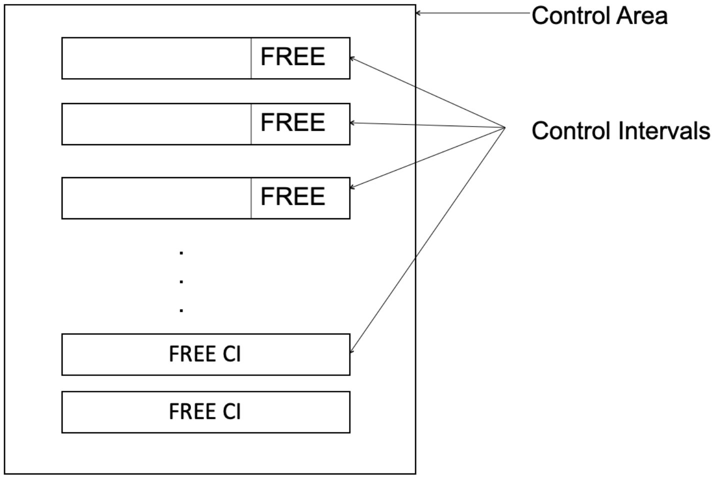
VSAM data sets are organized as *clusters*. For ESDS and RRDS, the cluster consists of only a data component. For KSDS data sets, the cluster contains a data component and an index component that supports random and sequential record access. VSAM data records are stored on DASD in *control intervals* (CIs) which are grouped into *control areas* (CAs). Within a cluster there can be multiple control areas. Within a control area there can be many control intervals. To allow for the addition of records each CI can contain free space. Within a CA there can be empty (free) CIs, and within a cluster there can be free CAs. VSAM is designed to provide efficient insertion and deletion of records. A single CI and CA are pictured below.

A Control Interval (CI)



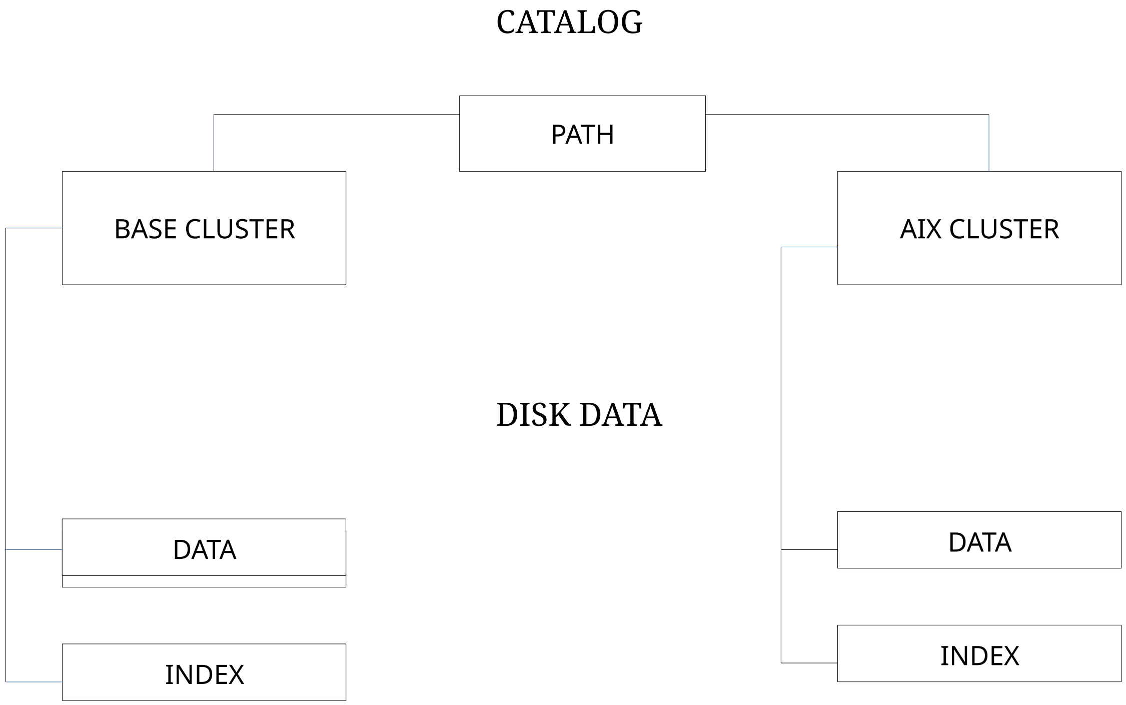
Control Intervals contain records of fixed or variable sizes. The records within a CI are sorted in Key sequence. Each record is described by a record descriptor field (RDF) and each interval contains a CI descriptor field.

A Control Area (CA)



- The Control Interval (CI) is the unit of data that is transferred between the DASD and virtual storage.
- CI sizes are multiples of 2K with 4K being a common choice.
- CIs can be constructed with a certain percentage of free space.
- CAs can be constructed with a certain percentage of free CIs.
- Clusters can be constructed with a certain percentage of free CAs.

VSAM manages data sets dynamically by storing information in each CI and CA. When a CI fills up, it is split into two CIs by copying approximately half the records into a free CI. This is called a *CI split.* When a CA fills up, it is split into two CAs by copying records into a free CA. This is called a *CA split*. VSAM tries to keep records that are logically close together based on keys, physically close on the DASD.



Defining a KSDS Cluster

All VSAM data is assumed to be variable length and records are automatically blocked into CIs. VSAM has it own data space and a catalog to manage it. We will use the Access Method Service (AMS) utility to define a VSAM cluster. Below is the JCL I used to create a KSDS cluster with 80-byte records RECSZ(80,80) (min,max), with a 5-byte key starting in the first byte of the record (KEY(5,0), a CI size of 4k, an initial storage size of a single track with room for sixteen more extents (tracks). The cluster name is KC2486.KSDS.DATASET. The data component of the cluster is called KC2486.KSDS.DATASETDATA, and the index component is called KC2486.KSDS.DATASET.INDEX.

```
//KC02486A JOB (KC024861),'WOOLBRIGHT',REGION=0M,CLASS=A,MSGCLASS=H,
// NOTIFY=KC02486,MSGLEVEL=(1,1),TIME=(0,1)
//STEP01 EXEC  PGM=IDCAMS
//SYSIN  DD  *
   DELETE  KC02486.KSDS.DATASET
   DEFINE  CLUSTER                              -
       (NAME(KC02486.KSDS.DATASET)              -
        INDEXED                                 -
        KEYS(5 0)                               -
        RECSZ(80 80)                            -
        CISZ(4096)                              -
        SHR(3 3)                                -
        TRK(1 1))                               -
      DATA                                      -
       (NAME(KC02486.KSDS.DATASET.DATA))        -
      INDEX                                     -
       (NAME(KC02486.KSDS.DATASET.INDEX))
/*
//SYSOUT   DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
```

Allocation of the dataset, defined above, is usually done in a separate job step before the cluster is loaded with records. It is a common practice to Delete and then Define the cluster to make sure the clusteris empty of records. The JCL above does that. After the cluster is defined, it is possible to start adding records. The first record added to a VSAM cluster must be added sequentially. Once the cluster has at least one record, it can be processed randomly with a key.

Here are some of the parameters and their meaning in the JCL above:

```
INDEXED—A KSDS
NOINDEXED—An ESDS
NUMBERED—Am RRDS
KEYS(Key Length, Key Offset)
CISZ(size)—The CI size
FREESPACE(CI, CA)—The percentage of freespace in each element
```

Access Method Services (AMS) is an all-purpose utility that provides the following options among others:

```
DEFINE CLUSTER—Creates an empty cluster
PRINT—Prints the records in a cluster
REPRO—Reproduces the records in a cluster
LISTCAT—Lists information if a VSAM catalog
DELETE—Deletes a cluster
DEFINE ALTERNATEINDEX—Deletes an Alternate Index into a cluster
DEFINE PATH—Creates a path for a cluster
BLDINDEX—Builds the index of a cluster
```

Printing a Dataset

The following JCL snippet invokes IDCAMS to print the file specified as IFILE in a DUMP format. Two other formats are CHAR and HEX. This is a helpful utility because on some systems, you will not be able to view a VSAM dataset directly.

```
//PRINT    EXEC PGM=IDCAMS
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
           PRINT  INFILE(IFILE)  -
           DUMP
/*
//IFILE    DD  DSN=KC02486.PAYROLL.MASTER,DISP=SHR
//
```

Copying a Dataset

This JCL snippet invoke IDCAMS to copy FILEIN to FILEOUT.

```
//REPRO    EXEC PGM=IDCAMS
//FILEIN   DD  DSN=TSYSAD2.PGM1.RESULTS,DISP=SHR
//FILEOUT  DD  DSN=TSYSAD2.I10.PG1.RESULTS,DISP=(NEW,CATLG,DELETE),
//       UNIT=SYSDA,DCB=(RECFM=FB,LRECL=80),
//       SPACE=(TRK,(1,1),RLSE)
//SYSIN    DD  *
              INFILE(FILEIN)  -
              OUTFILE(FILEOUT)
/*
//AMSDUMP  DD  SYSOUT=*
//
```

VSAM Assembler Macros

Macros are provided to work with VSAM data sets. There are two types of macros that are used to process VSAM data sets, Control Block and Action Request.

1) Control Block Macros—These macros are not executed, but generate storage that describes a VSAM data set and its processing options. Some examples:

- **ACB** (Access Method Control Block) - used to create the control block that represents the VSAM file
- **RPL** (Request Parameter List) – used to create a control block that describes a type of request that will be made on the file
- **EXLST** (Exit List) – used to used to specify the addresses of special processing routines (like end of file)

Here is an example of how these three macros might be coded in an assembler program,

```
FILEOUT  ACB   AM=VSAM,
               MACRF=(KEY,SEQ,RST,OUT),
               DDNAME=FILEOUT,
               EXLST=FILEOEX
WRTREQ   RPL   AM=VSAM,
               ACB=FILEOUT,
               AREA=RECOUT,
               RECLEN=80,
               OPTCD=(KEY,SEQ,NUP,MVE)
FILEOEX  EXLST AM=VSAM,
               SYNAD=SYNADERR,
               LERAD=LOGICERR
```

In this example, we use the Access Control Block (ACB) to describe a KSDS called FILEOUT which we will process sequentially for output. The ACB references two error routines in the EXLST control block: SYNADERR and LOGICERR. The Request Parameter List (RPL) references the ACB and describes a write request in Move mode from an 80-byte record area called RECOUT. We will write the records sequentially without update.

2) Action Request Macros—These macros (including OPEN, POINT, GET, PUT, ERASE, MODCB and CLOSE) are executable and are used to retrieve, update, delete, or insert VSAM records.

VSAM Request Error Strategy

After each VSAM executable operation, VSAM provides a return code in register 15. The return code should be tested after each operation to insure the program is proceeding normally. As always, a return code of 0 represents a normal execution of the macro. Here is an example:

```
         OPEN  (FILEOUT,(OUTPUT))     A VSAM OUTPUT FILE
         LTR   R15,R15                DID THE KSDS OPEN?
         JNZ   BADOPEN                IF NOT, QUIT
         ...

BADOPEN  DS   0H   ERROR ROUTINE
```

Unlike QSAM, which would abend on a bad OPEN, VSAM depends on the programmer checking the return code and making a decision to continue execution or not.

Control Block Macros

ACB Macro Parameter Details

VSAM is a very complex access method that allows much flexility and control over dataset management. In this chapter, we discuss some of the more common parameters and features. Consult the *z/OS DFSMS Macro Instructions for Datasets* manual for many more options and details.

AM=VSAM This parameter can be omitted since it is the default.
DDNAME= The JCL Ddname.
EXLIST= A reference to the EXLST address.
MACRF= A listing of all the types of programming access needed by the program.

- **KEY** – Keyed access to a KSDS of RRDS
- **DIR** – Direct processing
- **SEQ** – Sequential processing
- **SKP** – Skip sequential processing

IN - Retrieve records

OUT - Store records

RST - Data set is reusable (high RBA reset to 0 on open)

STRNO= The number of record pointers needed by the program for concurrent data set positioning.

RPL Macro Parameter Details

AM=VSAM This parameter can be omitted since it is the default

ACB= The ACB label for this RPL

AREA= The physical location of records during GETs and PUTs. In locate mode this is a fullword with the address of the record.

AREALEN= The length of the AREA during GETs and PUTs. This number is 4 for locate mode.

ARG= The address of a field that contains the search argument (key) for direct, skip-sequential, and positioning.

KEYLEN= The length of the ARG field.**OPTCD=** A list of sub-parameters that control the request.

DIR - Direct access

SEQ - Sequential access

SKP - Skip-sequential access

FWD - Retrieve records in a forward direction

BWD - Retrieve records in a backward direction

NUP - Only for adds and read-only requests

UPD - Read for update

KEQ - Exact match must occur for a direct GET, and a skip-sequential GET or POINT

KGE - A record with a key greater than or equal to the ARG value will be returned after a direct or skip sequential GET, or an equivalent record pointer after a POINT

FKS - Use the key length in the VSAM catalog in place of KEYLEN for KEQ and KGE comparisons

GEN - Use the KEYLEN value in place of the catalog value for KEQ and KGE comparisons. KEYLEN can only be equal or shorter than the catalog value. This type of search uses only partial keys.

LOC - Locate mode I/O

MVE - Move mode I/O

EXLST Macro Parameter Details

AM=VSAM This parameter can be omitted since it is the default
EODAD= The label where control is returned when the end of data occurs
SYNAD= The label where control is returned after a physical error occurs
LERAD= The label where control is returned after a logical error occurs

Action Request Macros

OPEN MACRO Details

USE the ACB name to open the file. VSAM ignores INPUT and OUTPUT options on the OPEN. After every OPEN, check the return code in register 15. If it is not 0, you should investigate the reason. Here is an example,

```
         OPEN  MYACB
         LTR   R15,R15
         JNE   ERROR
          ...
ERROR    EQU   *
```

CLOSE MACRO Details

Closes the named file. Check the return code in R15 to see if the dataset was properly closed.

```
         CLOSE MYACB
         LTR   R15,R15
         JNE   ERROR
          ...
ERROR    EQU  *
```

GET MACRO Details

Retrieves a record from a VSAM dataset. The RPL references the ACB name and qualifies the kind of request. Test the return code to see if the operation was successful.

In MVE mode, the record will be in the AREA. In LOC mode, the AREA will contain A fullword pointer.

```
         GET   RPL=MYRPL
         LTR   R15,R15
         JNE   ERROR
          ...
ERROR    EQU  *
```

PUT MACRO Details

This macro writes a record to a dataset. If the dataset is empty an KSDS, the first record must be added sequentially (**SEQ**). When executing PUT, the record is moved from the AREA parameter if the request is for MVE mode. If in LOC mode, the AREA fullword contains the address of the record we are writing. PUT releases the record to be written on DASD. CLOSE flushes any records remaining in system buffers.

```
         PUT RPL=MYRPL
         LTR   R15,R15
         JNE   ERROR
          ...
ERROR    EQU   *
```

POINT MACRO Details

This macro sets an internal next-record pointer based on the ARG value. POINT is the basis for skip-sequential processing. Using POINT, you can read forward sequentially, stop, move forward with POINT, and then continue reading sequentially. This is called skip-sequential processing. Using POINT, you can dynamically move the next record pointer forward or backward.

```
       (Set the ARG value)
         POINT RPL=MYRPL
         LTR   R15,R15
         JNE   ERROR
          ...
ERROR    EQU  *
```

ERASE MACRO Details

This macro sets deletes a record from a dataset. Before issuing the ERASE, the record must have been retrieved for update (UPD) with a GET operation.

```
         ERASE RPL=MYRPL
         LTR   R15,R15
         JNE   ERROR
          ...
ERROR    EQU  *
```

SHOWCB MACRO Details

This macro is used to display fields in an ACB by moving them to an addressable location. The code below names FWD as the local field where the data from the ACB is moved. LENGTH is the length of the area where the fields are returned. LRECL is the length of records in the data component of the KSDS (maximum length for variable-length records). LRECL is the data field moved from the ACB to a local storage field, FWD.

```
  SHOWCB  RPL=MYRPL,AREA=FWD,    MOVE RECLEN TO FWD                X
               LENGTH=4,                                           X
               FIELDS=(LRECL)
```

Sequentially Writing to a KSDS (Move Mode)

To write to a dataset, invoke a PUT macro that references an RPL (the request parameter list that describes the operation). Here is an example following the control block macros that describe the dataset.

```
         MVC    RECAREAM,SOMEDATA
         PUT    RPL=WRTREQ
         LTR    R15,R15
         JNZ    BADWRT
         . . .

FILEOUT  ACB   AM=VSAM,
               MACRF=(KEY,SEQ,RST,OUT),
               DDNAME=FILEOUT,
               EXLST=FILEOEX
WRTREQ   RPL   AM=VSAM,
               ACB=FILEOUT,
               AREA=RECOUT,
               RECLEN=80,
               OPTCD=(KEY,SEQ,NUP,MVE)
FILEOEX  EXLST AM=VSAM,
               SYNAD=SYNADERR,
               LERAD=LOGICERR
```

Sequentially Writing to a KSDS (Locate Mode)

To write to a dataset, invoke a PUT macro that references an RPL (the request parameter list that describes the operation). Here is an example following the control block macros that describe the dataset.

```
         PUT    RPL=WRTREQ          LOCATE MODE PUT
         LTR    R15,R15             OPERATION OK?
         JNZ    BADWRT              NO, BRANCH TO ERROR ROUTINE
         L      R9,RECADR           YES, R9 HAS ADR OF OUPUT BUFF
         MVC    0(L’DATA1,R9),DATA1 MOVE THE DATA TO OUTPUT BUFF
         . . .

FILEOUT  ACB   AM=VSAM,
               MACRF=(KEY,SEQ,RST,OUT),
               DDNAME=FILEOUT,
               EXLST=FILEOEX
WRTREQ   RPL   AM=VSAM,
               ACB=FILEOUT,
               AREA=RECADR,
               AREALEN=4,
               OPTCD=(KEY,SEQ,NUP,LOC)
FILEOEX  EXLST AM=VSAM,
               SYNAD=SYNADERR,
               LERAD=LOGICERR
REDADR   DS    F              THE RECORD ADDRESS
DATA1    DS    CL80           THE RECORD WE WANT TO WRITE
```

As with all locate mode PUTs, the PUT is issued to locate the output buffer and then the record is moved to the buffer.

Sequentially Reading a KSDS (Move Mode)

With Move Mode, the record is delivered directly in our program’s buffer, in this case, RECAREA. This is specified as the AREA in the RPL,

```
         GET    RPL=RDREQ           REQUEST THE NEXT REC
         LTR    R15,R15             DID THAT WORK?
         JNZ    BADGET              NO, TAKE THE BRANCH
         PUT    FILEOUT,RECAREA     YES, REC DELIVERED IN
         . . .
DONE     DS    0H                   EOD ROUTINE
         . . .
FILEIN   ACB    AM=VSAM,
                MACRF=(KEY,SEQ,IN),
                DDNAME=FILEIN,
                EXLST=FILEINEX
RDREQ    RPL    AM=VSAM,
                ACB=FILEIN,
                AREA=RECAREA,
                AREALEN=80,
                OPTCD=(KEY,SEQ,NUP,MVE)
FILEINEX EXLST  AM=VSAM,
                EODAD=DONE
                . . .
RECAREA  DS     CL80           THE RECORD WE WANT TO READ
```

Sequentially Reading a KSDS (Locate Mode)

After issuing a successful GET request, the address of the next record in the input buffer is stored in RECADR. Load that address and use a DSECT to address the contents.

```
         GET    RPL=RDREQ       READ A RECORD
         LTR    R15,R15         DID THAT WORK?
         JNZ    BADREAD         NO, TAKE THE BRANCH TO ERROR RTN
         L      R7,RECADR       YES, RECADR CONTAINS ADR OF A REC
         USING  RECSECT,R7      DROP THE DSECT ON THE REC
         ...
DONE     DS     0H
         ...
FILEIN   ACB    AM=VSAM,
                MACRF=(KEY,SEQ,IN),
                DDNAME=FILEIN,
                EXLST=FILEINEX
RDREQ    RPL    AM=VSAM,
                ACB=FILEIN,
                AREA=RECADR,
                AREALEN=4,
                OPTCD=(KEY,SEQ,NUP,LOC)
FILEinEX EXLST  AM=VSAM,
                EODAD=DONE
REDADR   DS     F              THE RECORD ADDRESS
DATA1    DS     CL80           THE RECORD WE WANT TO WRITE
```

Modifying a Control Block

VSAM is flexible and dynamic. We can create and modify control blocks dynamically, providing for explicit control of our I/O as the program proceeds. In this example, macro MODCB is used to supply an output buffer for the PUT by dynamically changing the RPL. Register 7 contains the address of the output buffer.

```
         LA     R7,DATA                POINT AT OUTPUT AREA
         MODCB  RPL=WRTREQ,AREA=(R7)   MODIFY RPL WITH THE AREA
         LTR    R15,R15                DID THAT WORK?
         JNZ    BADMOD                 NO, BRANCH
         PUT    RPL=WRTREQ             YES, PUT OUT THE REC
         LTR    R15,R15                DID THAT WORK?
         JNZ    BADPUT                 NO, TAKE THE BRANCH
         . . .                         YES, CONTINUE ON
FILEOUT  ACB   AM=VSAM,
               MACRF=(KEY,SEQ,RST,OUT),
               DDNAME=FILEOUT,
               EXLST=FILEOEX
WRTREQ   RPL   AM=VSAM,
               ACB=FILEOUT,
               AREALEN=80,
               OPTCD=(KEY,SEQ,NUP,MVE)
FILEOEX  EXLST AM=VSAM,
               SYNAD=SYNADERR,
               LERAD=LOGICERR
         . . .
DATA     DS    CL80           THE RECORD WE WANT TO WRITE
```

Keyed Direct Record Retrieval

One of the most important uses of KSDS datasets is the direct retrieval of a records using the record key. Here is an example:

```
         MVC    KEYAREA,=C’12345’      INIT THE KEY FIELD
         GET    RPL=RETRVE             READ A REC
         LTR    R15,R15                DID THAT WORK?
         JNZ    BADREAD                NO, TAKE THE BRANCH
         PUT    FILEOUT,INREC
         . . .
FILEIN   ACB    AM=VSAM,
                MACRF=(KEY,DIR,IN),
                DDNAME=FILEIN
RETRVE   RPL    AM=VSAM,
                ACB=FILEIN,
                AREA=INREC,
                AREALEN=80,
                OPTCD=(KEY,DIR,NUP,MVE),
                ARG=KEYAREA,
                KEYLEN=5
INREC    DS     CL80
KEYAREA  DS     CL5
```

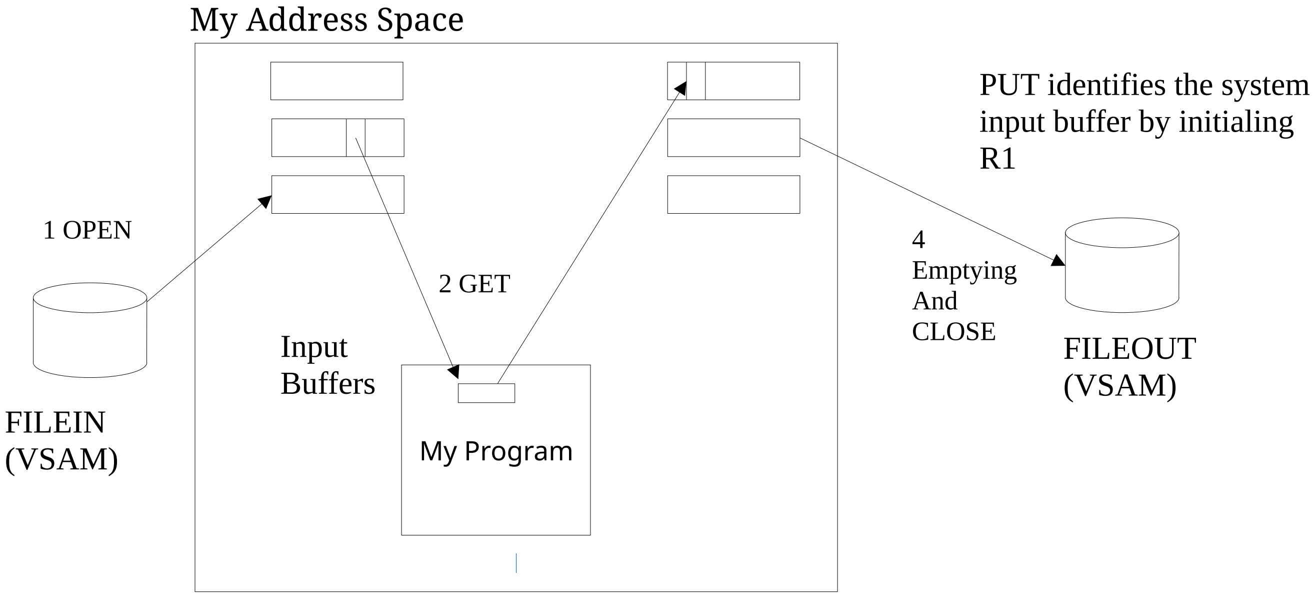
We move the key to KEYAREA and retrieve the record directly from the KSDS using the KSDS index to find it effiiciently.

Reading and Writing Records Efficiently

Move mode I/O is more expensive in terms of CPU cycles than Locate mode. Depending on several factors, it is possible to cut processing times in half by changing to Locate mode for I/O bound jobs. This applies to QSAM and VSAM applications.

Why Move Mode for Input and Output is Expensive

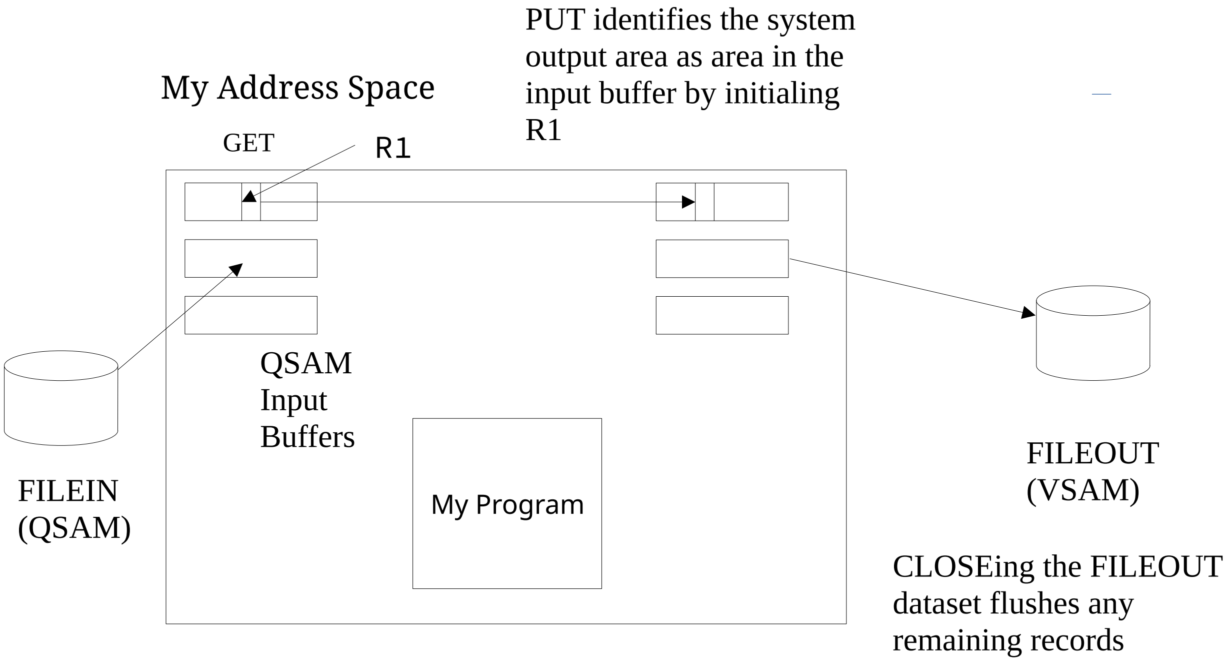
When an OPEN is issued in move mode, the operating system begins to read data into the system input buffers (1). A GET (2) operation moves a single record to local storage inside the program. A PUT (3) operation moves the record to an output system buffer. The operating system constantly empties the system buffers by writing data to DASD and flushes any remaining records when a CLOSE is executed (4). The important point is that records are moved at least four times. For large files with large record sizes, this movement is significant. By reducing the movement of data, we can decrease run times.



In the four input/output scenarios that follow, we illustrate the flexibility of VSAM to reduce the movement of data.

1) Reading QSAM Fixed Blocked and Writing VSAM

In the diagram below, VSAM dataset FILEOUT is defined as using Move mode I/O. Even so, the MODCB macro dynamically changes the output area of FILEOUT to be an input buffer record. As a result, input records are moved directly from system input buffers to the output system buffers. Records can be processed in the system buffers directly using a DSECT. For large files with large record sizes, these savings are significant. The operations are depicted in the address space depicted below.



PUT identifies the system output area as area in the input buffer by initialing R1

CLOSEing the FILEOUT dataset flushes any remaining records

The code that follows is a summary of the VSAM parts of the program illustrated above. At a minimum, addressability has to be established, the DSECT defined, any processing of the input records added, appropriate error routines included, and proper linkage code written. The VSAM cluster has to be defined using IDCAMS before running the program.

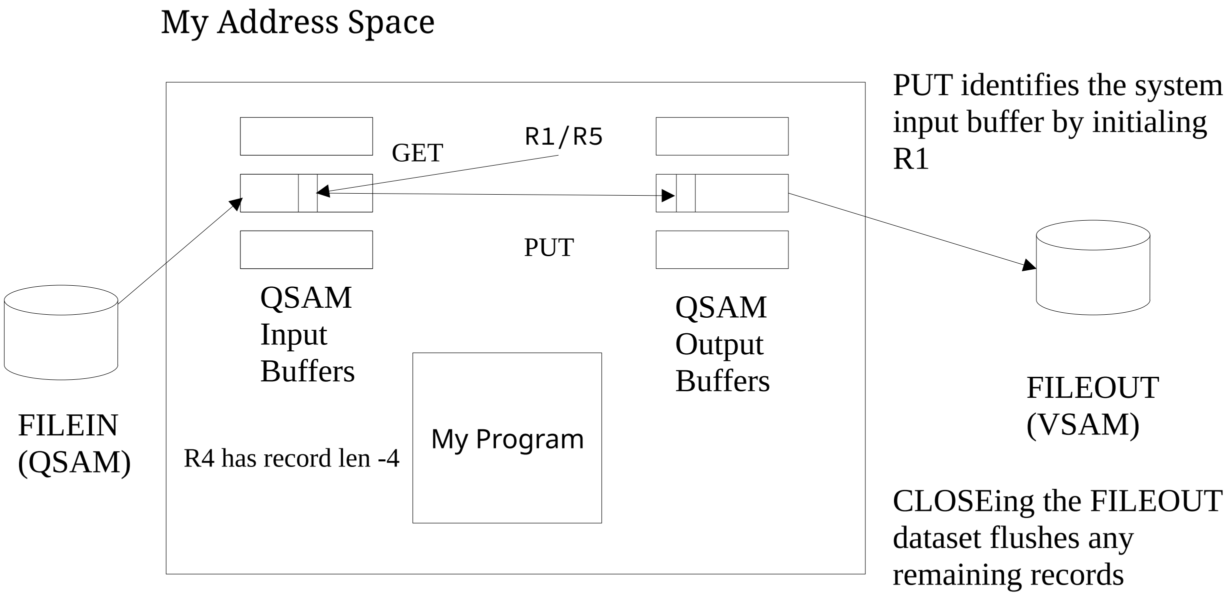
```
RECIN    DSECT
         . . .
RLEN     EQU         *-RECIN
MYPROG   CSECT
         ...
         OPEN    (FILEIN,(INPUT))       QSAM INPUT
         OPEN    (FILEOUT,(OUTPUT))     VSAM OUTPUT
         LTR     R15,R15                DID THAT OPEN?
         JNZ     ERR                    NO, TAKE THE BRANCH
RECLOOP  DS     0H                      YES, WE ARE GOOD
         GET     FILEIN                 LOCATE QSAM MODE READ
         LR      R5,R1                  NEXT REC IN R1
         USING   RECSECT,R5             DROP THE DSECT ON THE REC
         ...  (MODIFY THE RECORD HERE – DON’T CHANGE THE KEY!!)
         MODCB   RPL=MYRPL,AREA=(R5)    CHANGE OUTPUT AREA FOR VSAM TO INPUT BUFFER
         LTR     R15,R15                WAS THAT GOOD?
         JNZ     ERR                    NO, TAKE THE BRANCH
         PUT     RPL=MYRPL              YES, RELEASE THE REC FOR PRINTING
         LTR     R15,R15                WAS THAT GOOD?
         JNZ     ERR                    NO, TAKE THE BRANCH
         DROP    R5                     DONE WITH THE DSECT
         B       RECLOOP                KEEP LOOPING
EOF      DS     0H  ...
         CLOSE   FILEIN                 QSAM FILE CLOSE
         CLOSE   FILEOUT                VSAM FILE CLOSE
         LTR     R15,R15                DID THAT CLOSE?
         JNZ     ERR                    NO, JUMP TO ERROR RTN
         . . .
FILEIN   DCB     DSORG=PS,DEVD=DA,DDNAME=FILEIN,MACRF=GL,EODAD=EOF

FILEOUT  ACB    AM=VSAM,DDNAME=FILEOUT,EXLST=MYEXLST,MACRF=(KEY,SEQ,RST,OUT)

MYRPL    RPL     AM=VSAM,ACB=FILEOUT,RECLEN=RLEN,OPTCD=(KEY,SEQ,NUP,MVE)


MYEXLST  EXLST   AM=VSAM,SYNAD=MYSYNAD,LERAD=MYLERAD

ERR      DS     0H  ...
MYSYNAD  DS     0H  ...
MYLERAD  DS     0H  ...
         END     MYPROG
```

2) Reading QSAM Variable Blocked and Writing VSAM

In this scenario, each variable length input record begins with a 4-byte record descriptor word (RDW). The RDW has a 2-byte length followed by two bytes of zeroes (LLZZ). We use the same technique as in the first example to change the output AREA parameter to point at input buffer data. The RECLEN is also modified for the move mode PUT. Here is a sketch of the VSAM code.

```
RECSECT  DSECT
LLZZ     DS      F                      LENGTH IN LAST TWO BYTES
         . . .
RLEN     EQU     *-RECIN
MYPROG   CSECT
         ...
         OPEN    (FILEIN,(INPUT))       QSAM INPUT
         OPEN    (FILEOUT,(OUTPUT)      VSAM OUTPUT
         LTR     R15,R15                DID THAT OPEN?
         JNZ     ERR                    NO, TAKE THE BRANCH
RECLOOP  DS     0H                      YES, WE ARE GOOD
         GET     FILEIN                 LOCATE MODE READ
         LR      R5,R1                  NEXT REC IN R1 (LOCATE MODE)
         USING   RECSECT,R5             DROP THE DSECT ON THE REC
         ...  (Modify the input record here if needed)
         SR      R4,R4                  ZERO THE REG
         ICM     R4,B’0011’,LLZZ        GRAB LEN FROM LLZZ FOR R4
         SH      R4,=H’4’               ADJUST FOR RDW LEN (DATA LEN LEFT NOW)
         LA      R5,4(R5)               BUMP R5 PAST RDW
         MODCB   RPL=MYRPL,RECLEN=(R4),AREA=(R5)     CHANGE RECLEN AND AREA
         LTR     R15,R15                WAS THAT GOOD?
         JNZ     ERR                    NO, TAKE THE BRANCH
         PUT     RPL=MYRPL              PUT THE RECORD
         DROP    R5                     DONE WITH DSECT
         B       RECLOOP                KEEP LOOPING
EOF      DS     0H  ...
         CLOSE   FILEIN                 QSAM FILE CLOSE
         CLOSE   FILEOUT                VSAM FILE CLOSE
         LTR     R15,R15                DID THAT CLOSE?
         JNZ     ERR                    NO, JUMP TO ERROR RTN
         ...

FILEIN   DCB     DSORG=PS,DEVD=DA,DDNAME=FILEIN,MACRF=GL,EODAD=EOF
FILEOUT  ACB     AM=VSAM,DDNAME=FILEOUT,EXLST=MYEXLST,MACRF=(KEY,SEQ,RST,OUT)
MYRPL    RPL     AM=VSAM,ACB=FILEOUT,RECLEN=RLEN,OPTCD=(KEY,SEQ,NUP,MVE)
MYEXLST  EXLST   AM=VSAM,SYNAD=MYSYNAD,LERAD=MYLERAD
ERR      DS     0H  ...
MYSYNAD  DS     0H  ...
MYLERAD  DS     0H  ...
         END     MYPROG
```

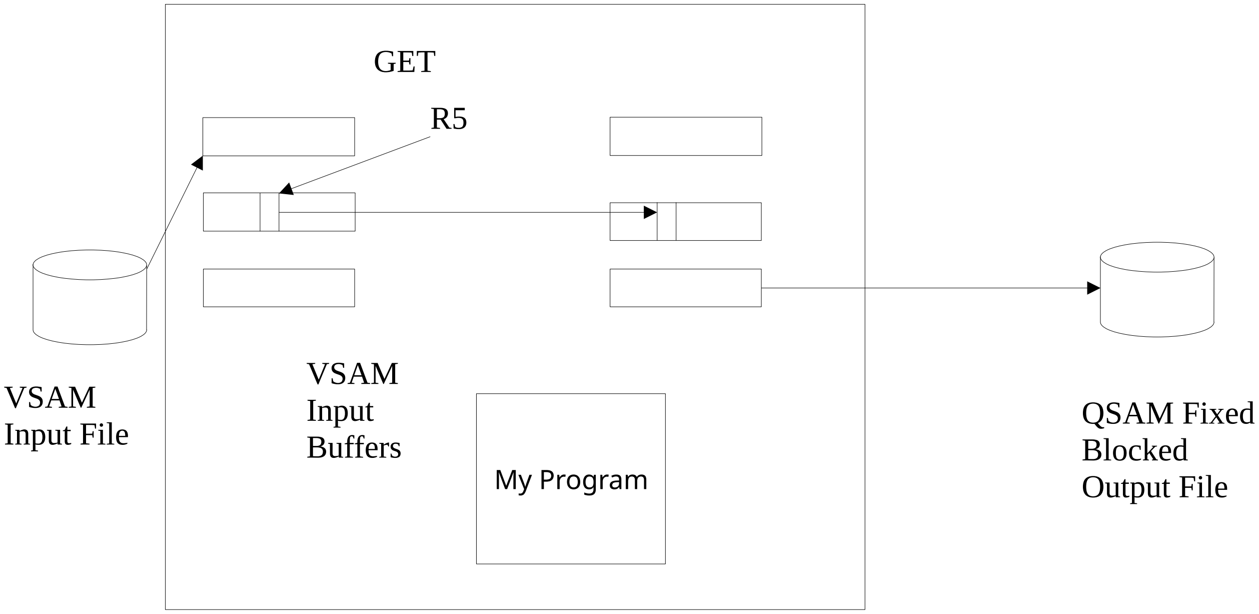
Although FILEOUT is defined as using Move mode I/O, the MODCB dynamically changes the output area to be inside an input buffer area. As a result, input data is moved directly from input buffers to output buffers. For large files with large record sizes, this savings is significant. This is pictured in the address space depicted below.



3) Reading VSAM and Writing QSAM Fixed Blocked

In this scenario, the VSAM input dataset is read in locate mode. The QSAM output file is written in move mode.

```
RECSECT  DSECT
         . . .
RLEN     EQU      *-RECIN
MYPROG   CSECT
         ...
         OPEN     (FILEIN,(INPUT))       VSAM INPUT
         LTR      R15,R15                DID THAT OPEN?
         JNZ      ERR                    NO, TAKE THE BRANCH
         OPEN     (FILEOUT,(OUTPUT)      QSAM OUTPUT
RECLOOP  DS      0H                      YES, WE ARE GOOD
         GET      RPL=MYRPL              GET VSAM REC IN LOC MODE
         L        R5,MYADDR              NEXT REC PTR IN MYADDR (LOCATE MODE)
         USING    RECSECT,R5             DROP THE DSECT ON THE REC
         ...  (Modify the input record here)
         PUT      FILEOUT,RECSECT        PUT THE RECORD
         DROP     R5
         B        RECLOOP                KEEP LOOPING
EOF      DS      0H  ...
         CLOSE    FILEIN                 VSAM FILE CLOSE
         LTR      R15,R15                DID THAT CLOSE?
         JNZ      ERR                    NO, JUMP TO ERROR RTN
         CLOSE    FILEOUT                QSAM FILE CLOSE
         ...

FILEOUT  DCB      DSORG=PS,DEVD=DA,DDNAME=FILEOUT,MACRF=PM

FILEIN   ACB      AM=VSAM,DDNAME=FILEIN,EXLST=MYEXLST,MACRF=(KEY,SEQ,IN)

MYRPL    RPL      AM=VSAM,ACB=FILEIN,AREA=MYADDR,AREALEN=4,OPTCD=(KEY,SEQ,NUP,LOC)

MYEXLST  EXLST    AM=VSAM,SYNAD=MYSYNAD,LERAD=MYLERAD

MYADDR   DS       F
ERR      DS      0H  ...
MYSYNAD  DS      0H  ...
MYLERAD  DS      0H  ...
         END      MYPROG
```

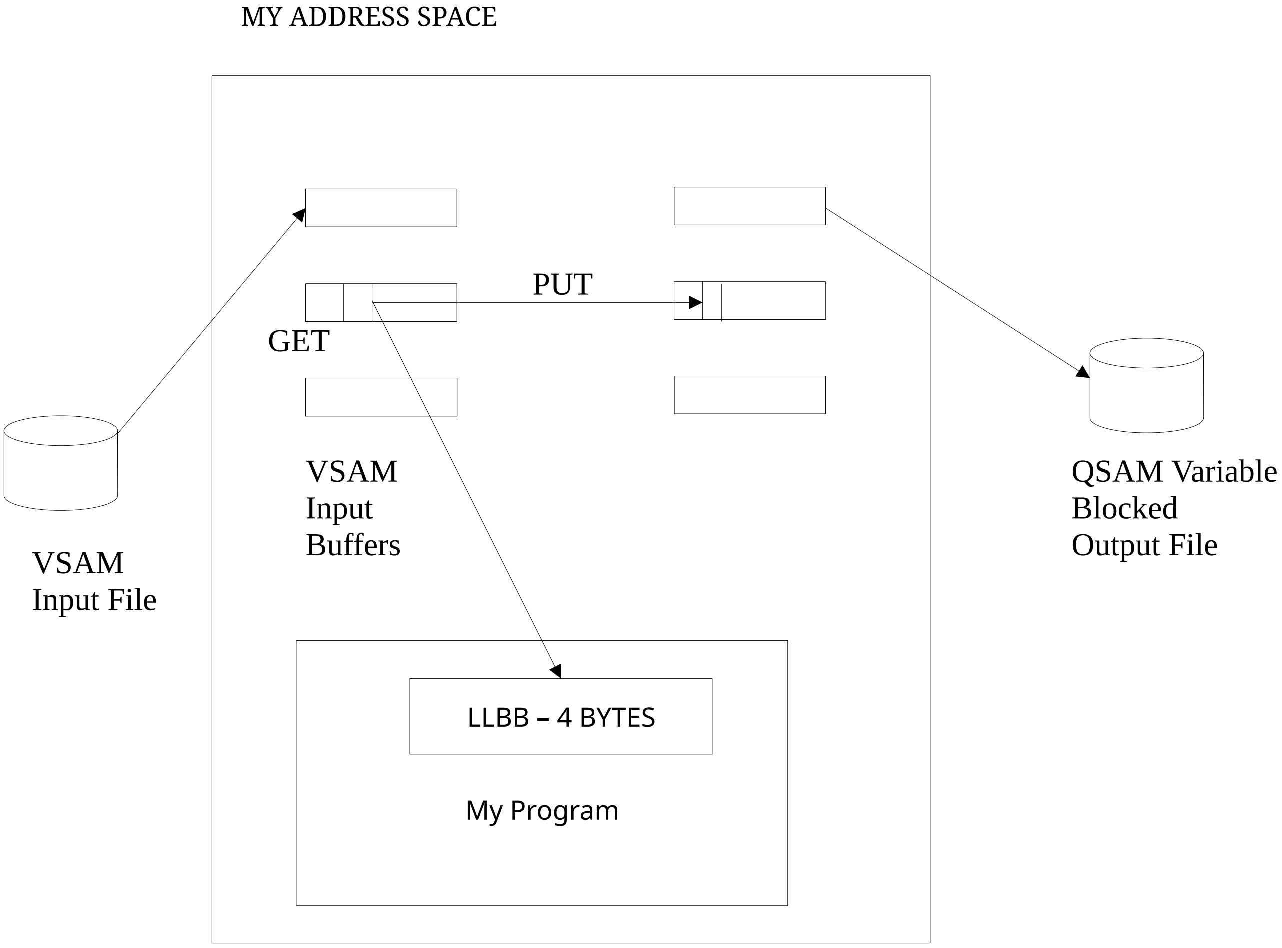


The VSAM input dataset is read in locate mode and R5 is loaded with the address of the record. We use a DSECT to address the input record. The PUT operation identifies a QSAM output buffer and moves the record to it. The operating system writes the output buffers into the the QSAM output file. The input and output buffers can contain one or more records.

4) Reading VSAM and Writing QSAM Variable Blocked

In the final scenario, the VSAM input dataset is read in locate mode. The QSAM output file is written in move mode.

```
MYPROG   CSECT
         ...
         OPEN    (FILEIN,(INPUT))       VSAM INPUT
         LTR     R15,R15                DID THAT OPEN?
         JNZ     ERR                    NO, TAKE THE BRANCH
         OPEN    (FILEOUT,(OUTPUT)      QSAM OUTPUT
RECLOOP  DS     0H                      YES, WE ARE GOOD
         GET     RPL=MYRPL              GET VSAM REC IN MOVE MODE
         SHOWCB  RPL=MYRPL,AREA=FWD,    MOVE RECLEN TO FWD                       X
               LENGTH=4,                                                         X
               FIELDS=(LRECL)
         L       R5,FWD                 PUT RECLEN IN REG 5
         LA      R4,4(R4,R0)            ADD LENGHT OF LLBB
         STCM    R4,B’0011’,LLBB        PUT LENGTH IN LLBB
         PUT     FILEOUT,LLBB           WRITE QSAM RECORD
         B       RECLOOP                KEEP LOOPING
EOF      DS     0H  ...
         CLOSE   FILEIN                 VSAM FILE CLOSE
         LTR     R15,R15                DID THAT CLOSE?
         JNZ     ERR                    NO, JUMP TO ERROR RTN
         CLOSE   FILEOUT                QSAM FILE CLOSE
         ...

FILEOUT  DCB     DSORG=PS,DEVD=DA,DDNAME=FILEOUT,MACRF=PM,

FILEIN   ACB     AM=VSAM,DDNAME=FILEIN,EXLST=MYEXLST,MACRF=(KEY,SEQ,IN)

MYRPL    RPL     AM=VSAM,ACB=FILEIN,AREA=MYBUFF,AREALEN=L’MYBUFF,               X
               OPTCD=(KEY,SEQ,NUP,MVE)

MYEXLST  EXLST   AM=VSAM,SYNAD=MYSYNAD,LERAD=MYLERAD,EODAD=EOF

LLBB     DS      F                      THE LENGTH OF THE VAR-LEN REC
MYBUFF   DS      CL500                  SHOULD BE MAX VSAM REC LEN
ERR      DS     0H  ...
MYSYNAD  DS     0H  ...
MYLERAD  DS     0H  ...
         END     MYPROG
```



While we are reading the VSAM file in move mode, only four bytes are being delivered to our local storage. The record is moved between input and output buffers.

Chapter Exercise

1) Use IDCAMS to delete and then define a cluster. Deleting a non-existent cluster produces an 8 return code. Running the same JCL again will return a 0 if the cluster is properly defined.

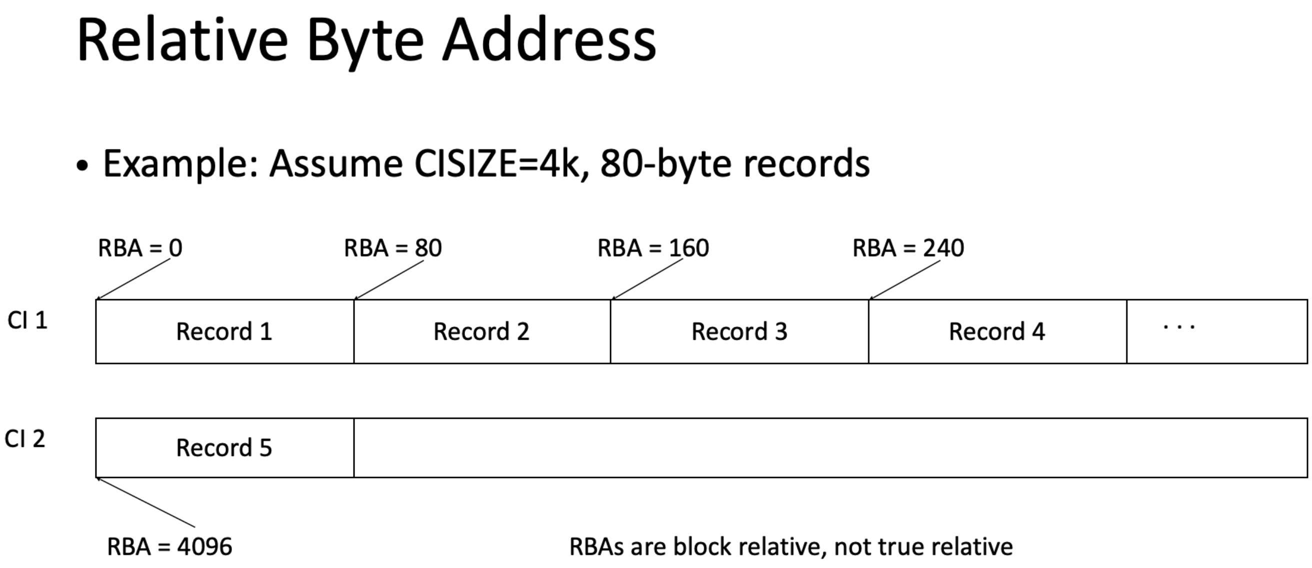
2) Write a program that reads a QSAM fixed blocked file and writes it to the VSAM cluster. You can use instream data that conforms to the definition of the VSAM cluster.

3) Use IDCAMS to print the records in the cluster and verify they are correct.

Digging Deeper

VSAM locates records by using three techniques.

1) Relative Byte Address (RBA) – Every byte of every record is sequentially numbered starting with the first byte which is numbered 0 and continuing forward to the last byte. Each offset to the first byte of a record is called the relative byte address or RBA. The first record has RBA = 0. The RBA of the second record is equal to the length of the first record. Using this numbering scheme, records can be retrieved by RBA. The RBA is volatile and RBAs change when records are added to r deleted from a cluster.



Relative Byte Address

- Example: Assume CISIZE=4k, 80-byte records

2) Relative Record Number (RRN) – RRNs are associated with Relative Record Datasets (RRDS). Records in an RRDS are numbered sequentially starting with RRN = 0 for the first record. The second record has RRN = 1 and so on. Records are stored in “slots” so the first record has RRN = 0 and is stored in slot 0, the second record has RRN = 1 and is stored in slot 1, and so on. Relative Record Datasets are used for simple tables similar to arrays.

3) Keys – Internal record keys are used to retrieve data in Key-sequenced Datasets (KSDS). The cluster consists of two parts: the data component and the index component. The index is a data structure that provides random and sequential access to the records in the data component. The key associated with the base cluster is the **primary key**. A KSDs can have other indexes with different organizations based on **secondary keys**. For every KSDS, each record must contain a unique, embedded, fixed-length key in the same position of each record – the primary key. Alternate index keys don’t have to be unique. Key sizes range from 1 to 255 bytes.

Generic Keys

A generic key is the high-order portion of a regular key. For example, a credit card company could maintain records using a full key that consists of a bank number followed by a customer number.

Bank Number	Customer Number

In this case we could use the bank number as a generic key. Using the generic key, we could move to the first customer in a given bank to begin processing records.

Things To Remember About a KSDS

- The most heavily used type of VSAM dataset
- Allocated with IDCAMS DEFINE - parameter INDEXED
- Records are initially added sequentially to an empty cluster in key sequence
- Records can be added, deleted and updated (primary keys can’t be altered)
- Records can be processed sequentially, skip-sequentially, or directly (by key or RBA),

Things To Remember About an ESDS

- Created with IDCAMS DEFINE – parameter NONINDEXED
- Records are sequenced by order of entry
- New records are added at the end
- Records can be updated, but the length can’t change
- Records can’t be physically deleted (use a flag to mark a record invalid)
- Records are usually accessed sequentially , but can also be accessed directly (by RBA)
- RBA positioning is allowed before sequential processing, but no skip-sequential processing can occur
- When a record is loaded or added, VSAM returns the RBA

Things To Remember About a Fixed-Length RRDS

- Defined using IDCAMS DEFINE – parameter NUMBERED
- The file consists of a collection of pre-formatted, fixed-length, logical record slots
- Each slot has a unique RRN
- Each logical record occupies a slot and is stored and retrieved by RRN
- Slots can be empty or occupied
- The cluster can be processed sequentially, directly, or skip sequentially
- Processing by RBA or key is not supported
- When processing sequentially, empty slots are skipped
- Typically, a RRDS is processed directly using the RRN. Good for small tables

When Things Go Wrong

- VSAM puts a non-zero return code in R15
- Return codes are paired with reason codes that are set in the access method control block (ACB) and the request parameter list (RPL).
- Reason codes that are set in the ACB indicate open or close errors.
- Reason codes that are set in the RPL indicate record management errors.
- Use SHOWCB to get access to the reason codes
- Look up the reason codes *z/OS DFSMS Macro Instructions for Data Sets*