Chapter 28: Virtual Storage Access Method (VSAM)

Author's Note: *This chapter is copied almost verbatim from the material in Chapter 19 of the textbook by Peter Abel. It is used by permission.*

Virtual **S**torage **A**ccess **M**ethod (VSAM) is a relatively recent file organization method for users of IBM OS/VS and DOS/VS. VSAM facilitates both sequential and random processing and supplies a number of useful utility programs.

The term ***file*** is somewhat ambiguous since it may reference an I/O device or the records that the device processes. To distinguish a collection of records, IBM OS literature uses the term ***data set***. VSAM provides three types of data sets:

1. Key-sequenced Data Set (KSDS). KSDS maintains records in sequence of key, such as employee or part number, and is equivalent to indexed sequential access method (ISAM).

2. Entry-sequenced Data Set (ESDS). ESDS maintains records in the sequence in which they were initially entered and is equivalent to sequential organization.

3. Relative-Record Data Set (RRDS). RRDS maintains records in order of relative record number and is equivalent to direct file organization.

Both OS/VS and DOS/VS handle VSAM the same way and use similar support programs and macros, although OS has a number of extended features.

Thorough coverage of assembler VSAM would require an entire textbook. However, this chapter supplies enough information to enable you to code programs that create, retrieve, and update a VSAM data set. For complete details, see the IBM Access Methods Services manual and the IBM DOS/VSE Macros or OS/VS Supervisor Services manuals.

In the table below, **RBA** stands for "**Relative Byte Address**", the byte's displacement from the start of the data set.

Feature	**Key–Sequenced**	**Entry–Sequenced**	**Relative–Record**
	KSDS	**ESDS**	**RRDS**
Record sequence	By key	In sequence in which entered	In sequence of relative record number
Record length	Fixed or variable	Fixed or variable	Fixed only
Access of records	By key via index or RBA	By RBA	By relative record number
Change of address	Can change record RBA	Cannot change record RBA	Cannot change relative record number
New records	Distributed free space for records	Space at end of data set.	Empty slot in data set.
Recovery of space	Reclaim space if record is deleted.	No delete, but can overwrite old record.	Can reuse deleted space.

Figure 28–1 Features of VSAM organization methods

CONTROL INTERVALS

For all three types of data sets, VSAM stores records in groups (one or more) of control intervals. You may select the control interval size, but if you allow VSAM to do so, it optimizes the size based on the record length and the type, of disk device being used. The maximum size of a control interval is 32,768 bytes. At the end of each control interval is control information that describes the data records:

Rec–1	Rec–2	Rec–3	….	Control Information

A control interval contains one or more data records, and a specified number of control intervals comprise a control area. VSAM addresses a data record by relative byte address (RBA)-its displacement from the start of the data set. Consequently, the first record of a data set is at RBA 0, and if records are 500 bytes long, the second record is at RBA 500.

The list in Fig. 28–1 compares the three types of VSAM organizations.

ACCESS METHOD SERVICES (AMS)

Before physically writing (or "loading") records in a VSAM data set, you first catalog its structure. The IBM utility package, **A**ccess **M**ethod **S**ervices (AMS), enables you to furnish VSAM with such details about the data set as its name, organization type, record length, key location, and password (if any). Since VSAM subsequently knows the physical characteristics of the data set, your program need not supply as much detailed information as would a program accessing an ISAM file.

The following describes the more important features of AMS. Full details are in the IBM OS/VS and DOS/VS Access Methods Services manual. You catalog a VSAM structure using an AMS program named IDCAMS, as follows:

OS: `//STEP EXEC PGM=IDCAMS`

DOS: `//EXEC IDCAMS,SIZE=AUTO`

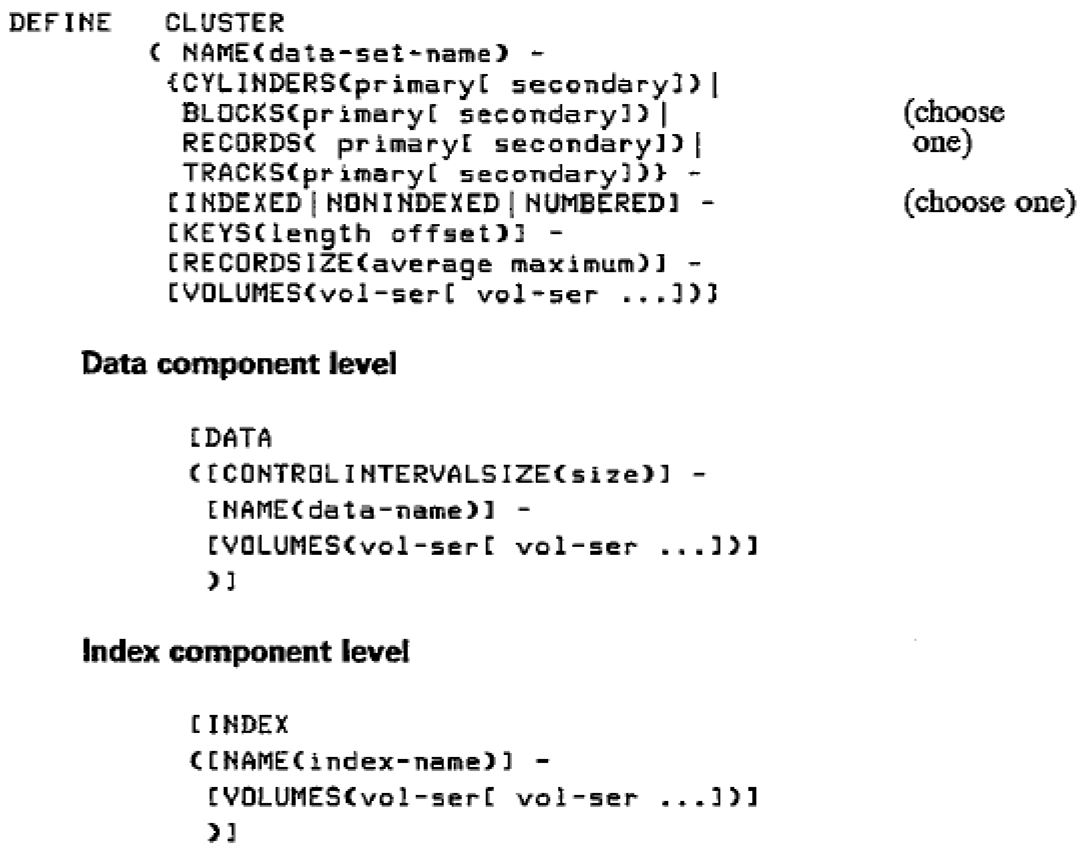
Immediately following the command are various entries that DEFINE the data set. The first group under CLUSTER provides required and optional entries that describe all the information that VSAM must maintain for the data set. The second group, DATA, creates an entry in the catalog for a data component, that is, the set of all control area and intervals for the storage of records. The third group, INDEX, creates an entry in the catalog for a KSDS index component for the handling of the KSDS indexes.

Figure 28–2 provides the most common DEFINE CLUSTER entries. Note that to indicate continuation, a hyphen (–) follows every entry except the last. The following notes apply to the figure.

Note:

SYMBOL	**MEANING**
[ ]	Optional entry, may be omitted
{ }	Select one of the following options
( )	You must code these parentheses
|	"or", indicates one of the choices listed in the brackets {A | B} means to select either A or B

Cluster Level



```
DEFINE   CLUSTER
       ( NAME(data-set-name) -
        {CYLINDERS(primary[ secondary])|
         BLOCKS(primary[ secondary])|                (choose
         RECORDS( primary[ secondary])|               one)
         TRACKS(primary[ secondary])} -
        [INDEXED|NONINDEXED|NUMBERED] -              (choose one)
        [KEYS(length offset)] -
        [RECORDSIZE(average maximum)] -
        [VOLUMES(vol-ser[ vol-ser ...])]
```

Data component level

```
        [DATA
        ([CONTROLINTERVALSIZE(size)] -
         [NAME(data-name)] -
         [VOLUMES(vol-ser[ vol-ser ...])]
         )]
```

Index component level

```
        [INDEX
        ([NAME(index-name)] -
         [VOLUMES(vol-ser[ vol-ser ...])]
         )]
```

Figure 28–2 Entries for defining a VSAM data set

- DEFINE CLUSTER (abbreviated DEF CL) provides various parameters all contained within parentheses.
- NAME is a required parameter that supplies the name of the data set. You can code the name up to 44 characters w**ith a period after each 8 or fewer characters, as `EMPLOYEE.RECORDS.P030`**. The name corresponds to job control, as follows:

OS: **`//FILEVS DD DSNAME=EMPLOYEE.RECORDS.P030`** ...

DOS: **`// DLBL FILEVS,'EMPLOYEE.RECDRDS.P030',0,VSAM`**

The name FILEVS in this example is whatever name you assign to the file definition (ACB) in your program, such as

```
filename ACB DDNAME=FILEVS
```

- BLOCKS. You may want to load the data set on an FBA device (such as 3310 or 3370) or on a CKD device (such as 3350 or 3380). For FBA devices, allocate the number of 512–byte BLOCKS for the data set. For CKD devices, the entry CYLINDERS (or CYL) or TRACKS allocates space. The entry RECORDS allocates space for either FBA or CKD. In all cases, indicate a primary allocation for a generous expected amount of space and an optional secondary allocation for expansion if required.

- Choose one entry to designate the type of data set: INDEXED designates key–sequenced, NONINDEXED is entry–sequenced, and NUMBERED is relative–record.
- KEYS for INDEXED only defines the length (from 1 to 255) and position of the key in each record. For example, KEYS (6 0) indicates that the key is 6 bytes long beginning in position 0 (the first byte).
- RECORDSIZE (or RECSZ) provides the average and maximum lengths in bytes of data records. For fixed–length records and for RRDS, the two entries are identical. For example, code (120b120) for l20–byte records.
- VOLUMES (or VOL) identifies the volume serial number(s) of the DASD volume(s) where the data set is to reside. You may specify VOLUMES at any of the three levels; for example, the DATA and INDEX components may reside on different volumes.

DEFINE CLUSTER supplies a number of additional specialized options described in the IBM AMS manual.

ACCESSING AND PROCESSING

VSAM furnishes two types of accessing, keyed and addressed, and three types of processing, sequential, direct, and skip sequential. The following chart shows the legal accessing and processing by type of organization:

Type	**Keyed Access**	**Addressed Access**
KSDS	Sequential Direct Skip sequential	Sequential Direct
ESDS		Sequential Direct
RRDS	Sequential Direct Skip sequential	

In simple terms, ***keyed accessing*** is concerned with the key (for KSDS) and relative record number (for RRDS). For example, if you read a KSDS sequentially, VSAM delivers the records in sequence by key (although they may be in a different sequence physically).

Addressed accessing is concerned with the RBA. For example, you can access a record in an ESDS using the RBA by which it was stored. For either type of accessing method, you can process records sequentially or directly (and by skip sequential for keyed access). Thus you always use addressed accessing for ESDS and keyed accessing for RRDS and may process either type sequentially or directly. KSDS, by contrast, permits both keyed access (the normal) and addressed access, with both sequential and direct processing.

KEY–SEQUENCED DATA SETS

A key-sequenced data set (KSDS) is considerably more complex than either ESDS or RRDS but is more useful and versatile. You always create ("load") a KSDS in ascending sequence by key and may process a KSDS directly by key or sequentially. Since KSDS stores and retrieves records according to key, each key in the data set must be unique.

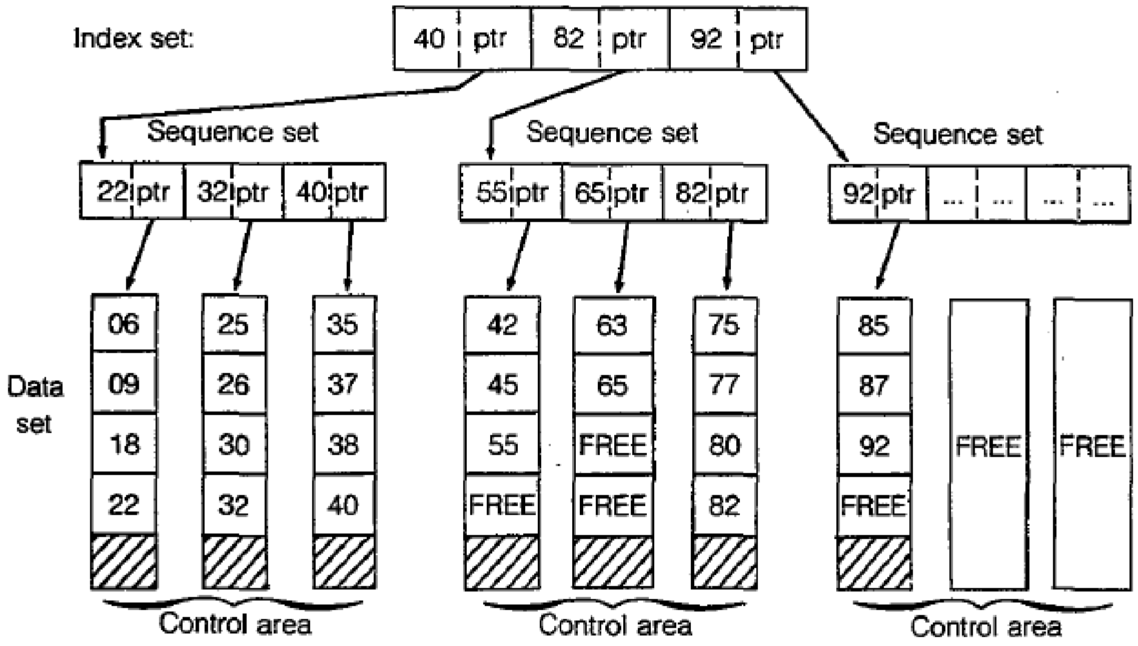


Figure 28–3 Key–sequenced organization

'Figure 28–3 provides a simplified view of a key-sequenced data set. The control intervals that contain the data records are depicted vertically, and for this example three control intervals comprise a control area. A *sequence set* contains an entry for each control interval in a control area. Entries within a sequence set consist of the highest key for each control interval and the address of the control interval; the address acts as a pointer to the beginning of the control interval. The highest keys for the first control area are 22, 32, and 40, respectively. VSAM stores each high key along with an address pointer in the sequence set for the first control area.

At a higher level, an *index set* (various levels depending on the size of the data set) contains high keys and address pointers for the sequence sets. In Fig. 28–3, the highest key for the first control area is 40. VSAM stores this value in the index set along with an address pointer for the first sequence.

When a program wants to access a record in the data set directly, VSAM locates the record first by means of the index set and then the sequence set. For example, a program requests access to a record with key 63. VSAM first checks the index set as follows:

RECORD KEY	INDEX SET	ACTION
63	40	Record key high, not in first control area.
63	82	Record key low, in second control area.

VSAM has determined that key 63 is in the second control area. It next examines the sequence set for the second control area to locate the correct control interval.
These are the steps:

RECORD KEY	SEQUENCE SET	ACTION
63	55	Record key high, not in first control interval.
63	65	Record key low, in second control interval.

VSAM has now determined that key 63 is in the second control interval of the second control area. The address pointer in the sequence set directs VSAM to the correct control interval. VSAM then reads the keys of the data set and locates key 63 as the first record that it delivers to the program.

Free Space
You normally allow a certain amount of free space in a data set for VSAM to insert new records. When creating a key–sequenced data set, you can tell VSAM to allocate free space in two ways:

1. Leave space at the end of each control interval.
2. Leave some control intervals vacant.

If a program deletes or shortens a record, VSAM reclaims the space by shifting to the left all following records in the control interval. If the program adds or lengthens a record, VSAM inserts the record in its correct space and moves to the right all following records in the control interval. VSAM updates RBAs and indexes accordingly.

A control interval may not contain enough space for an inserted record. In such a case, VSAM causes a ***control interval split*** by removing about half the records to a vacant control interval in the same control area. Although records are now no longer *physically* in key order, for VSAM they are *logically* in sequence. The updated sequence set controls the order for subsequent retrieval of records.

If there is no vacant control interval in a control area, VSAM causes a control area split, using free space outside the control area. Under normal conditions, such a split seldom occurs. To a large degree, a VSAM data set is self-organizing and requires reorganization less often than an ISAM file.

ENTRY·SEQUENCED DATA SETS
An entry-sequenced data set (ESDS) acts like sequential file organization but has the advantages of being under control of VSAM, some use of direct processing, and password facilities. Basically, the data set is in the sequence in which it is created, and you normally (but not necessarily) process from the start to the end of the data set. Sequential processing of an ESDS by RBA is known as addressed access, which is the method you use to create the data set. You may also process ESDS records directly by RBA. Since ESDS is not concerned with keys, the data set may legally contain duplicate records.

Assume an ESDS containing records with keys 001, 003, 004, and 006. The data set would appear as follows:

```
| 001 | 003 | 004 | 006 |
```

You may want to use ESDS for tables that are to load into programs, for small files that are always in ascending sequence, and for files extracted from a KSDS that are to be sorted.

RELATIVE-RECORD DATA SETS

A relative-record data set (RRDS) acts like direct file organization but also has the advantages of being under control of VSAM and offering keyed access and password facilities. Basically, records in the data set are located according to their keys. For example, a record with key 001 is in the first location, a record with key 003 is in the third location, and so forth. If there is no record with key 002, that location is empty, and you can subsequently insert the record.

Assume an RRDS containing records with keys 001, 003, 004, and 006. The data set would appear as follows:

```
| 001 | ... | 003 | 004 | ... | 006 |
```

Since RRDS stores and retrieves records according to key, each key in the data set must be unique.

You may want to use RRDS where you have a small to medium-sized file and keys are reasonably consecutive so that there are not large numbers of spaces. One example would be a data set with keys that are regions or states, and contents are product sales or population and demographic data.

You could also store keys after performing a computation on them. As a simple example, imagine a data set with keys 101, 103, 104, and 106. Rather than store them with those keys, you could subtract 100 from the key value and store the records with keys 001, 003, 004, and 006.

VSAM MACRO INSTRUCTIONS

VSAM uses a number of familiar macros as well as a few new ones to enable you to retrieve, add, change, and delete records. In the following list, for macros marked with an asterisk, see the IBM DOS/VS or OS/VS Supervisor and I/O Macros manual for details.

- To relate a program and the data:

ACB	(access method control block)
EXLST	(exit list)

- To connect and disconnect a program and a data set:

OPEN	(open a data set)
CLOSE	(close a data set)
TCLOSE	(temporary close)

- To define requests for accessing data:

RPL	(request parameter list)

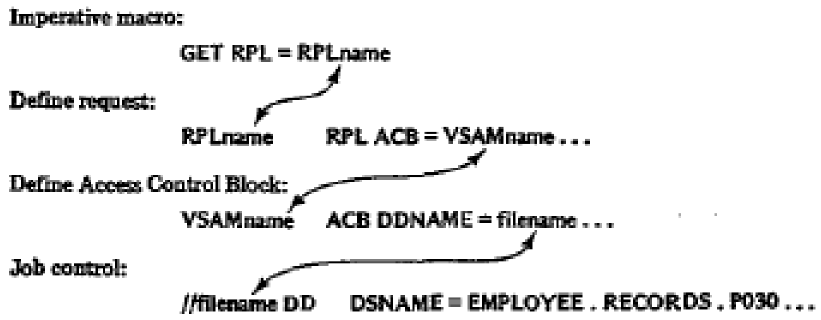
- To request access to a file:

GET	(get a record)
PUT	(write or rewrite a record)
POINT*	(position VSAM at a record)
ERASE	(erase a record previously retrieved with a GET)
ENDREQ*	(end a request)

- To manipulate the information that relates a program to the data:

GENCB*	(generate control block)
MODCB*	(modify control block)
SHOWCB	(show control block)
TESTCB*	(test control block)

A program that accesses a VSAM data set requires the usual OPEN to connect the data set and CLOSE to disconnect it, the GET macro to read records, and PUT to write or rewrite records. An important difference in the use of macros under VSAM is the RPL (**R**equest for **P**arameter **L**ist) macro. As shown in the following relationship, a GET or PUT specifies an RPL macro name rather than a file name. The RPL in turn specifies an ACB (**A**ccess **C**ontrol **B**lock) macro, which in its turn relates to the job control entry for the data set:



```
Imperative macro:
                GET RPL = RPLname
Define request:
                RPLname      RPL ACB = VSAMname . . .
Define Access Control Block:
                VSAMname     ACB DDNAME = filename . . .
Job control:
                //filename DD     DSNAME = EMPLOYEE . RECORDS . P030 . . .
```

The ACB macro is equivalent to the OS DCB or DOS DTF file definition macros. As well, the OPEN macro supplies information about the type of file organization, record length, and key. Each execution of OPEN, CLOSE, GET, PUT, and ERASE causes VSAM to check its validity and to insert a code into register 15 that you can check. A return code of X'00' means that the operation was successful. You can use the SHOWCB macro to determine the exact cause of the error.

THE ACB MACRO: ACCESS METHOD CONTROL BLOCK

The ACB macro identifies a data set that is to be processed. Its main purpose is to indicate the proposed type of processing (sequential or direct) and the use of exit routines, if any. The DEFINE CLUSTER command of AMS has already stored much of the information about the data set in the VSAM catalog. When a program opens the data set via the ACB, VSAM delivers this information to virtual storage.

Entries for an ACB macro may be in any sequence, and you may code only those that you need. Following is the general format, which you code like a DCB or DTF, with a comma following each entry and a continuation character in column 72. All operands are optional.

```
        ACB     AM=VSAM                                          +
                DDNAME=filename,                                 +
                EXLST=address                                    +
                MACRF=([ADR][,KEY]
                       [,DIR][,SEQ][,SKP]
                       [,IN][,OUT]
                       [,NRM][,AIX])                             +
                STRND=number
```

name	The name indicates the symbolic address for the ACB when assembled. If you omit the DDNAME operand from the ACB definition, this name should match the filename in your DLBL or DD job statement.
AM=VSAM	Code this parameter if your installation also uses VTAM; otherwise, the assembler assumes VSAM.
DDNAME	This entry provides the name of your data set that the program is to process. This name matches the filename in your DLBL or DD job statement.
EXLST	The address references a list of your addresses of routines that provide exits. Use the EXLST macro to generate the list, and enter its name as the address. A common use is to code an entry for an end–of–file exit for sequential reading. If you have no exit routines, omit the operand.
MACRF	The options define the type of processing that you plan. In the following, an underlined entry is a default:

ADR|KEY	Use ADR for addressed access (KS and ES) and KEY for keyed access (KS and RR).
DIR|SEQ|SKP	DIR provides direct processing, SEQ provides sequential processing, and SKP means skip sequential (for KS and RR).
IN|OUT	IN retrieves records and OUT permits retrieval, insertion, add–to–end, or update for keyed access and retrieval, update, or add–to–end for addressed access.
NRM|AIX	The DDNAME operand supplies the name of the data set (or path). NRM means normal processing of the data set, whereas AIX means that this is an alternate index.

Other MACRF options are **RST|NRS** for resetting catalog information and **NUB|UBF** for user buffers.

STRNO	The entry supplies the total number of RPLs (request parameter lists) that your program will use at the same time (the default is 1).

ACB also has provision for parameters that define the number and size of buffers; however, the macro has standard defaults.

In the program example in Fig. 28–4, the ACB macro VSMFILOT has only two entries and allows the rest to default. Access is keyed (KEY), processing is sequential (SEQ), and the file is output (OUT). There is no exit list, STRNO defaults to 1, and MACRF defaults to NRM (normal path).

The assembler does not generate an I/O module for an ACB, nor does the linkage editor include one. Instead, the system dynamically generates the module at execute time.

THE RPL MACRO: REQUEST PARAMETER UST

The request macros GET, PUT, ERASE, and POINT require a reference to an RPL macro. For example, the program in Fig. 28–4 issues the following GET macro:

```
    GET RPL=RPLISTIN
```

The operand supplies the name of the RPL macro that contains the information needed to access a record. If your program is to access a data set in different ways, you can code an RPL macro for each type of access; each RPL keeps track of its location in the data set.

The standard format for RPL is as follows. The name for the RPL macro is the one that you code in the GET or PUT operand. Every entry is optional. (ELB: Note the non–blank character in column 72 to indicate continuation on the next line.)

Name field	Operation	Operands	Column 72
RPLname	RPL	AM=VSAM,	+
		ACB=address,	+
		AREA=address,	+
		AREALEN=length,	+
		ARG=address,	+
		KEYLEN=length,	+
		OPTCD=(options),	+
		RECLEN = length	

AM	The entry VSAM specifies that this is a VSAM (not VTAM) control block.
ACB	The entry gives the name of the associated ACE that defines the data set.
AREA	The address references an I/O work area in which a record is available for output or is to be entered on input.
AREALEN	The entry supplies the length of the record area.
ARG	The address supplies the search argument-a key, including a relative record number or an RBA.
KEYLEN	The length is that of the key if processing by generic key. (For normal keyed access, the catalog supplies the key length.)

OPTCD	Processing options are SEQ, SKP, and DIR; request options are UPD (update) and NUP (no update). For example, a direct update would be (DIR,UPD).
RECLEN	For writing a record, your program supplies the length to VSAM, and for retrieval, VSAM supplies the length to your program. If records are variable length, you can use the SHOWCB and TESTCB macros to examine the field (see the IBM Supervisor manual).

THE OPEN MACRO

The OPEN macro ensures that your program has authority to access the specified data set and generates VSAM control blocks.

```
[label] OPEN address [,address ...... ]
```

The operand designates the address of one or more ACBs, which you may code either as a macro name or as a register notation (registers 2–12); for example:

```
         OPEN VSFILE
    or   LA 6,VSFILE
         OPEN (6)
```

You can code up to 16 filenames in one OPEN and can include both ACB names and DCB or D1F names. Note, however, that to facilitate debugging, avoid mixing them in the same OPEN. OPEN sets a return code in register 15 to indicate success (zero) or failure (nonzero), which your program can test:

X'00'	Opened all ACBs successfully.
X'04'	Opened all ACBs successfully but issued a warning message for one or more.
X'08'	Failed to open one or more ACBs.

On a failed OPEN or CLOSE, you can also check the diagnostics following program execution for a message such as **`OPEN ERROR X'6E'`**, and check Appendix K of the IBM Supervisor manual for an explanation of the code.

THE CLOSE MACRO

The CLOSE macro completes any I/O operations that are still outstanding, writes any remaining output buffers, and updates catalog entries for the data set.

```
[label ] CLOSE address [,address ... ]
```

You can code up to 16 names in one CLOSE and can include both ACB names and DCB or DTF names. CLOSE sets a return code in register 15 to indicate success or failure, which your program can test:

X'00'	Closed all ACBs successfully.
X'04'	Failed to close one or more ACBs successfully.
X'08'	Insufficient virtual storage space for close routine or could not locate modules.

THE REQUEST MACROS: GET. PUT. ERASE

The VSAM request macros are GET, PUT, ERASE, POINT, and ENDREQ. For each of these, VSAM sets register 15 with a return code to indicate success or failure of the operation, as follows:

X'00'	Successful operation.
X'04'	Request not accepted because of an active request from another task on the same RPL. End-of-file also causes this return code.
X'08'	A logical error; examine the specific error code in the RPL.
X'0C'	Uncorrectable I/O error; examine the specific error code in the RPL.

The GET Macro

GET retrieves a record from a data set. The operand specifies the address of an RPL that defines the data set being processed. The entry may either (1) cite the address by name or (2) use register notation, any register 2–12, in parentheses.

You may use register 1; its use is more efficient, but GET does not preserve its address.

```
1.      GET RPL=RPLname

2.      LA reg,RPLname
        GET RPL=(reg)
```

The RPL macro provides the address of your work area where GET is to deliver an input record. Register 13 must contain the address of a save area defined as 18·fullwords.

Under sequential input, GET delivers the next record in the data set. The OPTCD entry in the RPL macro would appear, for example, as OPTCD=(KEY,SEQ) or OPTCD=(ADR,SEQ). You have to provide for end–of–file by means of an EXLST operand in the associated ACB macro; see Fig. 28–4 for an example.

For non–sequential accessing, GET delivers the record that the key or relative record number specifies in the search argument field. The OPTCD entry in the RPL macro would appear, for example, as OPTCD = (KEY,SKP) or OPTCD= (KEY,DIR), or as an RBA in the search argument field, as OPTCD = (ADR,DIR).

You also use GET to update or delete a record.

The PUT Macro

PUT writes or rewrites a record in a data set. The operand of PUT specifies the address of an RPL that defines the data set being processed. The entry may either (1) cite the address by name or (2) use register notation, any register 2-12, in parentheses. You may use register 1; its use is more efficient, but PUT does not preserve its address.

```
1.      PUT RPL=RPLname

2.      LA reg,RPLname
        PUT RPL=(reg)
```

The RPL macro provides the address of your work area containing the record that PUT is to add or update in the data set. Register 13 must contain the address of a save area defined as 18 fullwords.

To create (load) or extend a data set, use sequential output. The OPTCD entry in the RPL macro would appear, for example, as OPTCD=(SEQ or SKP). SKP means "skip sequential" and enables you to start writing at any specific record.

For writing a KSDS or RRDS, if OPTCD contains any of the following, PUT stores a new record in key sequence or relative record sequence:

OPTCD=(KEY,SKP,NUP)	Skip, no update
OPTCD=(KEY,DIR,NUP)	Direct, no update
OPTCD=(KEY,SEG,NUP)	Sequential, no update

Note that VSAM does not allow you to change a key in a KSDS (delete the record and write a new one). To change a record, first GET it using OPTCD= UFD, change its contents (but not the key), and PUT it, also using OPTCD=UFD. To write a record in ESDS, use OPTCD=(ADR, ….).

The ERASE Macro
The purpose of the ERASE macro is to delete a record from a KSDS or an RRDS. To locate an unwanted record, you must previously issue a GET with an RPL specifying OPTCD=(UFD...).

```
[label] ERASE RPL=address or =(register)
```

For ESDS, a common practice is to define a delete byte in the record. To "delete" a record, insert a special character such as X'FF'; all programs that process the data set should bypass all records containing the delete byte. You can occasionally rewrite the data set, dropping all deletes.

THE EXLST MACRO

If your ACB macro indicates an EXLST operand, code a related EXLST macro. EXLST provides an optional list of addresses for user exit routines that handle end–of–file and error analysis. All operands in the macro are optional.

When VSAM detects the coded condition, the program enters your exit routine. Register 13 must contain the address of your register save area. For example, if you are reading sequentially, supply an end–of–data address (EODAD) in the EXLST macro-see the ACB for VSMFILIN in Fig. 28–4.

Name field	Operation	Operands	Column 72

[label]	EXLST	AM=VSAM,
		EODAD=address,
		LERAD=address,
		SYNAD=address
			+
			+
			+

Here are explanations of the operands for EXLST:

VSAM	Indicates a VSAM control block.
EODAD	Supplies the address of your erid-of-data routine. You may also read sequentially backward, and VSAM enters your routine when reading past the first record. The request return code for this condition is X'04'.
LERAD	Indicates the address of the routine that analyzes logical errors that occurred during GET, PUT, POINT, and ERASE. The request return code for this condition is X'08'.
SYNAD	Provides the address of your routine that analyzes physical I/O errors on GET, PUT, POINT, ERASE, and CLOSE. The request return code for this condition is X'OC'.

Other operands are EXCPAD and JRNAD.

THE SHOWCB MACRO

The original program in Fig. 28–4 contained an error that caused it to fail on a PUT operation. The use of the SHOWCB macro in the error routine for PUT (R30PUT) helped determine the actual cause of the error.

The purpose of SHOWCB is to display fields in an ACB, EXLST, or RPL.
Code SHOWCB following a VSAM macro where you want to identify errors that VSAM has detected. The SHOWCB in the PUT error routine in Fig. 28–4 is as follows:

```
         SHOWCB RPL=RPLISTOT,AREA=FDBKWD,FIELDS=(FDBK),LENGTH=4
….
FDBKWD   DC F'O'
```

AREA	Designates the name of a fullword where VSAM is to place an error code.
FIELDS	Tells SHOWCB the type of display; the keyword FDBK (feedback) causes a display of error codes for request macros.
LENGTH	Provides the length of the area in bytes.

On a failed request, VSAM stores the error code in the rightmost byte of the fullword area. These are some common error codes:

X‘08’	Attempt to store a record with a duplicate key.
X‘0C’	Out-of-sequence or duplicate record for KSDS or RRDS.
X‘10’	No record located on retrieval.
X‘1C’	No space available to store a record.

Your program can test for the type of error and display a message. For nonfatal errors, it could continue processing; for fatal errors, it could terminate.

The original error in Fig. 28–4 was caused by the fact that the RPL macro RPLISTOT did not contain an entry for RECLEN; the program terminated on the first PUT error, with register 15 containingX'08' (a "logical error"). Insertion of the SHOWCB macro in the next run revealed the cause of the error in FDBKWD: 00006C. Appendix K of the IBM Supervisor manual explains the error (in part) as follows: "The RECLEN value specified in the RPL macro was [either] larger than the allowed maximum [or] equal to zero...." Coding a RECLEN operand in the RPL macro solved the problem, and the program then executed through to normal termination. One added point: Technically, after" each SHOWCB, you should test register 15 for a successful or failed operation.

SAMPLE PROGRAM: LOADING A KEY-SEQUENCED DATA SET

The program in Fig. 28–4 reads records from the system reader and sequentially creates a key-sequenced data set. A DEFINE CLUSTER command has allocated space for this data set as INDEXED (KSDS), with three tracks, a 4-byte key starting in position 0, and an 80–byte record size. The program loads the entire data set and closes it on completion. For illustrative (but not practical) purposes, it reopens the data set and reads and prints each record. The PUT macro that writes records into the data set is:

```
PUT RPL=RPLISTOT
```

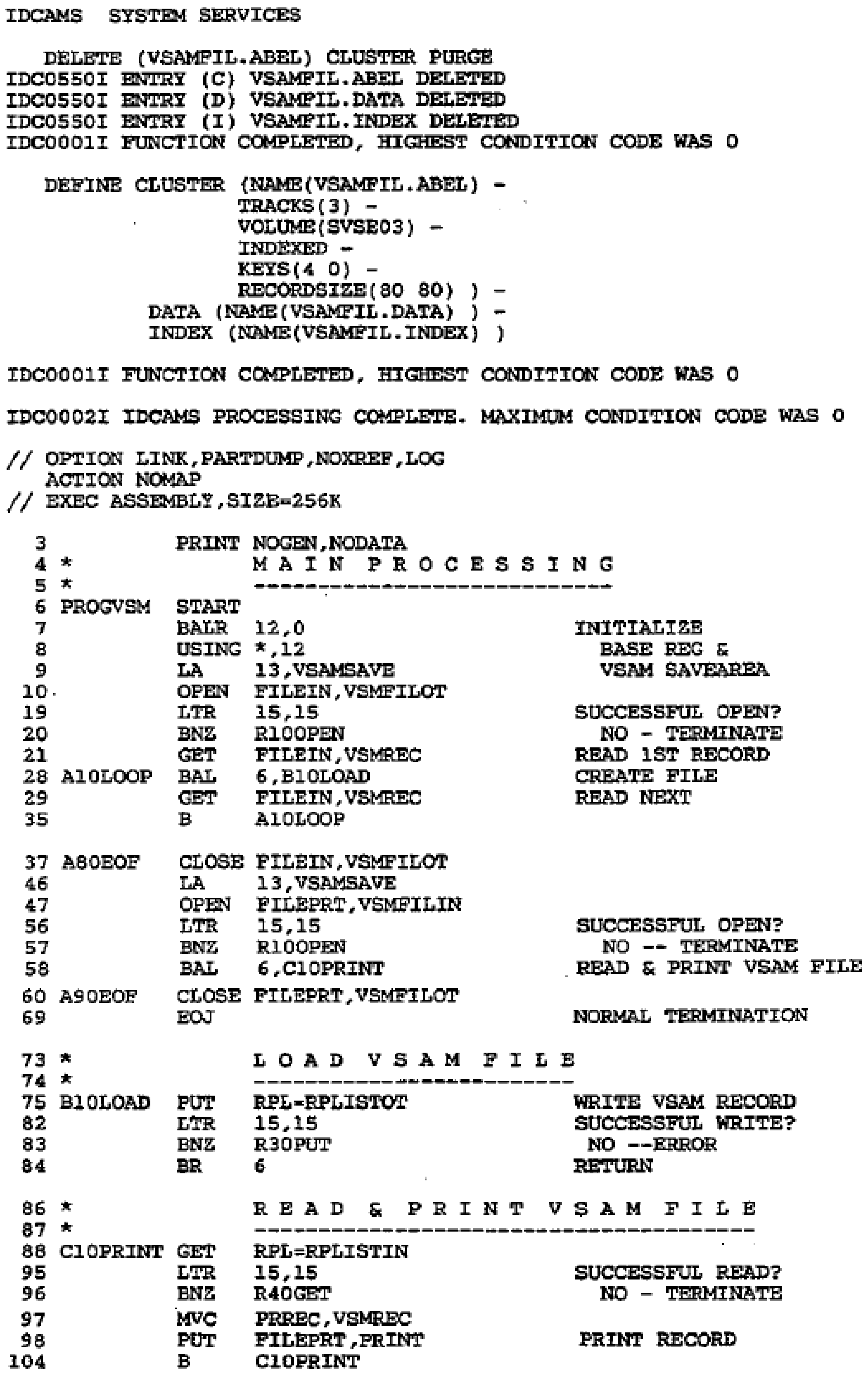
RPLISTOT defines the name of the ACB macro (VSMFILOT), the address of the output record, and its length. Although the example simply duplicates the record into the data set, in practice you would probably define various fields and store numeric values as packed or binary.

The ACB macro defines VSMFILOT for keyed accessing, sequential processing, and output. The DDNAME, VSAMFIL, in this example relates to the name for the data set in the DLBL job control entry (DD under OS*)*.

For reading the data set, the GET macro is

```
GET RPL=RPLISTIN
```

RPLISTIN defines the name of the ACB macro (VSMFILIN), the address in which GET is to read an input record, and the record length.



```
IDCAMS  SYSTEM SERVICES

   DELETE (VSAMFIL.ABEL) CLUSTER PURGE
IDC0550I ENTRY (C) VSAMFIL.ABEL DELETED
IDC0550I ENTRY (D) VSAMFIL.DATA DELETED
IDC0550I ENTRY (I) VSAMFIL.INDEX DELETED
IDC0001I FUNCTION COMPLETED, HIGHEST CONDITION CODE WAS 0

   DEFINE CLUSTER (NAME(VSAMFIL.ABEL) -
                  TRACKS(3) -
                  VOLUME(SVSE03) -
                  INDEXED -
                  KEYS(4 0) -
                  RECORDSIZE(80 80) ) -
           DATA (NAME(VSAMFIL.DATA) ) -
           INDEX (NAME(VSAMFIL.INDEX) )

IDC0001I FUNCTION COMPLETED, HIGHEST CONDITION CODE WAS 0

IDC0002I IDCAMS PROCESSING COMPLETE. MAXIMUM CONDITION CODE WAS 0

// OPTION LINK,PARTDUMP,NOXREF,LOG
   ACTION NOMAP
// EXEC ASSEMBLY,SIZE=256K

  3          PRINT NOGEN,NODATA
  4 *              M A I N   P R O C E S S I N G
  5 *              -----------------------------
  6 PROGVSM  START
  7          BALR  12,0                       INITIALIZE
  8          USING *,12                         BASE REG &
  9          LA    13,VSAMSAVE                  VSAM SAVEAREA
 10          OPEN  FILEIN,VSMFILOT
 19          LTR   15,15                      SUCCESSFUL OPEN?
 20          BNZ   R10OPEN                      NO - TERMINATE
 21          GET   FILEIN,VSMREC              READ 1ST RECORD
 28 A10LOOP  BAL   6,B10LOAD                  CREATE FILE
 29          GET   FILEIN,VSMREC              READ NEXT
 35          B     A10LOOP

 37 A80EOF   CLOSE FILEIN,VSMFILOT
 46          LA    13,VSAMSAVE
 47          OPEN  FILEPRT,VSMFILIN
 56          LTR   15,15                      SUCCESSFUL OPEN?
 57          BNZ   R10OPEN                      NO -- TERMINATE
 58          BAL   6,C10PRINT                 READ & PRINT VSAM FILE
 60 A90EOF   CLOSE FILEPRT,VSMFILOT
 69          EOJ                              NORMAL TERMINATION

 73 *              L O A D   V S A M   F I L E
 74 *              ------------------------
 75 B10LOAD  PUT   RPL=RPLISTOT               WRITE VSAM RECORD
 82          LTR   15,15                      SUCCESSFUL WRITE?
 83          BNZ   R30PUT                      NO --ERROR
 84          BR    6                          RETURN

 86 *              R E A D   &   P R I N T   V S A M   F I L E
 87 *              ---------------------------------------
 88 C10PRINT GET   RPL=RPLISTIN
 95          LTR   15,15                      SUCCESSFUL READ?
 96          BNZ   R40GET                       NO - TERMINATE
 97          MVC   PRREC,VSMREC
 98          PUT   FILEPRT,PRINT               PRINT RECORD
104          B     C10PRINT
```

Figure 28–4 Loading a key–sequenced data set

```
*              E R R O R   R O U T I N E S
*              -------------------------
R10OPEN  MVI   ERRCDE,C'O'             OPEN ERROR
         B     R90DUMP
R30PUT   MVI   ERRCDE,C'P'             PUT ERROR
         ST    15,SAVE15
         SHOWCB RPL=RPLISTOT,AREA=FDBKWD,FIELDS=(FDBK),LENGTH=4
         CLOSE FILEIN,VSMFILOT
         B     R90DUMP
R40GET   MVI   ERRCDE,C'G'             GET ERROR
         ST    15,SAVE15
         SHOWCB RPL=RPLISTIN,AREA=FDBKWD,FIELDS=(FDBK),LENGTH=4
         CLOSE FILEPRT,VSMFILOT
R90DUMP  EQU   *
         PDUMP ERRCDE,PRINT+133
         EOJ                           ABNORMAL TERMINATION
*              ------------------------
*              D E C L A R A T I V E S
*              ------------------------
FILEIN   DEFIN A80EOF                  DEFINE INPUT FILE
FILEPRT  DEFPR                         DEFINE PRINTER FILE
VSMFILOT ACB   DDNAME=VSAMFIL,         DEFINE VSAM O/P FILE    +
               MACRF=(KEY,SEQ,OUT)

RPLISTOT RPL   ACB=VSMFILOT,           RPL FOR VSMFILOT        +
               AREA=VSMREC,                                    +
               AREALEN=80,                                     +
               RECLEN=80,                                      +
               OPTCD=(KEY,SEQ,NUP)

VSMFILIN ACB   DDNAME=VSAMFIL,         DEFINE VSAM I/P FILE    +
               MACRF=(KEY,SEQ,IN),                             +
               EXLST=EOFDCB

EOFDCB   EXLST EODAD=A90EOF            EOF EXIT FOR VSAM I/P
RPLISTIN RPL   ACB=VSMFILIN,           RPL FOR VSMFILIN        +
               AREA=VSMREC,                                    +
               AREALEN=80,                                     +
               OPTCD=(KEY,SEQ,NUP)

VSAMSAVE DS    18F                     VSAM SAVEAREA
ERRCDE   DC    X'00'                   ERROR CODE
SAVE15   DS    F
FDBKWD   DC    F'0'
VSMREC   DS    0CL80                   INPUT/OUTPUT RECORD
RECKEY   DS    CL04                    *
         DS    CL76                    *

PRINT    DS    0CL133                  PRINT RECORD
         DC    X'09'                   *
PRREC    DC    CL80' '                 *
         DC    CL52' '                 *
         LTORG
               =C'$$BOPEN '
               =C'$$BCLOSE'
               =CL8'IKQVTMS'
               =CL8'$$BPDUMP'
               =A(ERRCDE,PRINT+133)
               =A(FILEIN)
               =A(VSMREC)
               =A(RPLISTOT)
               =A(RPLISTIN)
               =A(FILEPRT)
               =A(PRINT)
         END   PROGVSM

// EXEC LNKEDT,SIZE=128K

// DLBL VSAMFIL,'VSAMFIL.ABEL',,VSAM
// EXTENT SYS008,SVSE03
// ASSGN SYS008,X'303'
// EXEC ,SIZE=128K
```

Figure 28–4 Loading a key–sequenced data set (continued)

The ACB macro defines VSMFILIN for keyed access, sequential processing, and input. The DDNAME, VSAMFIL, relates to the name for the data set in the DLBL job control entry. Note that there is an ACB and RPL macro for both input and output, but both ACB macros specify the same DDNAME: VSAMFIL.

Error routines are for failures on OPEN, GET, and PUT. These rather primitive routines supply an error code and the contents of the declaratives; in practice, you may want to enlarge these routines. If you fail to provide error routines, your program may crash with no clear cause.

During testing, you may have changed the contents of a VSAM data set and now want to reload (re-create) the original data set. Except for updating with new keys, VSAM does not permit overwriting records in a data set. You have to use IDCAMS to DELETE and again DEFINE the data set as follows:

```
DELETE(data-set-name) CLUSTER PURGE …

DEFINE CLUSTER(NAME(data-set-name) ….)
```

Loading an ESDS
To convert the program in Fig. 19-4 from KSDS to ESDS, change DEFlNE CLUSTER from INDEXED to NONINDEXED and delete the KEYS and INDEX entries. Change the ACB MACRF from KEY to ADR, and change the RPL OPTCD from KEY to ADR – that's all.

KEYED DIRECT RETRIEVAL
Key-sequenced data sets provide for both sequential and direct processing by key. For direct processing, you must supply VSAM with the key of the record to be accessed. If you use a key to access a record directly, it must be the same length as the keys in the data set (as indicated in the KEYS operand of DEFINE CLUSTER), and the key must actually exist in the data set. For example, if you request a record with key 0028 and there is no such record, VSAM returns an error code in register 15.

Using the data set in Fig. 19-4, assume that a program is to access records directly. A user enters record key numbers via a terminal, and the program is to display the record on the screen. In this partial example, the RPL macro specifies the name (ARG) of the key to be in a 4-byte field named KEYFLD. These are the specific coding requirements for the ACB, RPL, and GET macros:

For updating a KSDS record, change the MACRF from IN to OUT, and change the OPTCD from NUP to UPD. GET the record, make the required changes to it (but not the key!), and PUT the record using the same RPL.

```
VSMFILE  ACB DDNAME=name,                +
             MACRF=(KEY,DIR,IN)
RPLIST   RPL ACB=VSMFILE,                +
             AREA=DCBREC,                +
             AREALEN=80,                 +
             ARG=KEYFLD,                 +
             OPTCD=(KEY,DIR,NUP)
KEYFLD   DS CL4
DCBREC   DS CL80
[Accept a key number from the terminal]
         MVC KEYFLD,keyno
         GET RPL=RPLI ST
         LTR 15,15
         BNZ error
[Display the record on the screen]
```

SORTING VSAM FILES
You can sort VSAM records into either ascending or descending sequence. You must first use DEFINE CLUSTER to allocate a vacant data set (NONINDEXED) for SORT to write the sorted data set. Here is a typical SORT specification:

```
// EXEC SORT,SIZE=256K
     SORT FIELDS=(1,4,CH,A,9,4,PD,D)
     RECORD TYPE=F,LENGTH=(150)
     INPFIL VSAM
     OUTFIL ESDS
     END
/*
```

SORT causes the SORT program to load into storage and begin execution.

SORT FIELDS defines the fields to be sorted, indicated by major control to minor, from left to right. In this example, the major sort field begins in position 1 (the first position), is 4 bytes long, is in character (CH) format, and is to be sorted in ascending (A) sequence. The minor sort field begins in position 9, is 4 bytes long, is in packed (PD) format, and is to be sorted in descending (0) sequence. The example could be a sort of departments in ascending sequence, and within each department are employee salaries in descending sequence.

RECORD TYPE indicates fixed (F) length and record length (150 bytes).

INPFIL informs SORT that the input file is VSAM; SORT can determine the type of data set from the VSAM catalog.

OUTFIL defines the type of output file, in this case entry-sequenced. This entry should match the DEFINE CLUSTER for this data set, NONINDEXED.

Job control commands for SORTIN and SORTOUT provide the names of the data sets. Since job control varies by operating system and by installation requirements, check with your installation before attempting the SORT utility.

VSAM UTILITY PRINT

IDCAMS furnishes a convenient utility program named PRINT that can print the contents of a VSAM, SAM, or ISAM data set. The following provides the steps for OS and for DOS:

```
OS:  //STEP EXEC PGM=IDCAMS
       PRINT INFILE(filename) CHARACTER or HEX or DUMP
     /*
DOS: // EXEC IDCAMS,SIZE=256K
       PRINT INFILE(filename) CHARACTER or HEX or DUMP
     /*
```

The options for PRINT indicate the format of the printout, in character, hexadecimal, or both (DUMP prints hex on the left and character format on the right).

INFILE(fiIename) matches the name in the OS DD or DOS DLBL job statement with any valid filename as long as the two are identical. The DD or DLBL statement notifies VSAM which data set is to print.

PRINT lists KSDS and ISAM data sets in key sequence and lists ESDS, RRDS, and SAM data sets in physical sequence. You can also print beginning and ending at a specific record.

KEY POINTS

- A key-sequenced data set (KSDS) maintains records in sequence of key, such as employee or part number, and is equivalent to indexed sequential access method.
- An entry-sequenced data set (ESDS) maintains records in the sequence in which they were initially entered and is equivalent to sequential organization.
- A relative-record data set (RRDS) maintains records in order of relative record number and is equivalent to direct file organization.
- For the three types of data sets, VSAM stores records in groups (one or more) of control intervals. At the end of each control interval is control information that describes the data records.
- Before physically writing (loading) records in a VSAM data set, you must first catalog its structure. Access method services (AMS) enables you to furnish VSAM with such details about the data set as its name, organization type, record length, key location, and password (if any).
- VSAM furnishes two types of accessing, keyed and addressed, and three types of processing, sequential, direct, and skip sequential.
- The most common errors in processing VSAM data sets occur because of the need to match definitions in the program, job control, and the cataloged VSAM data set.

- The data-set-name in job control (such as CUSTOMER.INQUIRY) must agree with the NAME(data–set–name) entry in DEFJNE CLUSTER. This name is the only one by which VSAM recognizes the data set. VSAM relates the ACB DDNAME in the program to the job control name and the job control name to the data-set-name.
- If a data set is cataloged as KSDS, ESDS, or RRDS, each program must access it accordingly.
- For KSDS, the length and starting position of the key in a record must agree with the KEYS entry in DEFINE CLUSTER and, for direct input, with the defined ARG in the OPTCD.
- Every program that references the data set defines the fields with identical formats and lengths in the same positions; the actual field names need not be identical. You may define as character any input field in a record that the program does not reference. The simplest practice is to catalog all record definitions in the assembler source library and COPY the definition into the program during assembly.
- After each OPEN, CLOSE, GET, PUT, and SHOWCB, test register 15 for success or failure, and use SHOWCB (as well as TESTCB) as a debugging aid.