IBM

World Trade Systems Centers

Boeblingen - Poughkeepsie - Raleigh - San Jose

VSAM PRIMER AND REFERENCE

G320-5774-01

VSAM PRIMER and REFERENCE

AUTHOR:
ULRICH SCHWENK
WORLD TRADE SYSTEMS CENTER
SANTA TERESA
CALIFORNIA

G320-5774-1

This document was created and formatted with SCRIPT/VS (program number 5796-PHL) under the CMS component of VM/370. For special handling of the figures the SCRIPT/VS code was temporarily modified to adjust for proper page length. The SCRIPT/VS output was then edited and modified with a VM/370 EXEC procedure (special handling for imbedded figures). The output was transmitted to OS/VS2 MVS via the VM/370 Networking PRPQ (program number 5799-ATA), and printed on the IBM 3800 Printing Subsystem. The fonts used are T11, FM10, GF15 plus a special character set to print the disk and tape diagrams.

This edition includes Technical Newsletter G320-5781, which added the Copyright information at the bottom of this page.

CONTENTS

FIGURES:

1.0 INTRODUCTION AND GUIDELINES TO THIS MANUAL

This manual is a guide and a reference source for the person using VSAM.

This document has not been submitted to any formal IBM test and has not been checked for technical accuracy. Potential users should evaluate its usefullness in their own environment prior to any implementation.

We want to acknowledge our debts to the many individuals who gave us assistance in answering technical questions. In particular we want to give credit to Roman Chiocca from Switzerland and David Pepper from Nottingham, England for their participation in development of this document.

1.1 DESCRIPTION OF THIS MANUAL

The objectives of this manual are:

- To give the new VSAM user the information required to understand, evaluate, and use VSAM properly.
- To clarify VSAM functions for current VSAM application programmers who will be working with VSAM. The practical, straightforward approach adopted should dispel much of the apparent complexity sometimes associated with VSAM.

Wherever possible an example is used to reinforce a description, and a modified version of the same example is used in any related section to give continuity to the explanation. Diagrams are used liberally to aid understanding.

This manual is not operating system dependent and is intended for DOS/VS, MVS, SVS, and VS1 users, who should have some familiarity with access methods.

Information contained in this manual includes:

- A general description of VSAM and Access Method Services concepts and capabilities, including catalog usage and comparisons with other access methods.
- Methods of calculating storage and data set space requirements.
- General system considerations.
- Descriptions and examples of using Access Method Services facilities.
- An explanation of VSAM installation procedures.
- Programming considerations.
- Performance considerations.

- Integrity considerations.

This manual is not intended as a replacement for the SRL manuals and IBM courses, but as a supplement to them. It will be particularly useful as an initial point of reference to give a basic understanding in specific areas, which may then be followed up by reference to the appropriate SRL manual or course. For example, though this manual gives an understanding of the principles of programming with VSAM, reference manuals would be needed to actually write code. Likewise, a clear understanding of how to use Access Method Services commands to create, modify, delete, print and move data sets is given, but the appropriate manual or course is necessary to properly code actual commands.

This manual is a learning guide and a source of field-developed techniques and advice. It does not eliminate the need for careful planning and education. This publication therefore intends to show, in an uncomplicated way, how to use VSAM simply and effectively.

Knowledge of concepts of Virtual Storage Systems (like DOS/VS, MVS,SVS,VS1) is required. Basic knowledge of other Access Methods (like SAM or ISAM) is desirable.

1.2 HOW TO USE THIS MANUAL

1.2.1 DOS/VS AND OS/VS USERS

All chapters can be followed in the order they are presented. Specific information is included in the Appendices. The index at the end of the manual can be used to find a specific item.

Restrictions or notes to specific Operating Systems are added where applicable.

1.2.2 MVS USERS NOT USING VSAM APPLICATIONS

Each MVS System must have a VSAM master catalog as the main system catalog.

Even if a user does not plan to use VSAM applications, he should be familiar with the basic VSAM structures (as the VSAM master catalog is also a VSAM data set).

The most important chapters for this group of user are the chapters 2.0, 3.0, 6.0, and appendix A.0.

It is suggested, however, to also read the other chapters, as most of them also contain information important for users, using only VSAM catalogs.

2 VSAM PRIMER and REFERENCE

1.3 BIBLIOGRAPHY OF MANUALS CONTAINING VSAM INFORMATION

- DOS/VS Rel.34:

DOS/VS Introduction to DOS/VS	GC33-5370
DOS/VS System Management Guide	GC33-5371
DOS/VS Data Management Guide	GC33-5372
DOS/VS Supervisor & I/O Macros	GC33-5373
DOS/VS Messages	GC33-5379
DOS/VS Access Method Services User's Guide	GC33-5382
DOS/VS Entry Users Guide	GC33-6074
DOS/VS VSAM LOGIC Liocs	SY33-8562
DOS/VS Handbook Volume 1	SY33-8571
DOS/VS Handbook Volume 2	SY33-8572

- OS/VS:

OS/VS VSAM Options For Advanced Applications	GC26-3819
OS/VS VSAM Programmer's Guide	GC26-3838
Planning for Enhanced VSAM Under OS/VS	GC26-3842
VS1 to MVS Conversion Notebook	GC28-0953

- OS/VS1 (additional to the OS/VS manuals listed above):

OS/VS1 Features Supplement	GC20-1752
OS/VS1 Planning And Use Guide	GC24-5090
OS/VS1 System Generation	GC26-3791
OS/VS1 Access Method Services	GC26-3840
OS/VS1 System Messages	GC38-1001

- OS/VS2 MVS (additional to the OS/VS manuals listed above):

OS/VS2 System Generation Reference	GC26-3792
OS/VS2 Access Method Services	GC26-3841
OS/VS2 MVS CVOL	GC26-3864
OS/VS2 Initialization and Tuning Guide	GC28-0681
OS/VS2 Conversion Notebook	GC28-0689
OS/VS2 Performance Notebook	GC28-0886
OS/VS2 CVOL Processor	GC35-0010
OS/VS2 System Messages	GC38-1002

- OS/VS2 SVS:

OS/VS2 VSAM Planning Guide	GC26-3799
OS/VS2 VSAM Programmer's Guide	GC26-3818
OS/VS2 Planning And Use Guide	GC28-0600
OS/VS2 Data Management For System Programmers	GC28-0631
OS/VS2 Access Method Services	GC35-0009
OS/VS2 VSAM System Information	GC35-3835
OS/VS2 VSAM Options For Advanced Applications	GT26-3819

- OS/VS2 SVS with ICR

OS/VS2 Access Method Services	GC26-3867
OS/VS2 VSAM Options For Advanced Applications	GC26-3870
OS/VS2 Planning For Enhanced VSAM	GC26-3869
OS/VS2 VSAM Programmer's Guide	GC26-3868

2.0 WHAT IS VSAM

2.1 BASIC DESCRIPTION OF VSAM

Virtual Storage Access Method (VSAM) is a component of DOS/VS and OS/VS data management and is supported in DOS/VS, OS/VS1, OS/VS2 SVS, and OS/VS2 MVS. In VM/370 the CMS/VSAM support is based on DOS/VS.

The VSAM Assembler Language macros used in DOS/VS and OS/VS are compatible. In addition, a VSAM data set contained on a DOS/VS volume can be processed by OS/VS programs. Similarly, a VSAM data set contained on an OS/VS volume can be processed by DOS/VS programs. This compatibility enables VSAM data sets to be processed both by DOS/VS and OS/VS, and makes a transition from DOS/VS to OS/VS much easier.

VSAM supports both sequential and direct processing and is designed to supersede ISAM, although the two access methods can coexist in the same Operating System. VSAM supports functions equivalent to those of ISAM, and provides better performance (see description starting in section 5.4 on page 75). VSAM data sets cannot be accessed by any other Access Method.

In addition, the three data organizations supported by VSAM can be used in place of BSAM or QSAM for sequential data sets and in place of BDAM for directly organized data sets (see description starting in section 5.1 on page 71). The new structure and features of VSAM make it more suited to data base and online environments than other access methods.

VSAM support consists of the following:

- Three data set organizations : Entry-Sequenced Data Sets (ESDS), Key-Sequenced Data Sets (KSDS), and Relative-Record Data Sets (RRDS). The user can thus choose the logical data organization that is best suited for his or her particular needs. They are supported on DASD (Direct Access Storage Device) only.

  - A VSAM ESDS is a sequential data set (similar to a SAM data set).

  - A VSAM KSDS is a sequential data set with an index (similar to an ISAM data set).

  - A VSAM RRDS is a data set with preformatted slots for fixed length records to be accessed by a record number (similar to a DAM data set).

- VSAM master and user catalogs which contain information about VSAM data sets, volumes, spaces, etc.

- The multifunction utility program Access Method Services, which provides required VSAM services, such as catalog or data set creation, loading, merging, printing, and VSAM catalog maintenance. This utility supports DASD as well as tape devices.

- Access method routines with which the user interfaces to process logical records in VSAM data sets.

- The ISAM interface routines, which make the transition from ISAM to VSAM possible with little or no modification of ISAM programs.

2.1.1 CREATING VSAM OBJECTS

Unlike nonVSAM data sets which are usually created by using JCL, VSAM objects (VSAM objects are data sets, data spaces, catalogs, etc.) may only be created by using the utility Access Method Services.

One group of commands creates VSAM objects. These are the DEFINE commands.

The term 'defining' a VSAM object means creating an entry in a VSAM catalog without any data transfer. The contents of the object may be loaded with a user program or with Access Method Services.

For further information and an overview of the various Access Method Services commands, see chapter 4.0 starting on page 57.

2.2 VSAM DATA SET STRUCTURE

The logical records of VSAM data sets are stored differently from the way in which logical records in nonVSAM data sets are stored.

2.2.1 CONTROL INTERVALS

It is fairly obvious that transferring a single data record or a complete data set between virtual and auxiliary storage could be an inefficient and time-consuming way of processing records. What is needed is to structure data records in a way to make VSAM as efficient as possible.

To accomplish this, so-called control intervals have been designed to contain the data records. The disk space on a direct access storage device (like IBM 3330, 3340, 3344, and 3350) has been divided into segments of information. The size of the segments can vary from one VSAM data set to another VSAM data set, but for a specific data set, the size of each segment is fixed, either by VSAM or by the the user (within limits acceptable to VSAM). These segments of information are called control intervals. Figure 1 illustrates the data records (and the control information describing them) stored in a control interval.

A control interval is the unit of data transferred between virtual storage and disk storage; for example VSAM will transfer the control interval shown in figure 1 to virtual storage if one or more of the data records in that control interval needs to be updated, deleted, etc.

cf = control fields
us = unused space

Figure 1.Control Interval

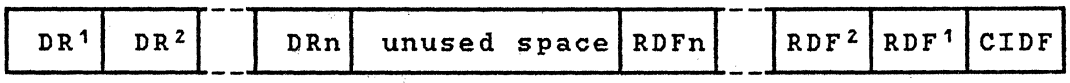
The control interval must be a certain size (a multiple of 512 bytes unless the control interval-size is greater than 8192, then the multiple is 2048). The maximum size of a control interval is 32,768 bytes.

If the user specified control interval is different from the allowed sizes (see page 195), VSAM will substitute the defined value by a default value (the control interval size should be carefully chosen, since a large default control interval will demand corresponding large buffer space).

A control interval can span several tracks depending on control interval size and track capacity but it cannot cross control area boundaries (the control area is described in the next section).

A control interval always contains control information fields plus an integral number of logical records (variable or fixed-length), and unused (free) space, or data records alone, or free space alone. A record may cross more than one control interval only when the data set has been defined with spanned format (described in section 2.2.3 on page 11).

Figure 2 shows that control information fields consist of two types of fields: one CIDF (Control Interval Definition Field), and one or more RDFs (Record Definition Fields).



DR = data record

Figure 2.Control interval structure with control fields

There is one Control Interval Definition Field (CIDF) per control interval and usually multiple Record Definition Fields (RDF) depending on the number of different logical records lengths.

As there is no parameter in VSAM to define the data set records as fixed-length or variable-length records (like RECFM for DOS/VS or OS/VS data sets), VSAM takes into consideration the length of

adjacent records in a control interval when writing a record (see also description of parameter RECORDSIZE on page 94). If two or more adjacent records have the same length, only two RDFs are used for this group. See following examples for records of the same and different length.

RDFs describe length of record and how many adjacent records are of the same length. With fixed-length records, only two RDFs are used, even if more than 2 records are stored in the control interval.

Control interval size = 512 bytes
Records of the same length = 80 bytes (only 2 RDFs (R a) in figure below needed).

LR 1	LR 2	LR 3	LR 4	LR 5	LR 6	f.s.	R a	R a	CIDF
80	80	80	80	80	80	22	3	3	4

Control interval size = 512 bytes
For records of different length 1 RDF per LR is used
(R 1 for LR 1, R 2 for LR 2, etc.)

LR 1	LR 2	LR 3	LR 4	LR 5	f.s.	R 5	R 4	R 3	R 2	R 1	CIDF
130	70	50	140	60	43	3	3	3	3	3	4

Control interval size = 512 bytes
Records of different length (for each group of adjacent records with the same length, 2 RDFs are used
(R a for LR 1-4, R^5 for LR 5, and R^6 for LR 6)

LR 1	LR 2	LR 3	LR 4	LR 5	LR 6	f.s.	R^6	R^5	R a	R a	CIDF
80	80	80	80	58	55	63	3	3	3	3	4

Legend:
LR = Logical Record
f.s. = free space
R x = Record Definition Field (3 bytes)
CIDF = Control Interval Definition Field (4 bytes)
The small numbers represent the length in bytes of data/control fields

Figure 3. Control intervals with fixed and variable-length records

Each control interval is divided into one or more fixed-length physical records and VSAM determines their size based on the control interval size and the device characteristics. This physical record is dependent on the control interval size and the device type and is purely a consideration when mapping control intervals to the storage device. A physical record is that record seen on the disk and its size is equal to the control interval wherever possible.

By defining the CI-size for a data set stored on a specific type of direct access device, the user implicitly chooses the physical record size (for further details see page 196).

The following figure shows the relationship between user defined control interval size and VSAM chosen physical record size on different DASD types. In this example it is assumed the control interval size is 6K. As a control interval must be stored in an integral number of physical records the next lower value is 2K. So VSAM chooses 2K as physical record size for the 3330. Since 2K is not used on 3340 (would result in only 75% track utilization) 1K is choosen for 3340. Page 196 contains a chart showing the physical record sizes and the number of records per track for each supported DASD type.

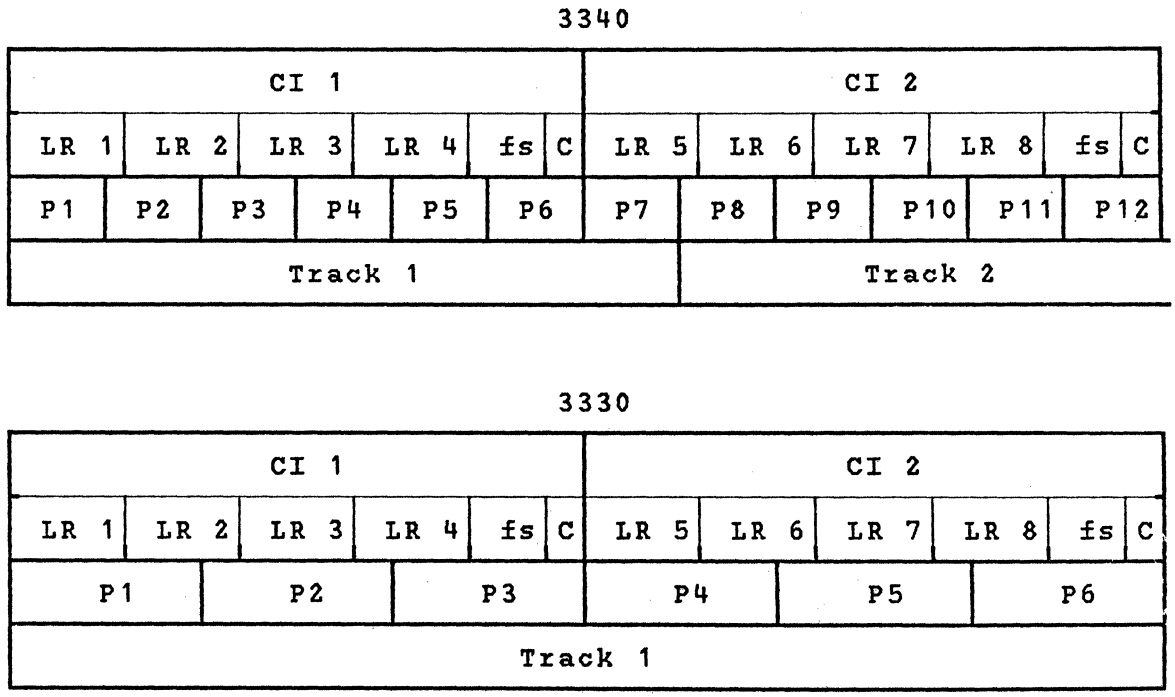


Figure 4. Logical to physical records relationship

With a single GET, one or more control intervals may be transferred between a VSAM data set and VSAM buffers in main storage (depending on direct or sequential access). Command-chained reads/writes are used when a control interval contains more than one physical disk record. From the VSAM buffer, the desired logical record (without control fields) is transferred to the user buffer (input area). For more information about specifying VSAM buffers, see section 8.1.5 on page 190.

A logical record in a key-sequenced data set or in an entry-sequenced data set (the data set organizations are described later) can span two or more control intervals within the same control area, in which case it is called a spanned record (see detailed description in section 2.2.3 on page 11).

The physical location of a logical record within a data set is given in the form of a relative byte address (RBA) rather than a CCHHR disk record address.

The RBA of a logical record is the offset of this logical record from the beginning of the data set. The first record in a data set has an RBA of 0. The second record has an RBA equal to the length of the first record and so on. The RBA of a logical record or index entry, therefore, is independent of the physical characteristics of the direct access device type on which it resides, the number of extents in the data set, etc. It only depends on its position in the sequence of records. This way of referencing data by RBA makes the data sets independent of the DASD device type.

The RBA is always expressed as a full word binary integer.

The RBA includes free space and control information (figure 8 on page 14 shows an example how RBA values are assigned).

2.2.2 CONTROL AREAS

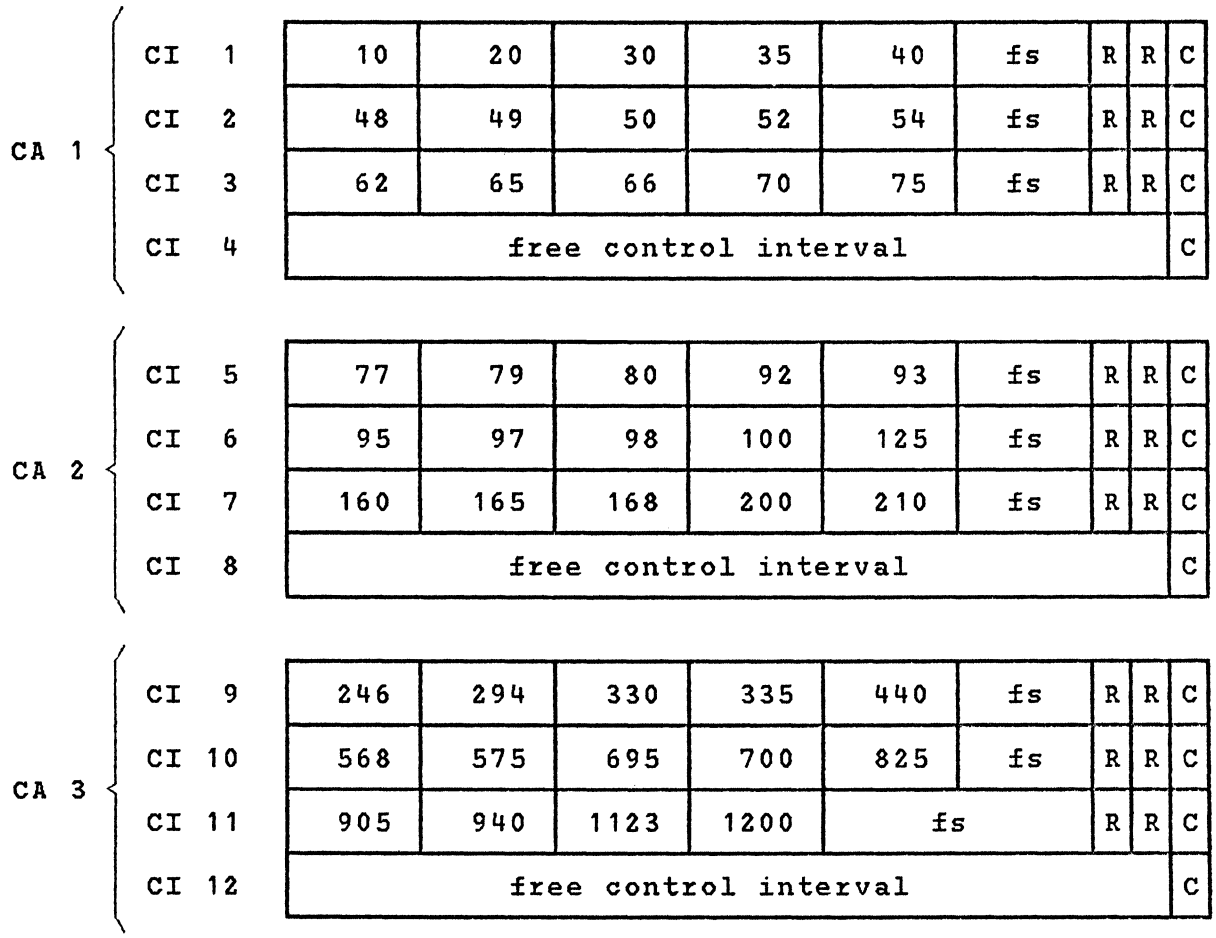
The control intervals in a VSAM data set are grouped together into fixed-length contiguous areas of direct-access storage called Control Areas (CAs). A control area is auxiliary storage space that is set aside by VSAM to receive or hold the control intervals of a particular data set.

The user does not have to specify the number of control intervals per control area of a data set; VSAM determines this number. Whenever the space in a data set needs to be extended because the control area cannot hold anymore data, VSAM extends the data set (see also description of VSAM data spaces and space allocation on page 40, 45, 189).

The maximum size of a control area is one cylinder and the minimum size is one track of a DASD. The user implicitly defines the control area size by specifying the amount of space to be allocated to a data set (for further information, see section 8.1.3.2 on page 187). Usually better performance is obtained when the size of one control area is one cylinder.

Figure 5 shows a possible structure of three control areas, used in the data component of a key-sequenced data set (KSDS) (the data set organizations are explained in the next section), which include four control intervals each.

All details, like 'free space', etc., are explained in following sections.



C = Control Interval Definition Field
CA = Control Area
CI = Control Interval
fs = free space
R = Record Definition Field

Figure 5. Structure of a VSAM control area (KSDS)

2.2.3 SPANNED RECORDS

If the data records are larger than the control interval size they need not be broken apart or be reformated; it can be specified (with the SPANNED parameter in the Access Method Services DEFINE command, which is described later) that they are allowed to extend across or span control interval boundaries.

Such records are called spanned records. Spanned records are applicable only to KSDS and RRDS data sets.

A spanned record must always begin on a control interval boundary; it fills one or more control intervals within a single control area. As shown in figure 6, the control interval that contains the last portion (segment) of a spanned record contains unused space if the segment does not completely fill the data space available in the control interval. This unused space can only be used to extend the spanned record; it cannot contain all or part of any other record.

If the parameter SPANNED was specified at DEFINE time (this parameter cannot be changed with ALTER; described later), VSAM determines dependent on the length of each record whether it is a spanned or a nonspanned record. If the record length is larger than CI size minus 7 (RDF+CIDF) the record is stored as spanned record starting in a new CI (this is also checked when a record is extended).

If spanned records are used for KSDS, the primary key must be within the first control interval.

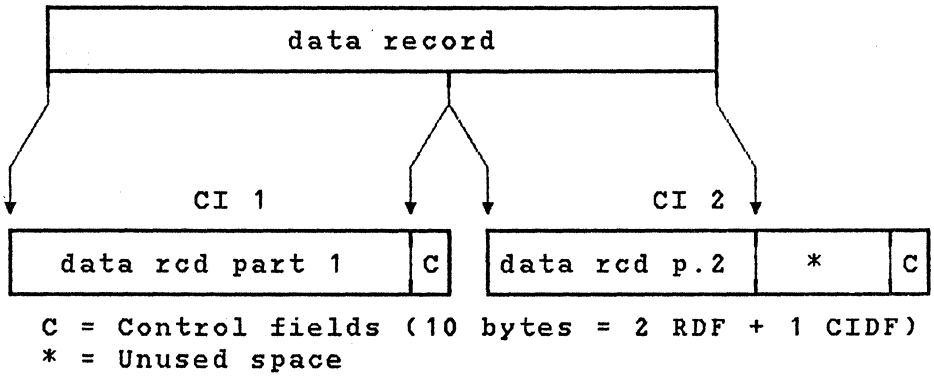


Figure 6. Structure of a spanned record

2.3 TYPES OF DATA SETS

VSAM supports three different data set organizations: key-sequenced, entry-sequenced, and relative-record. All three organizations allow both sequential and direct processing and record addition without rewrite of the complete data set. The primary difference between these three organizations is the sequence in which logical records are stored.

A key-sequenced data set (KSDS) uses logical records, either fixed or variable in length, which are placed in the data set in ascending collating sequence by a field, called the key.

To differentiate keys used in VSAM objects the key used in a base cluster (explained on page 29) is called a primary key or base key. The key used in an alternate index (explained on page 29) is called an alternate key.

Records added after the KSDS is created are inserted in primary key sequence and existing logical records are moved when necessary. In VSAM key-sequenced organization, a record must have a unique, embedded, fixed-length primary key located in the same position within each logical record. Primary keys can be a minimum of one byte and a maximum of 255 bytes.

To differentiate indexes used in VSAM objects the index in a KSDS base cluster is called the primary index (usually, the term 'index' in this manual means 'primary index'). A primary index is always created for a KSDS.

An optional index which can be built over an ESDS or KSDS base cluster is called an alternate index (the alternate index is described on page 29). The alternate index enables logical records to be accessed sequentially or directly through a different field. An alternate index is organized like a KSDS.

Figure 7 shows a logical picture of a KSDS with one control area which includes four control intervals (physically the entries in an index control interval are stored from right to left):

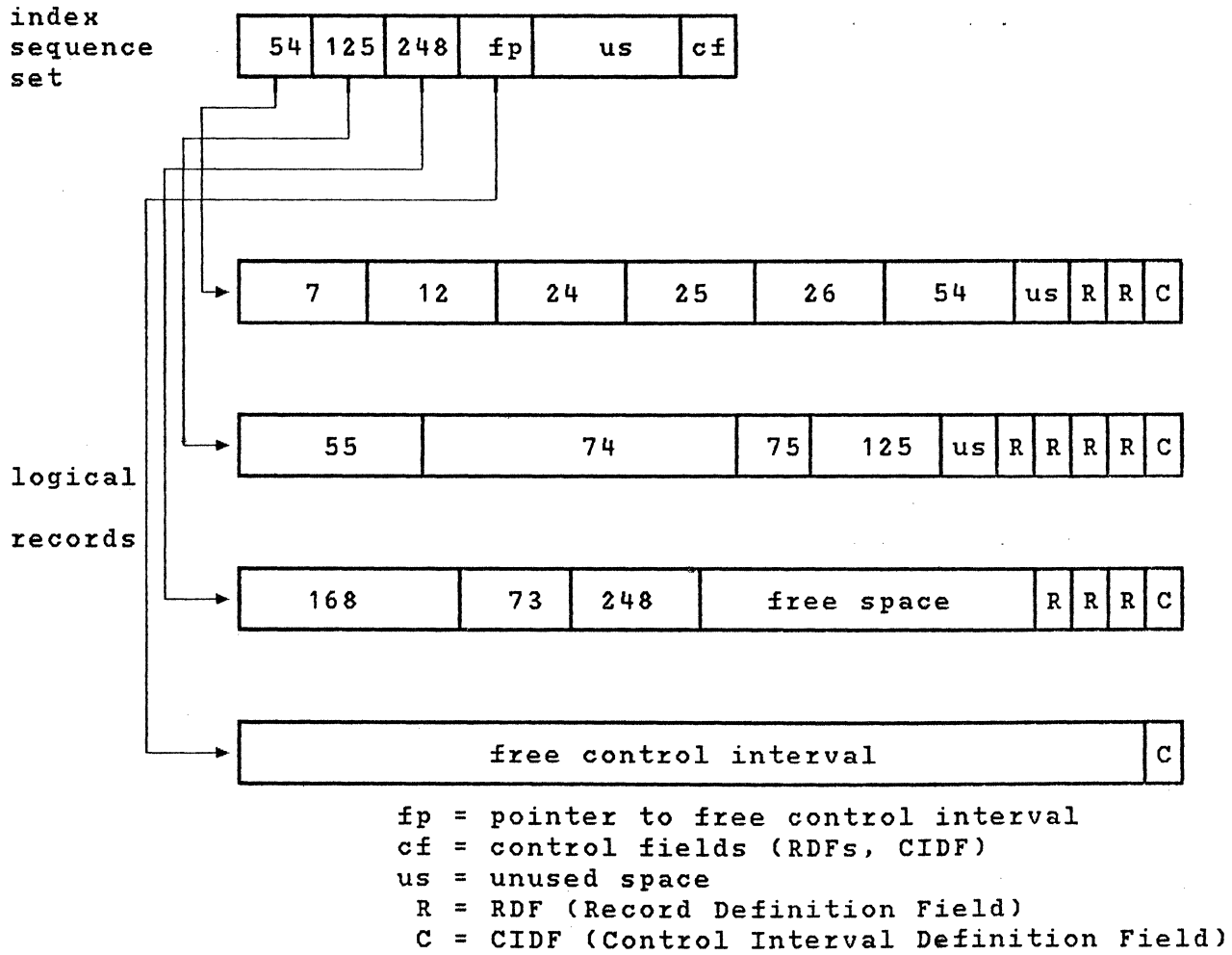


Figure 7. Key-sequenced data set structure

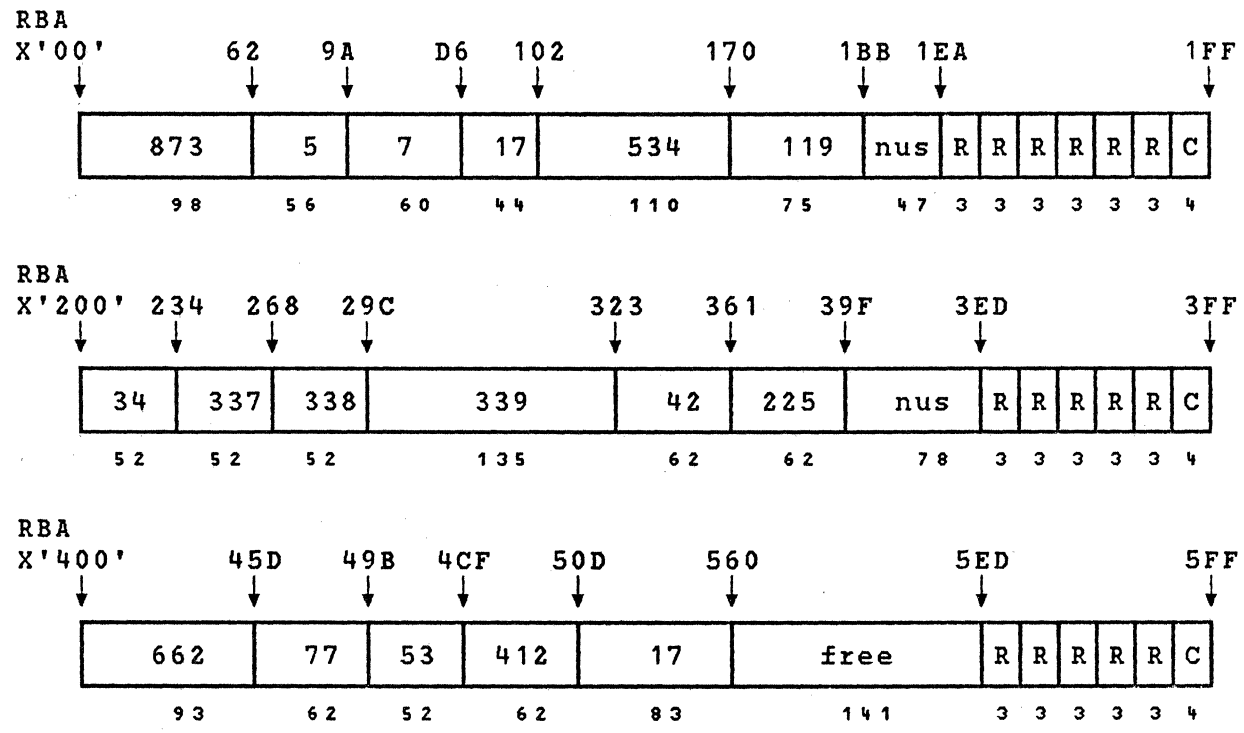
The KSDS and its components are described in more detail starting on page 18.

An entry-sequenced data set (ESDS), which is logically comparable to a SAM data set, contains either fixed- or variable-length records, sequenced in the order in which they were submitted for inclusion in the data set (like SAM data sets). Therefore, records are sequenced by their time of arrival rather than by any field in the logical record.

An ESDS accepts records of fixed and variable length, spanned or nonspanned records (spanned records are described on page 11). Records can be added at the end, and records can be updated but not extended. Records cannot be erased.

Figure 8 shows a possible structure of an ESDS. The RBA values are in hexadecimal (RBA values are always presented in a 4-byte field).

The numbers in the CIs represent keys in the records (not used when loading the data set or as search arguments). The control interval length is 512 bytes. The small numbers below the CIs represent the record/field length in bytes.



C = CIDF (Control Interval Definition Field)
nus = unusable space (remaining space)
R = RDF (Record Definition Field)
RBA = Relative Byte Address (see page 10)

Figure 8. Entry-sequenced data set structure

A relative record data set (RRDS) is processed by a relative record number and consists of a number of non-spanned fixed-length slots, each of which has a unique relative record number, an associated RDF, and contains one logical record. The slots within an RRDS are sequenced by ascending relative record number (from 1 to N).

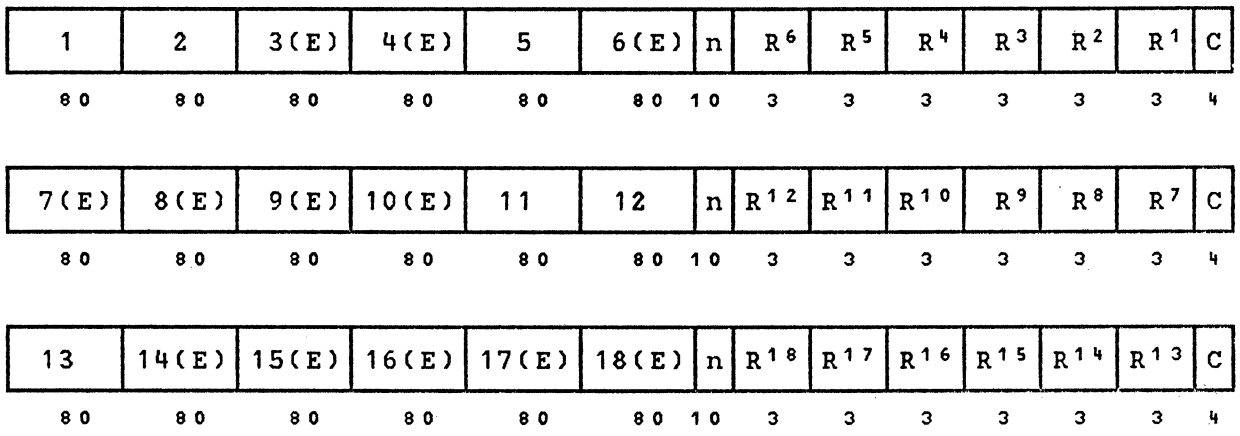
A record is placed in the slot specified by a user-supplied relative record number (for direct or skip-sequential inserts) or, if not specified, by a VSAM-supplied relative record number (for initial load or sequential inserts). This relative record number is never included in the logical record by VSAM.

Records can be retrieved by relative record number (direct, skip-sequential) or by physical sequence). Insertion and processing techniques are explained in section 2.6 on page 22.

In an RRDS one RDF (3 bytes) per slot is used, and Bit 5 in the first byte of each RDF determines, if a slot is 'occupied' (0) or 'empty' (1).
An RRDS cannot have any index. No free space definition is allowed.

Figure 9 shows a possible structure of an RRDS after loading a few relative records. Each slot has an RDF which includes the information indicating whether the slot is 'occupied' or 'empty' (see also section 2.6.3 on page 28). The numbers in the fields represent slot numbers (relative record numbers) containing records. The small numbers below the fields show the length of each field.

When adding a record and specifying a slot number which is higher than any slot number used so far, the data set is formatted automatically to the end of the control interval receiving the record.



(E) = empty slot n = unusable space (remaining space)
R^{1-18} = RDFs (Record Definition Fields) of the corresponding slots
C = CIDF (Control Interval Definition Field)

Figure 9. Relative record data set structure

A VSAM cluster is a logical definition for a VSAM data set and has one or two components.

The data component of a VSAM cluster consists of all control areas containing the control intervals which contain the data records.

The index component of a key-sequenced cluster consists of the index control intervals containing the index records which each points to the highest keys of a data control interval (see also section 2.4 on page 18).

The VSAM cluster consist of the following:

- KSDS cluster: - index component
  - data component
- ESDS cluster: - data component
- RRDS cluster: - data component

A VSAM data set is always represented in a VSAM catalog by a cluster entry, a data entry and (for a KSDS only) an index entry. All parts of a multivolume data set component must reside on the same device type.

2.3.1 REUSABLE DATA SET

Enhanced VSAM allows the use of VSAM data sets as Workfiles.

A non-reusable data set may be loaded once and only once. After the data set is loaded, it can be read, written into, and the data in it can be modified. But when the data becomes obsolete, the way to remove it, is to use the Access Method Services command DELETE which deletes the data set (DELETE is described on page 61). If the data set is to be used again it must be redefined with the Access Method Services command DEFINE (described on page 61).

Instead of using the DELETE - DEFINE sequence the REUSE function can be used. In order to specify a data set to be reusable, the REUSE parameter must be specified in the DEFINE CLUSTER|ALTERNATEINDEX command (these commands are described later in this manual).

A reusable data set may be a suballocated KSDS, ESDS or RRDS (see description on page 40) that resides on one or more volumes.

To explain the function of a reusable data set, another VSAM term must be explained. As mentioned earlier (page 10) the size of a VSAM data set is represented by RBA values. VSAM uses a so called high-used RBA field to determine if a data set is empty or not. After a data set is defined with a DEFINE CLUSTER command (described later) the 'high-used RBA' value is zero. After any loading and closing the data set this field shows the offset of the last byte in the data set (see explanation of LISTCAT printout on page 156).

In a reusable data set (i.e. a data set defined with the REUSE parameter) this 'high-used RBA' field can be reset to zero and VSAM can use this data set like a newly defined data set.

The ways to 'reset' a reusable data set are as follows:

1. Use the REUSE parameter in the Access Method Services command REPRO when reloading the data set.

2. Specify in the ASSEMBLER user program ACB the parameter MACRF=RST (see macro description on page 217 and figures 47 and 48 on pages 212 and 215).

3. Use High Level Languages (such as PL/I or COBOL/VS) and open the reusable data set for OUTPUT.

If a reusable data set is opened without specifying a 'reset' function the data set with its data is used like any other non-reusable data set.

When a reusable data set is opened and one of the 'reset' functions is specified, all secondary extents are unallocated and the data set 'high-used RBA' is set to zero. In addition the data set is not 'erased' (i.e. overwritten with X'00') during a 'reset'.

Restrictions:

1. KEYRANGES cannot be specified (see description on page 91).

2. Alternate indexes (see page 29) cannot be defined over a cluster with the REUSE attribute.

3. The data set cannot have more then 16 extents per volume.

4. The data set cannot be a UNIQUE cluster (described on page 40).

5. The Access Method Services command ALTER cannot be used to change the data set attribute from REUSE to NOREUSE or vice versa.

6. Reusable data sets on 3850 (MSS) must begin on cylinder boundary to prevent staging in RESET mode.

2.4 KSDS PRIMARY INDEX STRUCTURE

A KSDS consists of an index (index component) and logical records (data component) with different data set (component) names. All extents of the KSDS data component must reside on the same type of direct access devices. The index component, however, can be placed on a device type different from that of the data component.

The index for a VSAM KSDS data set contains key values and pointers. It is built when the KSDS is initially loaded. A VSAM index also contains information about available space in the index component.

Although it is an internal VSAM function it should be mentioned that the key in an index entry is stored in a compressed form. This is called key compression (for detailed description of key compression see section 8.6 on page 203). For index calculations (see page 198) an average compressed key length of 3 can generally be used.

In most cases a VSAM index consists of 2-3 index levels. The lowest level is called the sequence set. All levels above, which are automatically built by VSAM when necessary, are collectively referred to as the index set.

In the sequence set, there is one index record per data CA. Each index record is designed to contain a number of index entries equal to the number of data CIs which fit in its CA.

An index record (equivalent to one index control interval) is fixed in length, either 512, 1024, 2048 or 4096 bytes (index CI sizes are described on page 195)).

Each index record contains a pointer to an index record in the next lower index level or to a control interval in the data component.

An index entry (in the index record) consists of the highest key associated with that CI (in compressed form), control information (for expanding the compressed key) and a displacement value to be added to the RBA of the beginning of the CA. (derived from the index header). The sum of which addresses the correct data CI.

If a data set consists of more then one CA, more than one index sequence set level record (CI) exists. In this case VSAM automatically builds another index level (L2). Each entry in the index L2 record points to one Sequence Set record (L1) (see figure 10).

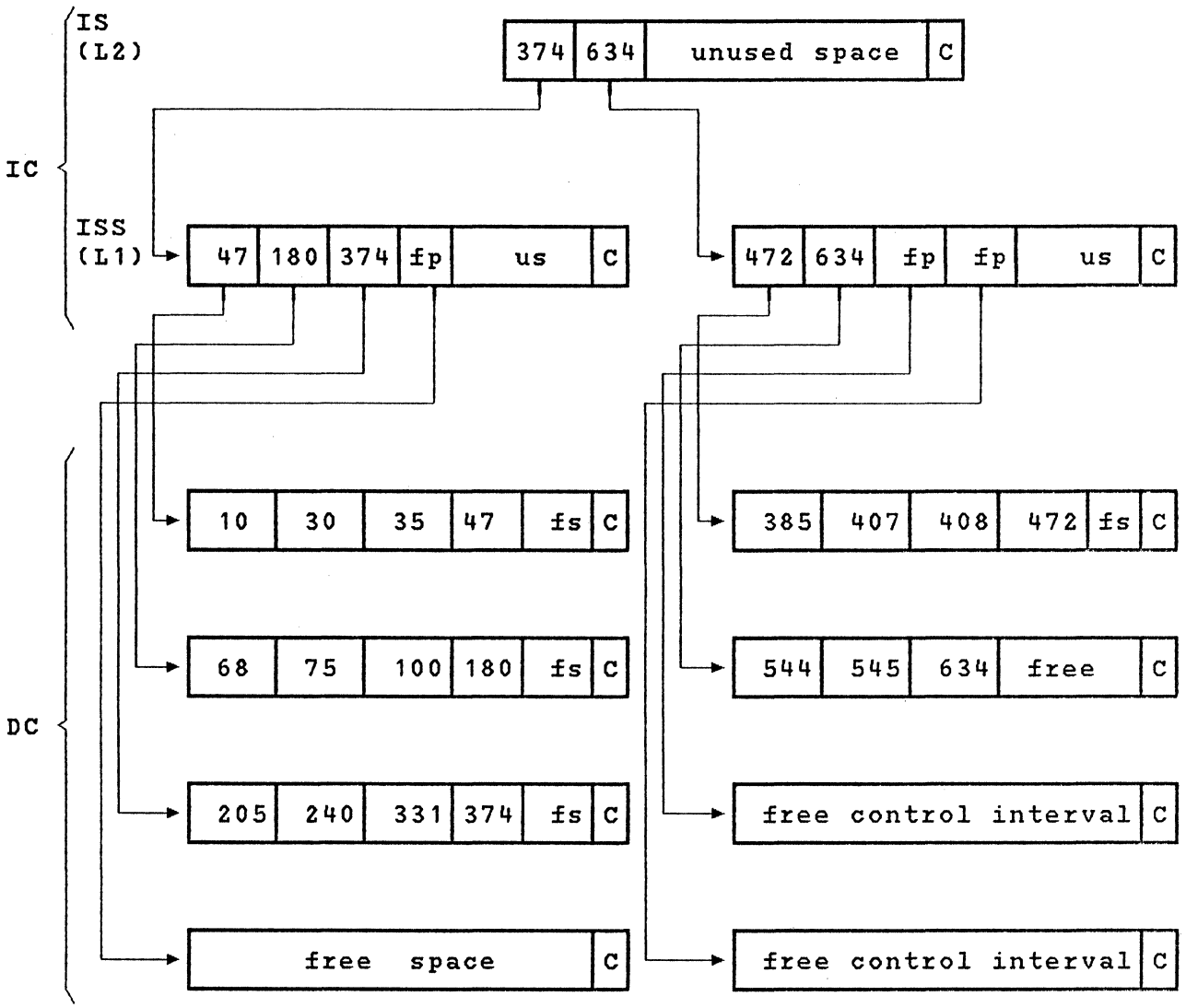
If the data set is that big that the index L2 record cannot hold all entries for all the Sequence Set records, another index L2 record has to be built by VSAM.

Since the highest index level can only consist of one index record, VSAM will build another index level (L3), etc.

For index performance considerations see section 8.1.4 on page 190

Figure 10 shows a logical picture of a KSDS after loading with two CAs, containing four CIs each. Records with 80 bytes each were loaded. A free space value of 20% free space in the CI and 25% in the CA was specified (see page 88), so 1 record (minimum = (512 * 20%) = 102) per CI and one CI per CA (25% * 4) (CI/CA) is held free when loading the data set. In each CI with a length of 512 bytes and control fields with a total of 10 bytes, 4 data records were loaded (102 bytes were left free).

The index component consists of two levels: the Index Sequence Set and the Index Set.



C = control fields (RDFs, CIDF)
DC = Data Component
fp = pointer to free CI
fs = free space (FREESPACE def.)
IC = Index Component (IS + ISS)
IS = Index Set
ISS = Index Sequence Set
L1,L2 = Index Level 1 and 2
us = unused space

Figure 10. Key-sequenced data set structure

2.5 INSERTING, EXTENDING AND DELETING OF RECORDS

A KSDS can be defined to reserve space for later insertions when the data set is loaded (or sequential insertions performed). This reserved space is called free space.

The amount of free space in a CI and/or CA (in %) is specified in the FREESPACE parameter of the DEFINE CLUSTER command (see also page 88, and example on page 106).

When a record must be added to a control interval, records with keys higher than the one to be inserted are shifted to the right of the CI within the free space limit. As long as enough free space is available in this CI, there is no further data CI manipulation.

As described in the following sections (starting section 2.6 on page 22) VSAM allows three types of access to insert, delete and extend records into a KSDS:

- Direct Access
- Sequential Access
- Skip-sequential Access

The Direct Access logic is described in the following paragraphs.

Sequential Access is used for loading data sets and for insertion of groups of pre-sorted records and is also called 'Mass Insertion'. Mass insertion is described in detail in section 8.8 starting on page 207.

Skip-sequential Access is described in section 2.6.1.2 on page 23.

The following examples illustrate the usage of free space:
This CI is 512 bytes long. A free space value of 30% has been specified. With a fixed record length of 80 bytes, the maximum CI capacity is 6 records. But after loading the data set, each CI contains only 4 records and 182 bytes of free space (VSAM calculated a minimum of 153 bytes free space).

10	20	30	40	free space	R^2	R^1	C

- Record 25 is to be inserted. The control interval is read into a VSAM buffer and records 30 and 40 are shifted to the right.

10	20	25	30	40	fr.sp.	$*R^2$	R^1	* C

- Records 20 and 40 are to be deleted. The remaining records are shifted to the left if necessary and the freed space is now available for inserts, updates, etc.

10	25	30	free space	$*R^2$	R^1	* C

- Record 10 must be expanded to 148 bytes.

10	25	30	free space	$+R^3$	$*R^2$	$*R^1$	* C

R^1-R^3 = RDFs (Record Definition Fields)
C = CIDF (describes free space and offset)
* = value changed
+ = new control block

Note: Not one of these changes require a change in the related index sequence set, even if the high key record in a CI is deleted.

Figure 11. Inserting, deleting and extending records

If the record to be inserted will not fit in the control interval, a control interval split takes place to provide sufficient space in the control interval to contain the record. The control interval split is described in detail in section 8.5 on page 201.

2.6 PROCESSING OF DATA SETS

2.6.1 KSDS PROCESSING

The following request types are allowed for KSDS processing:

- add/load records
- insert records (between existing records)
- modify existing records
- retrieve records
- extend/shorten existing records
- delete/erase records

As described in the following sections the records in a KSDS can be processed sequentially, skip-sequentially, or directly using the primary key as search argument. This is called keyed processing and it is the normal way to process a KSDS.

Another method of processing a KSDS directly or sequentially (but not recommended) is addressed processing using an RBA as search argument (see page 26). When a split occurs, or when records are added, deleted or changed in size, RBAs of some records will change, and so addressed access is not suggested for normal use.

One other way of processing a KSDS is control interval processing, which is used only for very specific applications. For further information see section 5.2.4 on page 74.

2.6.1.1 KSDS KEYED SEQUENTIAL PROCESSING

Keyed sequential processing is used to load a KSDS and to retrieve, update, delete and add logical records to an existing KSDS. When keyed sequential processing is used, records can be processed in ascending or in descending sequence by primary key. When retrieving records, key values need not be user-supplied, since VSAM automatically obtains the next logical record in sequence. Following types of operations can be performed using keyed-sequential processing:

- Records can be processed in ascending primary key sequence. This is called forward processing.

- Records can be processed in descending primary key sequence. This is called previous record processing or backward processing.

- A mass sequential insertion technique is used by VSAM when additions of contiguous records are sequenced and sequential processing is used. More information is included in section 8.8 on page 207. See also section 2.6.1.6 on page 24, which contains an overview of keyed sequential processing logic.

For keyed-sequential processing the index sequence set is used to find the next logical control interval.

Search arguments are not required or used.

2.6.1.2 KSDS KEYED SKIP-SEQUENTIAL PROCESSING

Keyed skip-sequential processing, is a variation of direct processing. It can be used for retrieval, update, addition and deletion operations. The logical records must be processed in ascending sequence without pre-positioning. Only the sequence set of the primary index is used for skip-sequential processing based on the primary key.

When a relatively small number of transactions that are in primary (or alternate) key sequence and clustered in sequence are to be processed, skip-sequential processing can be used to retrieve the records directly by key.

Skip sequential processing can be used to avoid retrieving the entire data set sequentially to process a relatively small percentage of the total number of records, or to avoid using direct retrieval of the desired records, which causes the primary index to be searched from the top to the bottom level for each record, but skip-sequential does read all sequence set records between those in question, therefore it is not suggested to use skip-sequential, when a record from the beginning and one from the end of the data set has to be processed.

See also section 2.6.1.6 on page 24, which contains an overview of keyed skip-sequential processing logic.

2.6.1.3 KSDS KEYED DIRECT PROCESSING

With keyed direct processing records can be retrieved, updated, deleted and added. A key value must be presented by the user for each logical record that is to be processed. For retrieval operation, the supplied key can be the full key, or a generic key (the leftmost part of the key) which matches exactly or is greater then the key of the desired record, and the record retrieved can have the exact or next greater key. The primary index is searched from the top to the bottom level to locate the requested logical record. See also Section 2.6.1.6 on page 24, which contains an overview of keyed direct processing logic.

2.6.1.4 KSDS ADDRESSED DIRECT PROCESSING

Addressed direct processing is not suggested to use for a KSDS, since RBA values of many records may change when a control interval or control area split occurs.

2.6.1.5 KSDS CONTROL INTERVAL PROCESSING

Control interval processing should be used only for very specific or unusual applications. For further information, see section 5.2.4 on page 74.

2.6.1.6 KSDS OVERVIEW OF KEYED PROCESSING

The following demonstrates the three suggested types of KSDS access. Refer to figure 12 where records to be processed are marked with an asterisk '*'.

- Keyed sequential access:

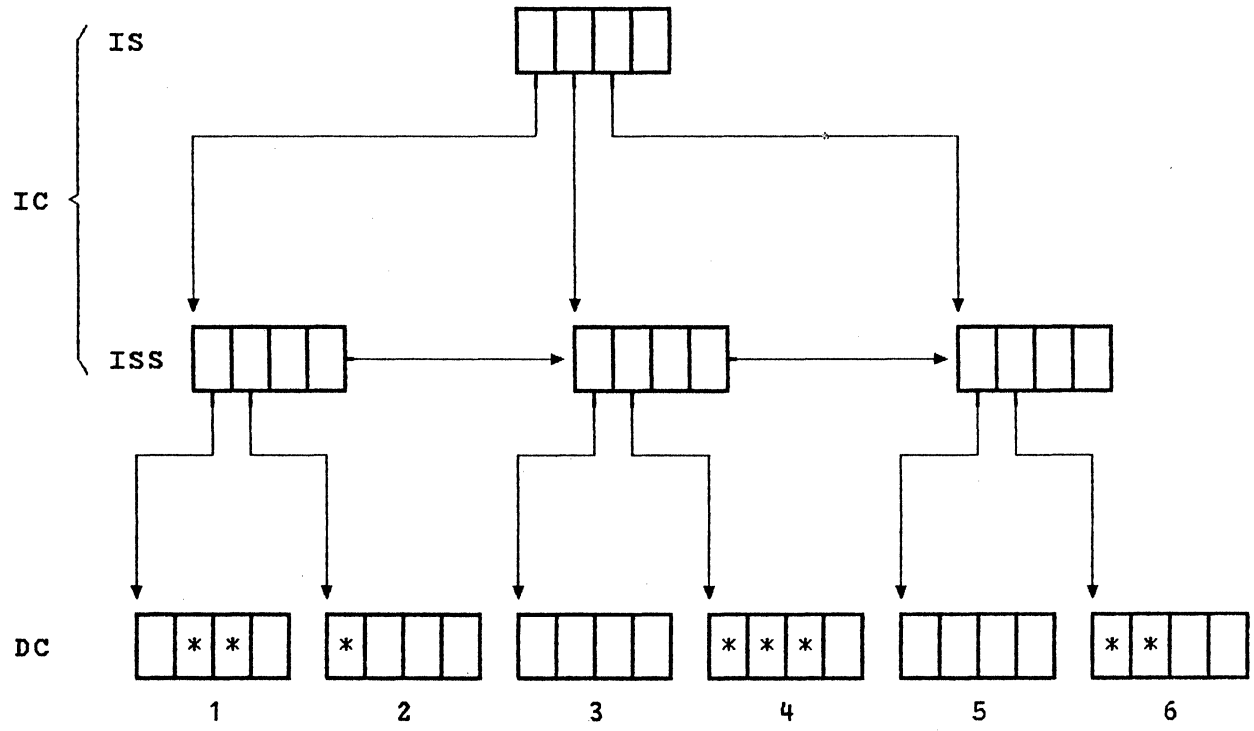
  For retrieval, as keys (search arguments) are not required (only used for positioning) all 6 CIs have to be read and all records returned to the user, although CIs #3 and #5 do not contain any requested records. The Sequence Set is used to find the next logical control interval.

- Keyed skip sequential access:

  The search arguments must be presented in ascending sequence. Here only CIs #1, #2, #4 and #6 are read. For each request the Sequence Set is used to find the next logical control interval and to check whether it contains the requested record.

- Keyed direct access:

  For each request a search argument must be presented in any order, and one record (control interval) from both the Index Set and the Sequence Set, and also the control interval containing the record must be read (not if the appropriate Sequence Set record and/or the data control interval was/were already read into the buffers by the previous operation and are still valid).



DC = Data component (data control intervals)
IC = Index component
IS = Index Set (highest index level)
ISS = Sequence Set (lowest index level)

Figure 12. Keyed sequential, skip-sequential, and direct processing.

2.6.1.7 POSITIONING FOR A KSDS

- Sequential processing:

  Sequential processing can be started from the beginning or somewhere in the middle of a data set. If processing is to begin in the middle of a data set positioning is necessary before sequential operation can be performed.

  Positioning can be done in two ways:

  - execute a POINT macro instruction (see page 221) supplying a key as search argument.

  - in a user ASSEMBLER program issue a 'direct' request, then change RPL (see page 218) with the MODCB macro (see page 219) from 'direct' to 'sequential' or 'skip-sequential'.

- Skip-sequential processing:

  If the first skip-sequential search is the first access after opening the data set, a direct search is initiated by VSAM to find the first record. From then on the index sequence set level will be used to find the subsequent records. If other operations were performed before (like read sequential, etc.), either the last position of that operation will be used as a starting point to search the Sequence Set records, or a re-positioning is necessary (see previous paragraph 'Sequential processing').

- Direct processing (keyed):

  No positioning is necessary since each direct request will be a top-down search through the index and access the CI directly.

2.6.2 ESDS PROCESSING

The following request types are allowed for ESDS processing:

- add/load records (only at the end)
- modify existing records (no record length change)
- retrieve records
- delete records (no erase; for 'mark delete' see the following section)

Addressed sequential, addressed direct, and control interval processing are supported for ESDS.

The former two are called addressed processing and it is the normal way to process an ESDS.

No keyed processing is allowed.

2.6.2.1 ESDS ADDRESSED SEQUENTIAL PROCESSING

For addressed sequential processing, the RBA given by the user is only used for positioning requests (see section 2.6.2.5 on page 27). VSAM automatically stores records in sequence of arrival and retrieves them in stored (physical) sequence.

When addressed sequential processing is used to process records in ascending RBA sequence, existing records can be retrieved, updated (but not changed in size) and marked deleted (not erased). 'Marked deleted' means the user can modify a bit or byte in a record field he designated as a 'delete bit/byte'. The user must provide support to prevent copying these records as VSAM does not recognize this 'delete bit/byte'

New records are only added at the end of the data set.

2.6.2.2 ESDS SKIP-SEQUENTIAL PROCESSING

Skip-sequential processing is not supported for ESDS.

2.6.2.3 ESDS ADDRESSED DIRECT PROCESSING

Addressed direct processing by user-supplied RBAs, can be used to retrieve records, update their contents (but not change their size) and flag records to be deleted (no physical deletion). New records cannot be added.

The user can obtain the RBA values of the individual records when loading the data set by issuing a SHOWCB macro instruction (see page 219) to obtain the 'RBA' field in the RPL (Request Parameter List, see page 218) just used (for more information see the "DOS/VS Macro Reference' manual or the 'OS/VS VSAM Programmer's Guide' as described on page 2).

2.6.2.4 ESDS CONTROL INTERVAL PROCESSING

Control interval processing should be used only for very specific or unusual applications. For further information, see section 5.2.4 on page 74.

2.6.2.5 POSITIONING FOR AN ESDS

- Sequential processing:

  Sequential processing can be started from the beginning or somewhere in the middle of a data set. If processing is to begin in the middle of a data set positioning is necessary before sequential operation can be performed.

  Positioning can be done in two ways:

  - execute a POINT macro instruction (see page 221) supplying a RBA as search argument.
  - in a user ASSEMBLER program issue a 'direct' request, then change RPL (see page 218) with the MODCB macro (see page 219) from 'direct' to 'sequential'.

- Direct processing:

  Direct processing can be accomplished when the correct RBA for the record is supplied by the user as search argument (see also previous section 2.6.2.3).

2.6.3 RRDS PROCESSING

The following request types are allowed for KSDS processing:

- add/load records
- insert records (between existing records into an empty 'slot')
- modify existing records (no record length change)
- delete/erase records

For RRDSs, keyed-sequential, keyed-skip-sequential, keyed-direct and control interval processing are supported. The relative record number is used as an argument for keyed processing. The former three correspond to keyed processing which is the normal way to process an RRDS.

No addressed processing is allowed.

An RRDS can be created using keyed sequential, skip-sequential, or keyed direct processing.

VSAM always uses the relative record number as search argument. It converts it to an RBA value and determines the control interval containing the requested record. If a record in a slot flagged as 'empty' is requested, a 'no-record-found' condition is returned.

A user cannot use an RBA value to address a record in an RRDS.

A request to add a record in a slot beyond the current end of file indicator causes VSAM to preformat the data set from the current end of file up to the control interval where the new record is supposed to reside. Loading an RRDS with keyed direct processing may result in long preformatting times if a record with a high number is loaded first.

2.6.3.1 RRDS SEQUENTIAL PROCESSING

RRDS sequential process is treated the same way as ESDS sequential processing (see section 2.6.2.1 on page 26) Empty slots are automatically skipped by VSAM.

2.6.3.2 RRDS KEYED SKIP-SEQUENTIAL PROCESSING

Basically skip-sequential is treated like an RRDS direct request, but the position is maintained. The buffer handling is treated like for sequential, i.e. the buffer is not written back after each write operation.

Records must be in ascending sequence.

2.6.3.3 RRDS DIRECT PROCESSING

An RRDS can be processed directly by supplying the relative record number as key. VSAM calculates the RBA and accesses the appropriate record/slot.

RRDS direct addressed processing (supplying RBAs) is not supported.

2.6.3.4 RRDS CONTROL INTERVAL PROCESSING

Control interval processing should be used only for very specific or unusual applications. For further information, see section 5.2.4 on page 74.

2.6.3.5 POSITIONING FOR AN RRDS

- Sequential processing:

  Sequential processing can be started from the beginning or somewhere in the middle of a data set. If processing is to begin in the middle of a data set positioning is necessary before sequential operation can be performed.

  Positioning can be done in two ways:

  - execute a POINT macro instruction (see page 221) supplying a relative record number (used as key) as search argument.

  - in a user ASSEMBLER program issue a 'direct' request, then change RPL (see page 218) with the MODCB macro (see page 219) from 'direct' to 'sequential' or 'skip-sequential'.

- Skip-sequential processing:

  No prepositioning is necessary for skip-sequential processing since it is treated like a direct request.

- Direct processing:

  No prepositioning is required.

2.7 ALTERNATE INDEX, PATH

Alternate indexes enable the logical records of an ESDS or of a KSDS (in this context called a base cluster) to be accessed sequentially and directly by more than one key field. This eliminates the need to store the same data in different sequences on multiple data sets for the purpose of various applications, or to re-sort the data (see figures 13 and 14 on page 33 and 34).

Alternate indexes are not supported for RRDS.

Any field in the base cluster record can be used as an alternate key. It may also overlap with the primary key (in a KSDS), or with any other alternate key. The alternate key field must be a contiguous field with the same offset in each record (in a spanned record, this field must be located totally in the first control interval).

Each alternate index consists of an index component and a data component. These two components together form an alternate index (abbreviated AIX) for the base cluster. Its structure is like KSDS. An alternate index will contain the different alternate key values of a certain alternate key. For every alternate key field a different alternate index is needed.

The Access Method Services program allows you to define, and then to create alternate indexes (if BLDINDEX command is specified by the user). An alternate index can be defined only after its associated base cluster has been defined, and it can be built only after its base has been loaded with at least one record.

The primary index component of a base cluster and any alternate index can be placed on a different device type than that of the base cluster data component. The primary index component and the alternate index need not reside on the same type of direct access device either. In addition, the index component of an alternate index can be on a different device than the data component of an alternate index (there is one exception for DOS/VS (especially for the device IBM 3330): should the data component of a UNIQUE KSDS reside on a 3330, the index component must reside on a 3330 as well).

Unlike the primary keys, which must be unique, identical alternate keys may occur in more than one logical record. This allows the search with a given alternate key to read all base cluster records containing this alternate key.

The BLDINDEX command causes a sequential scan of the specified base cluster, during which alternate key values and primary keys (for a KSDS) or record RBAs (for an ESDS) are extracted and put together to form alternate index records. These records are sorted by ascending alternate keys (no SORT program product is required). The alternate index records are then constructed and written.

There may be more than one primary key/RBA per alternate key. The primary keys/RBAs will be in ascending sequence within an alternate index record after loading (see figures 13 and 14 on page 33 and 34). But this fact will not necessarily be true after the base cluster has been updated (especially the alternate key fields) or new records added, as any new primary key/RBA will be inserted at the end of the appropriate alternate index record.

An alternate index is itself a KSDS. The data component of an alternate index is similar in physical format to the data component of a KSDS base cluster. The alternate key record contains system header information, and for a KSDS base cluster the alternate key value (not compressed), and the primary key field (not compressed) of the logical record in the base data set that contains the alternate key. For an ESDS base cluster, the alternate key record contains RBA values instead of primary key values.

The index component of an alternate index contains compressed alternate keys in ascending collating sequence and is physically and logically structured like a primary index (see page 18).

Alternate index maintenance can be handled by the user or automatically by VSAM. Automatic alternate index updating can be specified when defining the alternate index (see section 7.6.4 on page 110). Specifying the UPGRADE attribute for an alternate index makes it part of the upgrade set of alternate indexes for a given KSDS or ESDS.

An alternate index can be part of the upgrade set for a KSDS or ESDS even though it is not explicitly used. The maximum number of alternate indexes that can be part of the upgrade set for a given KSDS or ESDS is 125. Having alternate indexes in an upgrade set adds processing requirements, and therefore affects performance when inserting or deleting records, or updating alternate keys in a record in the base cluster.

2.7.1 ALTERNATE INDEXES FOR KSDS

Consider as an example a KSDS data set whose records contain customer number, address and the name of the salesman who takes the orders (see figure 13 on page 33). The customer number is used as primary key value. If the company plans a special sale in a city, they may want to write a note to all customers living in this area. Without having an alternate index, the whole data set must be read and all records containing the name of the city must be extracted. Or, if they want a summary of all the customers served by a particular salesman, the whole data set must be read to find all records with the salesman's name. However both of these requirements could be easily met by using alternate indexes (section 2.7.5 on page 36 explains how to access the base cluster records via the two alternate indexes).

Since using an alternate index is time-consuming (create, upgrade, etc.), it should only be considered for frequently used applications. Otherwise sequentially extracting and sorting records, running BLDINDEX with the REUSE option may be faster (the reusability of a data set must be specified when the data set is defined).

2.7.2 ALTERNATE INDEXES FOR ESDS

Normally the only way to access an ESDS directly is to store the RBA value of each record in a separate user data set. The RBA value can be obtained by issuing a SHOWCB macro instruction (see page 219) to obtain the 'RBA' field in the RPL (Request Parameter List; see page 218) just used.

One or more alternate indexes can be built for an existing ESDS when the record contains a field which can be used as alternate key value. An alternate index for an ESDS has the same structure as an alternate index for a KSDS base cluster. The only difference is that the alternate index records contain RBA values (4 bytes) instead of primary key values (see figure 14 on page 34).

2.7.3 EXAMPLES FOR ALTERNATE INDEXES

Figures 13 and 14 contain the structure of two alternate indexes connected to a KSDS and an ESDS respectively.

For the data sets, the following is assumed:

- The base cluster consists of 1 control area
- The control area contains 3 control intervals
- The record length in the base cluster is 320 bytes (X'140')
- The control interval sizes (CISIZE) for base cluster and alternate indexes were defined as follows:
  - Index control intervals = 512 bytes
  - Data control intervals = 1024 bytes

The JCL statements and the Access Method Services commands to create a structure like this (DEFINE, BLDINDEX etc.) are contained in chapter 7.0.

Notes to figure 13 and 14:

In figure 13 and 14 the control fields (RDFs, CIDF) and the unused space at the end of each CI are not shown. The records in the alternate indexes also contain header information, and the records are shorter than the records in the base cluster, so actually more records can be placed into a control interval.

VSAM places logical records in CI format, ensuring that (with the exception of spanned processing) each CI starts with the first byte of the logical record. In this example in each base cluster control interval 54 bytes (1024 - 3*320 - 10 bytes control information) remain unused.

Notes to figure 14:

The RBAs are always written as a full word binary integer (the RBA values in the figure were shortened due to limited space).

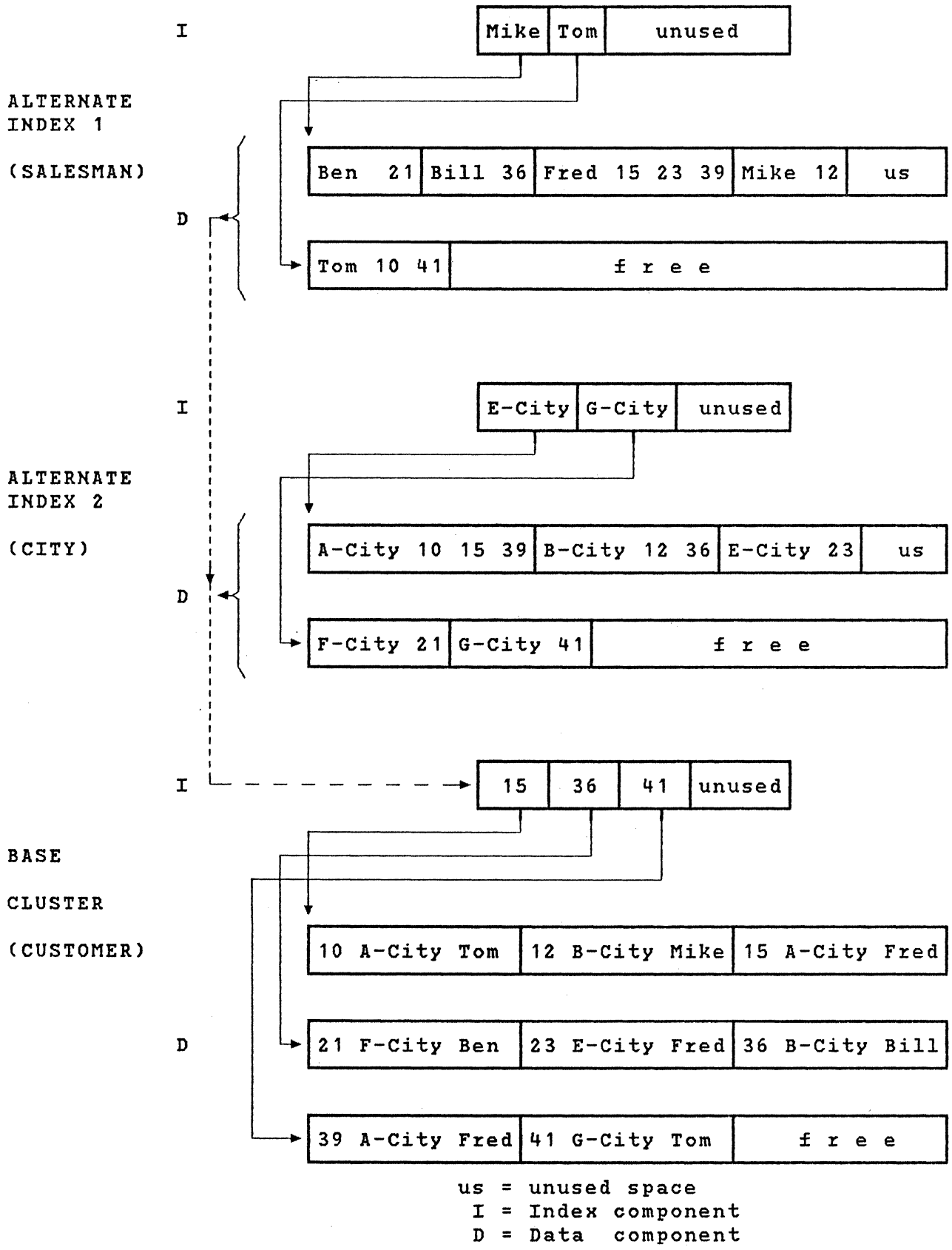
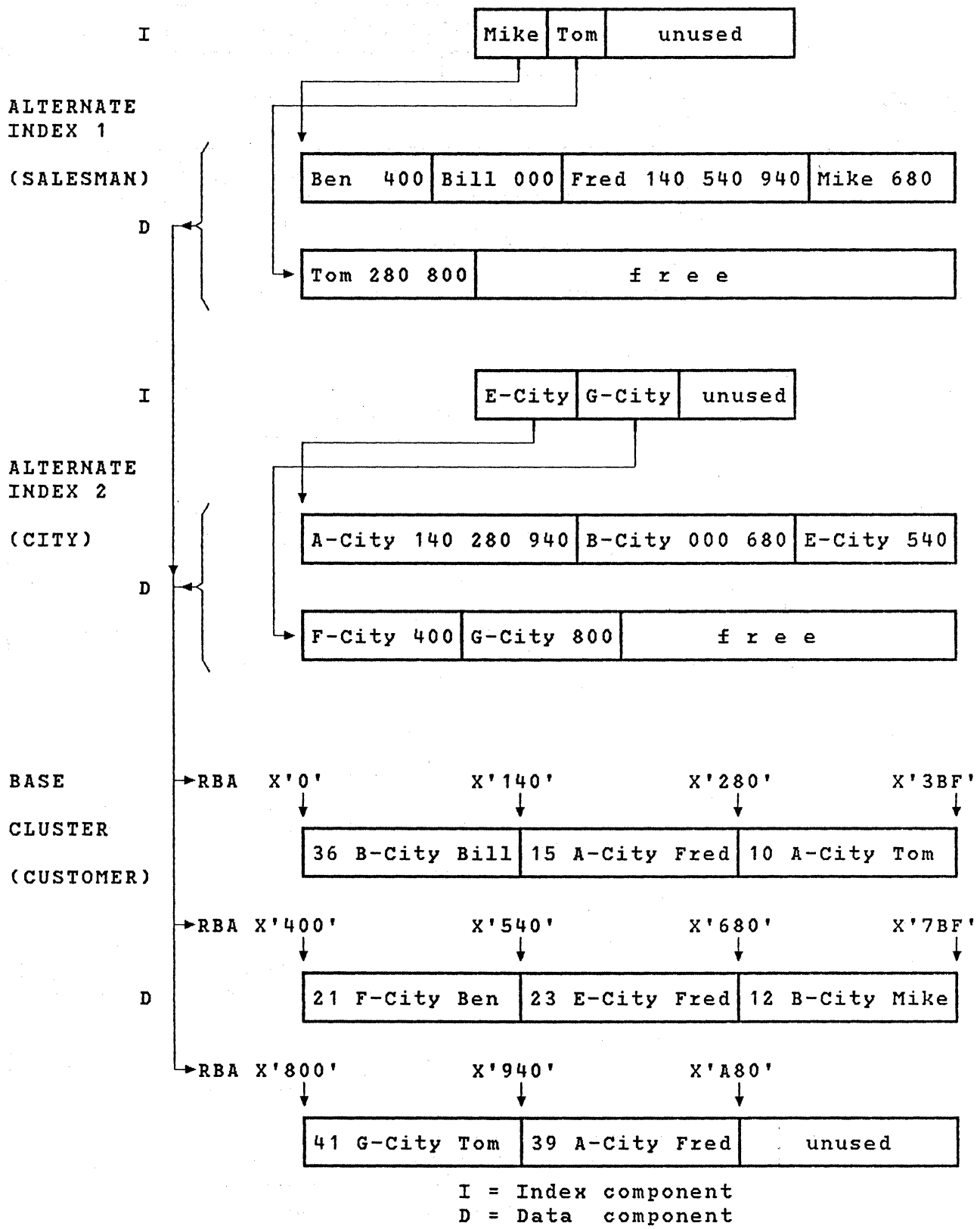


Figure 13. Alternate indexes with KSDS base cluster



Note: See notes on page 32.

Figure 14. Alternate indexes with ESDS base cluster

2.7.4 PATH

Before accessing a KSDS or ESDS (from now on both types of data set will be simply called the base cluster) via an alternate index, a path must be defined. A path is the means by which a base cluster is accessed by way of its alternate indexes. A path is defined and named using the Access Method Services program. At least one path must be defined for each of the alternate indexes through which the base cluster is to be accessed. When a given alternate index is to be used to process a base cluster, the associated path must be specified in an OPEN macro instruction. This causes both the base cluster and the alternate index to be opened.

If a path connects a base cluster with an alternate index belonging to the upgrade set (see page 31) of this base cluster, the parameter UPDATE/NOUPDATE can be defined for the path to allow/prevent VSAM to update the alternate index, when base cluster records are inserted, deleted or modified (if the alternate key changes). See also description on page 110.

In the following example the data sets were defined as follows: ALTERNATE INDEX 1 with the name SALESMAN, ALTERNATE INDEX 2 with CITY, and the BASE CLUSTER with CUSTOMER. Then PATH 1 was defined with the name SALECUST, and PATH 2 with CITYCUST (the JCL and control statements for this example are included in section 7.6.4 starting on page 111).

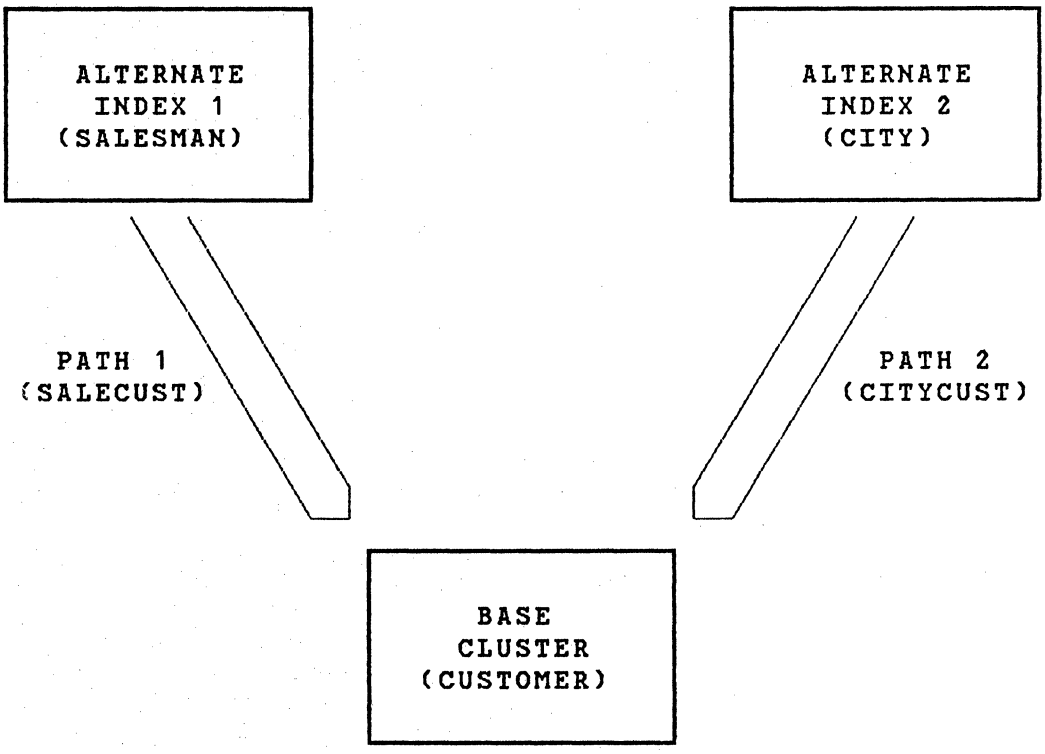


Figure 15. Alternate Indexes with Path

2.7.5 ACCESS TO BASE CLUSTER AND ALTERNATE INDEX W/WO PATH

The following simple examples show the function of paths and alternate indexes to accessing the base cluster (use figures 13 and 15 as a reference). The records read by each GET can be optionally placed by VSAM (RPL OPTCD=MVE) into the user's program input area.

All accesses are sequential with no search argument. AIX is the abbreviation for Alternate Index.

- Access the base cluster directly:

  OPEN base cluster CUSTOMER

  GET ——————→ 10 A-City Tom

  GET ——————→ 12 B-City Mike

- Access the base cluster via AIX1:

  OPEN path SALECUST

  GET ——————→ 21 F-City Ben

  GET ——————→ 36 B-City Bill

- Access the base cluster via AIX2:

  OPEN path CITYCUST

  GET ——————→ 10 A-City Tom

  GET ——————→ 15 A-City Fred

- Opening AIX1 by its data set name

  OPEN alternate index SALESMAN

  GET ——————→ Ben 21

  GET ——————→ Bill 36

- Opening AIX2 by its data set name

OPEN alternate index CITY

GET ⟶ A-City 10 15 39

GET ⟶ B-City 12 36

The following examples show the effect of the UPGRADE/NOUPGRADE attribute when inserting records into a KSDS cluster or into its alternate index (use figure 13 on page 33 as reference). When accessing the base cluster and the alternate index via a path, it is assumed the path was defined with the UPDATE attribute, which allows automatic updating of the alternate indexes if necessary.

Example 1: Insert Record '40 G-City Bill' into base cluster CUSTOMER.

- Access the base cluster CUSTOMER directly or with a path (UPGRADE defined for AIX1 and AIX2)

OPEN base cl. CUSTOMER (VSAM opens also AIX1,AIX2) (*='40')
or OPEN path SALECUST (VSAM opens base cl.,AIX1,AIX2) (*='Bill')
or OPEN path CITYCUST (VSAM opens base cl.,AIX1,AIX2) (*='G-City')
('*' is the search argument used for the insert)

PUT '40 G-City Bill' ➤ update of base cluster, AIX1 and AIX2.

Base before insert

39 A-City Fred	41 G-City Tom	f r e e

Base after insert

39 A-City Fred	40 G-City Bill	41 G-City Tom

AIX1 before insert

Ben 21	Bill 36	Fred 15 23 39	Mike 12	us

AIX1 after insert

Ben 21	Bill 36 40	Fred 15 23 39	Mike 12	us

AIX2 before after

F-City 21	G-City 41	f r e e

AIX2 after insert

F-City 21	G-City 41 40	f r e e

Example 2: Change Record '21 F-City Ben' into '21 G-City Ben'

- Access the base cluster CUSTOMER directly (UPGRADE defined for AIX1 and AIX2)
  OPEN base cl. CUSTOMER (VSAM opens also AIX1, AIX2) (* = '21')
  ('*' is the search argument used for the insert)

  GET '21' (this is a GET UPDATE to change a record)

modify the record (in the user input area)

  PUT '21 G-City Ben' (this is a PUT UPDATE to change the rcd).
  As a result the base cluster and AIX2 is updated
  (AIX1 is not updated, as Ben still points to 21).

Base before modification or update

21 F-City Ben	23 E-City Fred	36 B-City Bill

Base after modification or update

21 G-City Ben	23 E-City Fred	36 B-City Bill

AIX2 before modification or update

F-City 21	G-City 41 40	f r e e

AIX2 after modification or update

G-City 41 40 21	f r e e

Note: The pointer (keys or RBAs) in an alternate index record is always added at the end of the record and is not sorted between the existing pointers.

2.7.5.1 UPDATE RESTRICTIONS FOR BASE CLUSTER AND AIX

All data fields in the base cluster may be changed without restriction. When changing fields containing either the base key or an alternate key, the following restrictions apply:

- the base key (key of the base cluster index) cannot be changed, regardless of which index is used for access.
- an alternate key cannot be changed if the GET for UPDATE was done through the alternate index.
- an alternate key can be changed if the GET for UPDATE was done through the base cluster.

2.8 VSAM DATA SET COMPARISON

Figure 16 shows the differences between the three VSAM data set organizations. For further details see previous sections in chapter 2.0.

Function/Structure	KSDS	ESDS	RRDS
Records sequence :	primary key	arrival seq.	rel. rec. no.
Access is:	sequential or direct by key or RBA	sequential or direct by RBA	sequential or direct by rel.record no
Record format:	fixed variable spanned	fixed variable spanned	fixed - -
Defined free space used to:	insert change length	- -	- -
New records :	anywhere	at EOF (End of file)	at specified slot if empty
Change record length:	yes	no	no
Delete a record:	yes[1]	no	yes[2]
Reuse space of a deleted record:	yes[1]	-	yes[2]
Alternate Index :	yes	yes	no
Reusable File :	yes,if no AIX	yes,if no AIX	yes

[1] Deletion of a KSDS record results in physically adjusting logical records within a data CI.

An attempt to insert a record later:

- with the same key, will be added back into the CI, at the same place provided no other changes have occurred in the CI.
- with a key within the range of the CI will be added to the CI, utilizing the space made available by the previous deletion provided there is space for the logical record (and additional RDFs, if required).

[2] Deletion of an RRDS record results in that "slot's" data being set to binary zeros and its control fields being set to reflect it is 'not present'. Reuse of this space is determined by the user in selecting the relative record number of this slot for addition of a record.

Figure 16. VSAM data sets comparison.

2.9 VSAM DATA SPACES

VSAM data sets can coexist on the same disk with nonVSAM data sets. Space defined on a volume for exclusive use of VSAM is called a VSAM data space (for VSAM ownership concept see section 2.10 on page 43). In OS/VS a VSAM data space can consist of a maximum of 16 extents on a volume which need not be contiguous (no restriction in DOS/VS). A volume can contain multiple data spaces. The maximum size of a data space is one volume. Several data spaces on more than one volume may be treated logically as one data space (e.g. for suballocated multivolume data sets).

VSAM uses two different data spaces:

- The suballocatable data space
- The unique data space

A suballocatable data space can contain one or more data sets and a data set can occupy one or more suballocatable data spaces on one or more volumes. A suballocatable data space is created by an Access Method Services command DEFINE SPACE or DEFINE MASTERCATALOG/USERCATALOG (see chapter 7.0 later in this manual). All suballocatable data spaces on one volume are treated as one logical space, and it is possible to delete all empty suballocatable data spaces on a volume together. Individual suballocatable data spaces can not be deleted (see also section 2.9.2 on page 43).

A VSAM cluster in a suballocatable data space is called a suballocated cluster or a suballocated data set.

In this manual the expression 'VSAM data space/set' usually means a suballocatable data space or a suballocated data set.

A unique data space (also called non-suballocatable data space contains only one data set. This data set occupies only one non-suballocatable data space on a volume (with up to 16 extents (OS/VS only)), but multiple volumes can be used for this data set. A unique data space is automatically built when a cluster or alternate index is defined with the UNIQUE attribute (see page 98). DOS/VS users must specify a space value in the EXTENT job control statement (not in an Access Method Services parameter), since a unique data set occupies its own VSAM data space; the definition is similar to a DEFINE SPACE command (see DOS/VS notes below).

A VSAM cluster/data set in a non-suballocatable space is called a unique cluster or unique data set.

Unique data spaces cannot be shared with other VSAM data sets.

VSAM data spaces cannot be shared with nonVSAM data sets.

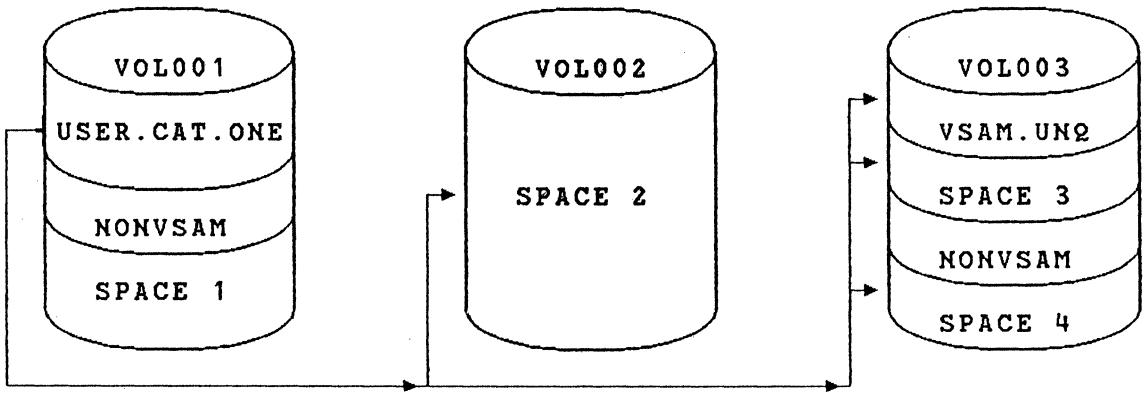
DOS/VS notes to unique clusters:

A data space or a unique cluster can not be extended (no DADSM in DOS/VS). A unique cluster can not be defined as first VSAM object on a volume in a recoverable catalog (see page 250).

Figure 17 shows examples for volumes containing a VSAM user catalog (USER.CAT.ONE) and data spaces.

SPACEs 1-4 are suballocatable data spaces created with DEFINE SPACE. The data set VSAM.UNQ was created with DEFINE CLUSTER with the parameter UNIQUE.

The VSAM user catalog USER.CAT.ONE contains information about the VSAM data spaces and the VSAM unique data set/data space. In OS/VS also the NONVSAM data sets could be cataloged and accessed via the VSAM user catalog.



This is a volume with a VSAM catalog, 1 VSAM suballocatable data space, and 1 NONVSAM data set.

This is a volume using the whole volume as one VSAM suballocatable data space.

This is a volume with 1 VSAM unique data space, 2 VSAM suballocatable data spaces, and 1 NONVSAM data set.

Figure 17. VSAM data spaces

2.9.1 CHOOSING SUBALLOCATABLE OR UNIQUE CLUSTER

The following is a list of advantages for the two types of allocation:

Advantages of SUBALLOCATION:

1. Since it is the default, there is no need to specify it on an Access Method Services DEFINE command.

2. If the suballocated cluster is on a different volume from the owning catalog, it is possible to DELETE the cluster without the cluster's volume being mounted (if a nonrecoverable catalog is used). This is because all the information which needs to be changed by the DELETE is in the catalog.

3. The REUSE attribute, which allows the Enhanced VSAM user to reload VSAM clusters without the need for DELETE/DEFINE, is only supported for SUBALLOCATION clusters.

4. DOS/VS: when using suballocatable cluster, VSAM space management allows dynamic space management otherwise not supported in DOS/VS. No EXTENT information is needed in JCL to specify begin and end of cluster.

   OS/VS: VSAM space management tries harder than DADSM to avoid fragmentation. When looking for a place to put a suballocated cluster, VSAM will try to find the smallest contiguous amount of free space which satisfies the cluster's primary allocation requirement, even if a more suitable allocation is possible elsewhere on the volume.

Advantages of UNIQUE:

1. There is only one VSAM cluster component per VTOC entry, with the possibility of making VTOC names the same as cluster component names. The organization of handling DASD space is the same as for nonVSAM data sets. VTOC related 'emergency' operations, such as SCRATCH and zapping of password bits, can be isolated to one VSAM cluster component.

2. On a volume with a large amount of VSAM space and a large number of clusters, use of UNIQUE allocation may result in better performance for DEFINE. This is because VSAM space management expends a lot more energy in making the best possible fit, for the DEFINE request (in DOS/VS EXTENT information for begin and end of data set must be specified in JCL, no VSAM space management needed). This may result in high CPU times and many I/O operations to the catalog for suballocated DEFINEs.

2.9.2 VSAM DATA SPACE AND CLUSTER NAMES

A VSAM data space defined with DEFINE MASTER/USERCATALOG or DEFINE SPACE (see section 7.3 and following sections starting on page 100) has no NAME parameter and can therefore not individually be deleted.

A unique cluster should be named with meaningful names for each CLUSTER, DATA and INDEX (KSDS or Alternate Index) to identify the components on a LISTVTOC or a LISTCAT list (see printout examples later in this manual).

A suballocatable cluster should be named with meaningful names for CLUSTER, DATA and INDEX (KSDS or Alternate Index) to identify the components on a LISTCAT list (see printout examples later in this manual).

The following is an example on how the components could be named for the cluster X.PAYROL:

```
Cluster : X.PAYROL
Data    : X.PAYROL.D
Index   : X.PAYROL.I
```

The NAME parameter is described in detail in section 7.2.5 on page 93.

2.10 VSAM OWNERSHIP CONCEPT

The concept of VSAM volume ownership is illustrated by the following rules:

- An indefinite number of user catalogs can be connected to the master catalog.
- A user catalog can be connected to more than one master catalog (in different systems).
- User catalogs can not be cataloged in other user catalogs (if a master catalog with user catalog connector entries is used as a user catalog, its user catalog connector entries are not used).
- A volume containing a catalog is owned by that catalog only.
- A volume may only belong to one catalog.
- A catalog can own more than one volume.
- A VSAM data set must be cataloged (defined) in the same catalog which owns the data set's volumes. This restriction does not apply to nonVSAM data sets, which can be on any volume, and cataloged in more than one VSAM catalog and/or CVOLs (MVS only).
- MVS CVOLs can only be cataloged in the master catalog.
- VSAM ownership can only be relinquished with DELETE SPACE or DELETE CATALOG (also with ALTER REMOVEVOLUMES and IKQVDU (DOS/VS) (described later)).

2.10.1 THE VSAM 'OWNERSHIP BIT'

VSAM ownership of a volume is controlled by the VSAM ownership bit. If this bit is ON, the volume is owned by a VSAM catalog. There is no indication which catalog owns the volume. When a catalog attempts to allocate space on a volume (by means of DOS/VS system routines or OS-DADSM routines), the following sequence is started.

The VSAM ownership bit is bit 0 in byte 84 (X'54') in the F4-DSCB (Label) (see section A.2.2 on page 246 note 4).

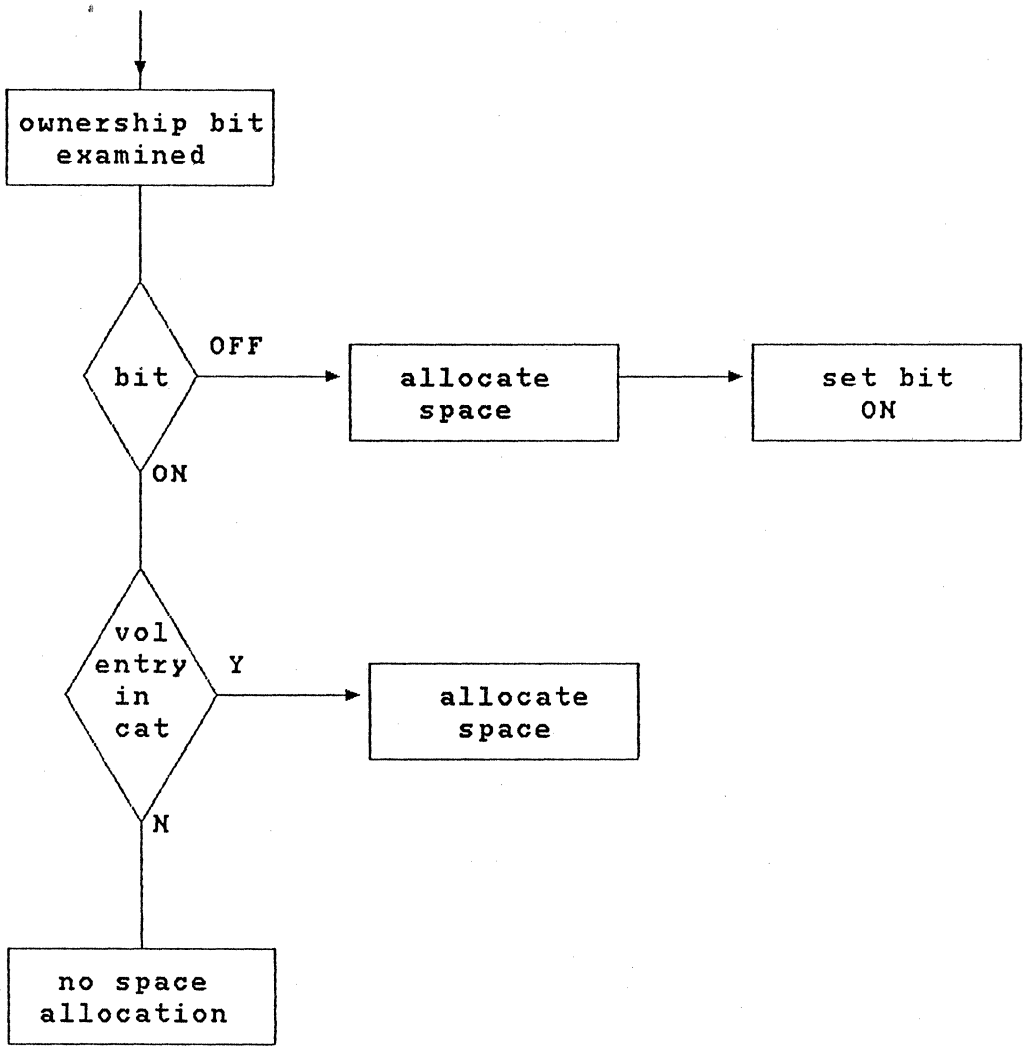


Figure 18. VSAM ownership concept

2.10.2 THE VSAM 'DATA SET SECURITY BIT'

When a VSAM catalog, A VSAM space, or a UNIQUE VSAM data set is allocated on a DASD volume a F1-DSCB (Label) is written into the VTOC.

To protect this VSAM object from being accessed by other access methods the 'data set security bit', bit 3 in byte 93 (X'5D') is set ON (see section A.2.2, page 246 notes 10 and 14). In OS/VS systems the 'data set security bit' is also called and used as 'OS/VS password protection bit' (not to be mixed up with the VSAM password protection).

2.11 VSAM SPACE ALLOCATION

In DOS/VS, VSAM routines are used to allocate space for catalogs and data spaces (JCL EXTENT values necessary), and to suballocate space for VSAM data sets in a suballocatable space (no JCL EXTENT values necessary).

In OS/VS1 and OS/VS2 (SVS and MVS), VSAM catalog/DADSM routines, instead of OS/VS catalog and DADSM routines, are used to process the catalog and to allocate space on VSAM catalog and data set volumes. VSAM routines are used to allocate space in a suballocatable data space.

There are five major Access Method Services commands (see command overview starting in section 4.0 on page 57) which cause an attempt to allocate space. The table below shows these five commands and the space management functions which occur when these 5 commands are executed and an attempt is made to allocate space on a volume (see also description of VTOC on page 245).

POSSIBLE FUNCTIONS	DEFINE CATALOG	DEFINE SPACE	DEFINE SPACE CANDID.	DEFINE CLUSTER UNIQUE	DEFINE CLUSTER SUBALL.
test ownership bit for OFF	*	*1	*	*1	
set ownership bit ON	*	*1	*	*1	
create a volume entry in catalog	*	*1	*	*1	
DOS/VS VSAM routines (EXTENT value req.) or OS/DADSM routines allocate space from disk space and write F1-DSCB (Label) with X'10' in offset X'5D' (VSAM protection bit)	*	*	*2	*3	
DOS/VS VSAM routines or OS/VSAM routines allocate the space from VSAM data space					*

* indicates that the function takes place for the command.

1 this function takes place only when this is the first VSAM allocation on this volume.

2 For a nonrecoverable catalog no space is allocated and no F1 DSCB is written. In OS/VS if the catalog is recoverable, a data space with a primary and secondary allocation of 1 cylinder is allocated by OS/VS DADSM on the same volume as the data space to contain the CRA and a F1-DSCB is written for it (in DOS/VS this command is not allowed for a recoverable catalog).

3 Two F1 DSCBs (Labels) are written one for the data component and the second for the index component of the cluster (in DOS/VS using a recoverable catalog, the volume must already be owned by that catalog, before executing this command).

Figure 19. VSAM space allocation

3.0 CATALOG CONSIDERATIONS

3.1 CATALOG PHILOSOPHY

All VSAM data sets must be cataloged in a VSAM catalog.

Information required to process a VSAM data set, such as its location and physical characteristics, is contained in the VSAM catalog.

There must be one VSAM master catalog for a DOS/VS or OS/VS system which is planned to use VSAM.

In MVS the VSAM master catalog is the standard system catalog.

Note that a VSAM master catalog cannot be stored on a 3850 mass storage volume.

Besides the VSAM master catalog a theoretical unlimited number of VSAM user catalogs can be created (with Access Method Services) and used. VSAM user catalogs are optional and have the same structure as the VSAM master catalog.

VSAM user catalogs should be used to reduce the size of the VSAM master catalog (to reduce catalog processing time), minimize the effect of a damaged master catalog, and make VSAM data sets portable from one system to another (a pack containing a VSAM user catalog and its VSAM data sets can easily be moved from one system to the other without loading or reloading the data sets which is a time consuming operation; only the user catalog must be connected to the other VSAM master catalog with Access Method Services.

Each catalog is an individual or special form of key sequenced data set. Each VSAM user catalog has a connecting pointer entry in the VSAM master catalog. Each VSAM data set is cataloged either in the VSAM master catalog or in one VSAM user catalog, but not both.

All VSAM data sets on the same volume must be cataloged in the same VSAM catalog. Then, all VSAM data spaces (suballocatable or unique) on the same volume belong to the same catalog (because of this, users who share a volume must also share the catalog). Duplicate data set names in the same VSAM catalog are not permitted but a given data set name can appear in more than one VSAM catalog.

DOS/VS does allow nonVSAM entries in a VSAM catalog, but makes no use of these entries (non-VSAM entries cannot be defined under DOS/VS when using a recoverable catalog).

In OS/VS nonVSAM data sets can also be cataloged in a VSAM catalog. In MVS, a generation data group can also be cataloged in a VSAM catalog.

Figure 20 shows a possible structure of VSAM environment, consisting of the VSAM master catalog (VSAM.MCAT) and two VSAM user catalogs.

The master catalog owns volume SYSRES (the catalog volume) and volume VOL004 with the VSAM data set VSAM.STATUS on it.

The master catalog also contains user catalog connector entries pointing to USER.CAT.ONE and USER.CAT.TWO.

VSAM user catalog USER.CAT.ONE 'owns' volume VOL001 and VOL002 containing the VSAM data sets PAYROLL.F, EMPLOYEE.E, and INVOICE.

VSAM user catalog USER.CAT.TWO 'owns' volume VOL003 containing the VSAM data set NEWCUST.

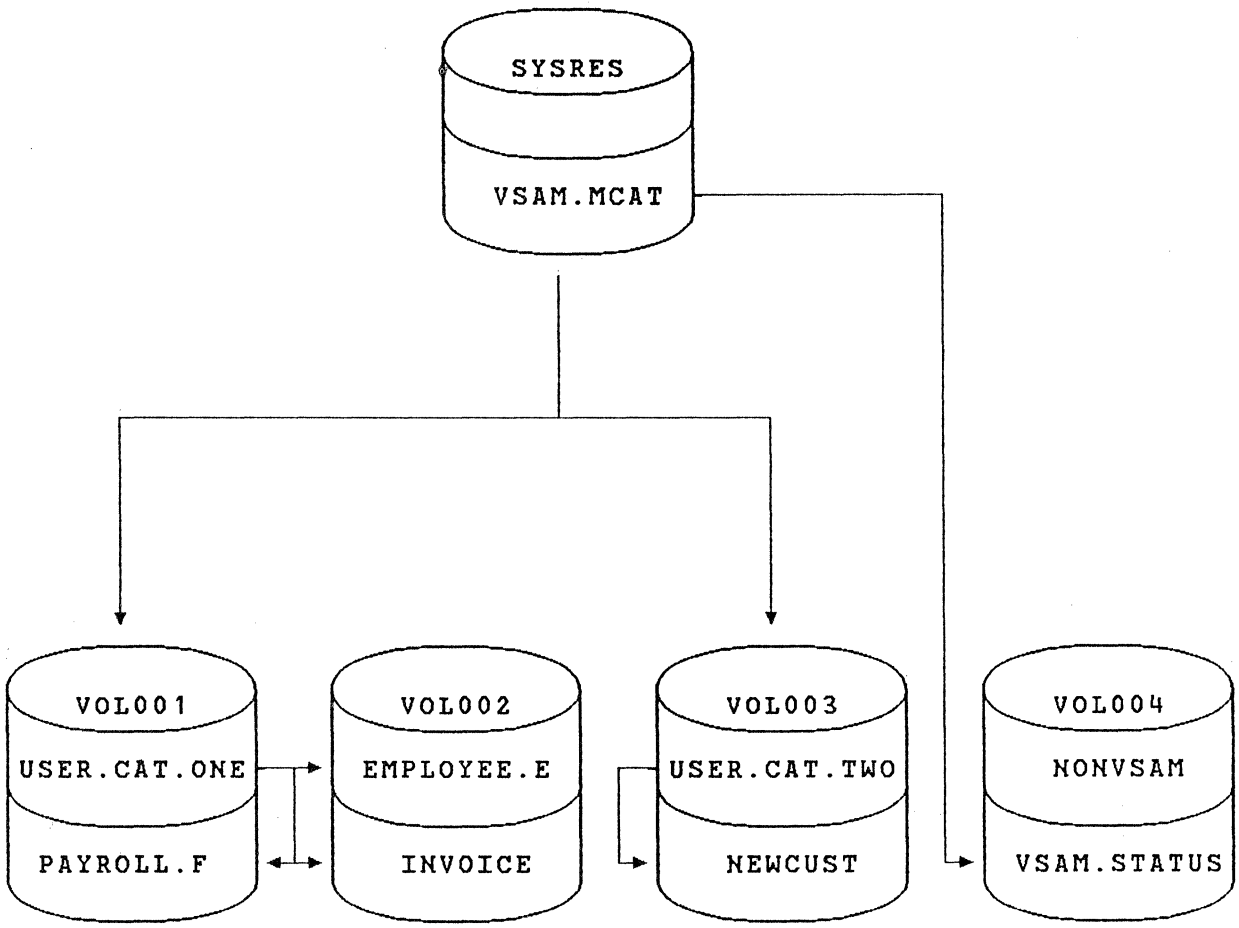
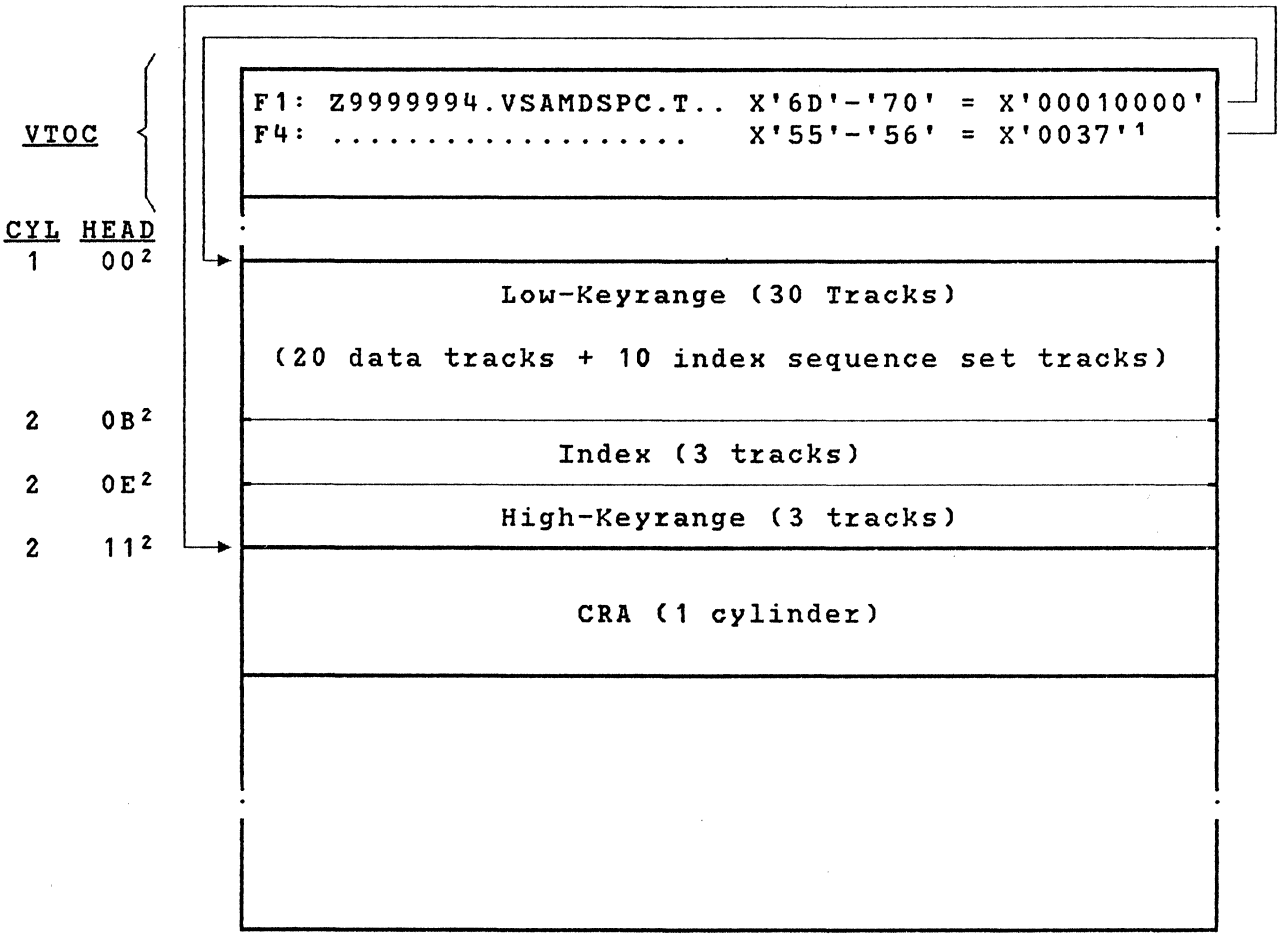


Figure 20. VSAM catalogs

3.2 CATALOG STRUCTURE

The following is the catalog layout of the user catalog, which is defined in section 7.4 on page 102. For further VTOC information see printout and explanation of the VTOC in section A.2.2 on page 245. The CRA (catalog recovery area) is explained on page 250.



Notes:

1 The CRA and its pointer in the F4 DSCB are only included for a recoverable VSAM catalog (a printout of a F4 DSCB is shown on page 245).

X'0037' is the relative track number where the CRA starts. The calculation can be done as follows:

X'0037' = 55 = 38 + 17 = Cyl 2 + 17 (X'11') tracks

2 The starting and ending values for the catalog are extracted from LISTCAT ALL output as explained and shown in section 7.9.2.1 on page 148.

Figure 21. VSAM catalog allocation

A VSAM catalog is a special KSDS, and consists of an index and a data component. The data component, however, consists of two parts: Low-Keyrange and High-Keyrange (see detailed description of normal Keyranges in section A.1.1 on page 241 and section A.1.2 on page 241).

The catalog Keyranges are used in a different way then normal Keyranges (it is actually a design philosophy; see also comments on page 242). The splitting of the data component into High-Keyrange and Low-Keyrange and placing the index between is only used in a catalog KSDS.

When defining a catalog, the size of the entire data component and the size of the index can be specified, but the size of the index component will always be one so-called 'allocation unit' (one control area), which is three or five tracks (allocation unit sizes are described on page 249). One allocation unit is one control area.

The defined data component size is automatically rounded down into an integer number of allocation units. Then 10% (in DOS/VS Rel.34 20%) (minimum 1 allocation unit) is used for the High-Keyrange and the rest for Low-Keyrange.

To improve the performance, the index component is placed between the Low-Keyrange and the High-Keyrange, and the IMBED function (see description on page 89) is always used for a VSAM catalog.

3.2.1 CATALOG CONTENTS

The following information is recorded in catalog records for a VSAM data set:

- Device type and volume serial numbers of volumes containing the data set.
- Location of the extents of the data set and secondary allocations, if any.
- Attributes of the data set, such as control interval size, physical record size, number of control intervals in a control area, location of the primary key field for a KSDS, etc.
- Statistics, such as the number of insertions, updates, retrievals, splits, etc.
- Password protection information.
- An indication of the connection between data sets and their index(es): the index and data components of a KSDS; the index and data components of an alternate index; the alternate index and base cluster of a path; and an alternate index and upgrade set and its base cluster.
- Historic information, such as creation and expiration dates, owner identification, etc.

A VSAM catalog also contains information regarding the location of data spaces and available space on volumes that contain VSAM data sets. Therefore, a volume containing a VSAM data set need not be mounted in order to determine whether it contains available space (but if it is to take any of that space and the catalog is recoverable the owned volume in question must be mounted).

3.2.1.1 CATALOG ENTRY TYPES

Several types of entries are used in a VSAM catalog to describe the various objects the catalog describes (data sets, available space, etc.).

The entry types are as follows (their entry types are included in parenthesis):

- Alias (MVS only)(X)
- Alternate index
  - Alternate index (G)
  - Data component (D)
  - Index component (I)
  - Upgrade set (Y)
- Catalog control record (L)
- Cluster
  - Data set cluster (C)
  - Data component (D)
  - Index component (KSDS only) (I)
- Extension record (for all records,except the volume record)(E)
- Free record (F)
- Generation data group (MVS only) (B)
- NonVSAM (OS/VS only),(can be defined in DOS/VS (not recoverable catalog)) (A)
- Path (R)
- Self-describing entries (see page 242) (no entry type)
- User catalog (used in master catalog only) (U)
- Volume (contains volume and space information) (V)
- Volume extension record (W)

A given data set may require more than one entry type for its description, plus extension entries when the data set is extended and/or various volumes are used. An ESDS, for example, requires a cluster and data component entry.

3.2.1.2 CATALOG SHARING

A master or user catalog in one VS1 or VS2 operating system can be shared with another VS1 or VS2 operating system as a master or user catalog with one exception: a master catalog in one VS2 operating system can be shared with another VS2 system only as a user catalog (not as a master catalog). Catalog management routines control this sharing.

No sharing of master or user catalogs is supported under two or more DOS/VS systems simultaneously. Nevertheless, read-only sharing of the catalogs is possible. Therefore no changes of the data sets is allowed, which may change the content of the catalog.

3.3 VSAM PASSWORDS

An expanded password protection facility is supported for VSAM. Optionally, passwords can be defined for clusters, cluster components (data component and index component), alternate indexes and components, paths, and VSAM catalogs. VSAM passwords are kept in VSAM catalog entries only. The password can be supplied by the programmer via the ACB (a VSAM control block similar to the DOS/VS DTF or the OS/VS DCB used by other access methods; see description on page 217).

If password protection is indicated for a VSAM data set and the ACB does not specify a password (see page 55) or specifies it incorrectly, the operator may supply the correct password for the data set to be opened. A MVS TSO user can also supply VSAM passwords. The number of retries allowed (from 0 to 7) can be specified with the ATTEMPTS parameter (default is 2) in the DEFINE command.

Four levels of password protection are provided:

- LEVEL 1: MASTER PASSWORD

  Full access, which allows access to a data set, its index(es), and its catalog entry. Any operation (read, add, update, delete) can be performed on the data set and its catalog entry. The master password of the base KSDS must be specified, for example, when an alternate index is to be created for the base, and for other cases.
  The master password is required for altering (ALTER) and deleting (DELETE) VSAM data sets.

- LEVEL 2: CONTROL INTERVAL PASSWORD (for special usage)

  Control interval access, which allows the user to read and write entire control intervals using the control interval interface. All read, write, and update operations can be performed at the logical record level as well. This facility is not provided for general use and should be reserved for system programmer use only.

- LEVEL 3: UPDATE PASSWORD

  Update access, which allows logical records to be retrieved, updated, deleted, or added.

For the catalog the following applies:

- DOS/VS, VS1, SVS: Update password is required for defining VSAM data sets and nonVSAM data sets in VS1 and SVS). No passwords are required in VS1 and SVS for altering (ALTER) and deleting (DELETE) a nonVSAM data set.

- MVS: Update password is required for defining (DEFINE) VSAM and nonVSAM data sets and for altering (ALTER) and deleting (DELETE) nonVSAM data sets (including GDGs and ALIASes).

- LEVEL 4: READ PASSWORD

  Read access, which allows access to a data set for read operations only. Read access to the catalog entries of the data set (except password information) is permitted also. No writing is allowed.

  A password can be defined for a given VSAM data set for each level protection: master password, control interval access password, read- write-add-delete password, and read-only password. When multiple passwords are defined for a data set, the password given when the data set is opened establishes the level of protection to be in effect for this OPEN.

Authorization to process a VSAM data set can be supplemented by a user-written security authorization routine.

If supplied, such a routine must reside in the following library:

- DOS/VS : System CIL or Private CIL (Core Image Library).

- OS/VS : SYS1.LINKLIB or in a library to be located by the system (e.g. in a JOBLIB).

It is entered during OPEN processing after password verification has been performed by VSAM, unless the master access password was specified. A user security authorization record of up to 255 bytes maximum can also be added to the catalog entry for the data set. This record can supply data to the user-written security authorization routine during its processing. For further information see the appropriate Access Method Services manuals as listed on page 2.

The VSAM catalog must be password protected to support password protection for cluster passwords.

If passwords are not specified on all levels, the highest level password is propagated to the higher levels (see figure 22).

The following chart shows some combinations of specified passwords and the passwords assigned by VSAM:

Passwords specified by the user				Passwords assigned by VSAM			
Read	Update	CI	Master	Read	Update	CI	Master
R	-	-	-	R	R	R	R
-	U	-	-	-	U	U	U
-	-	C	-	-	-	C	C
-	-	-	M	-	-	-	M
R	U	-	-	R	U	U	U
R	U	-	M	R	U	-	M
R	-	-	M	R	-	-	M
-	U	-	M	-	U	-	M

Figure 22. VSAM passwords

Passwords for catalogs are strongly recommended to prevent unauthorized deletion of VSAM objects (e.g. clusters, data spaces and catalogs).

In OS/VS systems the VSAM master catalog master password is required for volume cleanup (see page 181) and other severe commands. Therefore it is very important to have at least the VSAM master catalog password protected.

Passwords can be defined using Access Method Services commands (see definition in sections 7.4 on page 102 and 7.6.4.11 on page 121).

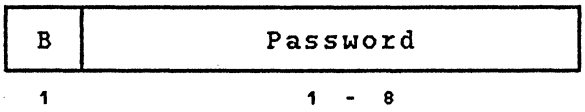
Section 3.3.1 on page 55 contains information about how to specify a password in a user-written program.

3.3.1 HOW TO SPECIFY PASSWORDS IN A USER-WRITTEN PROGRAM

As shown in figure 47 and 48 on pages 212 and 215 the ACB macro (see also description section 8.11.1.1 on page 217) contains a parameter:

PASSWD=address

'address' can be substituted by a symbolic field name pointing to a field (maximum length 1+8 bytes) with the following format:



B = length of the password (binary value)

If a password is not supplied when a password protected data set is to be opened, the system operator is prompted to enter the password.

This prompting of the operator is controlled by the number of attempts to enter the password defined for the data set with the ATTEMPTS parameter (if this parameter was not specified, the default value ATTEMPTS=2 is assumed).

4.0 ACCESS METHOD SERVICES (OVERVIEW OF FUNCTIONS)

Access Method Services consists of the following functions (commands):

- ALTER changes attributes in the VSAM catalog for previously defined VSAM objects.
- BLDINDEX sorts and loads pointer information into a newly defined alternate index.
- CHKLIST lists data sets, opened when a checkpoint was taken (OS/VS only).
- CNVTCAT converts OS/VS catalogs into VSAM catalogs (MVS only) (this is a very special function and is not described in this manual).
- DEFINE defines VSAM objects, such as catalogs, clusters, etc.
- DELETE deletes VSAM objects.
- EXPORT copies/exports a VSAM data set with related catalog information to a sequential data set (using the catalog to access the data sets).
- EXPORTRA exports data sets with related catalog information, and nonVSAM, GDG, and Alias catalog entries (using the catalog recovery area to access the data sets).
- IMPORT imports and reorganizes a VSAM data set, which was previously exported by EXPORT.
- IMPORTRA imports and reorganizes data sets and imports VSAM catalog entries, which were previously exported by EXPORTRA.
- LISTCAT lists VSAM catalog information.
- LISTCRA lists and optionally compares the catalog recovery area information with the the catalog records.
- PRINT prints contents of ISAM, SAM, or VSAM data sets.
- REPRO loads, copies, converts, merges, reorganizes data sets and catalogs.
- RESETCAT synchronizes and resets a catalog (using catalog recovery areas)
- VERIFY resets catalog information for a VSAM data set, which was not closed in a normal way (e.g., hardware malfunction).

The following commands are used to control the execution of Access Method Services commands (modal commands):

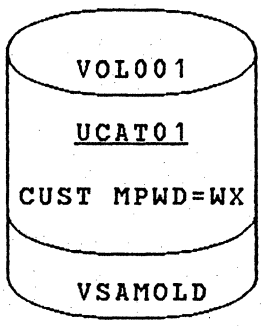
- IF and DO control sequence of execution based on values of condition codes issued by VSAM or changed by the user through the SET command.
- PARM specifies processing options to be used during execution.
- SET changes or resets condition codes.

4.1 ALTER

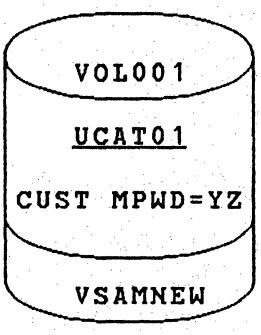
This command is used to change attributes in previously defined catalog entries, for example to change data set passwords, free space specifications, etc.

See the appropriate Access Method Services manual (listed on page 2) for correct coding; an example is also shown on page 121.

In figure 23 the master password of entry CUST is altered from WX to YZ. UCAT01 is the user catalog name. The VSAM data set VSAMOLD is then renamed to VSAMNEW.



OLD



NEW

Command coding:

```
ALTER CUST                -
      MRPW(YZ)            -
      .....
ALTER VSAMOLD             -
      NEWNAME(VSAMNEW)
```

Figure 23. Logical function of ALTER

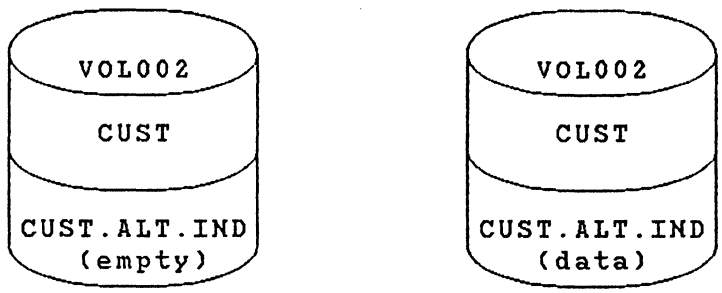
4.2 BLDINDEX

This command is used to create (load) an alternate index for a base cluster. The alternate index must already be defined, and the base cluster must have at least one record.

BLDINDEX reads all base cluster records and extracts the primary keys (for KSDS base cluster) or the RBAs (for ESDS base cluster), and the related alternate keys. This information is then sorted and loaded into the alternate index.

See the appropriate Access Method Services manual (listed on page 2) for correct coding; an example is also shown on page 113.

In figure 24 CUST.ALT.IND is an alternate index for base cluster CUST.



Command coding:

```
BLDINDEX INFILE (...) -
         OUTFILE(...) -
         .....
```

Figure 24. Logical function of BLDINDEX

4.3 CHKLIST (OS/VS ONLY)

This is not a VSAM command and is not used for normal VSAM processing.

During processing, a program can issue the CHKPT macro to record various information for use in restarting the program in the event of an error. This is called taking a checkpoint.

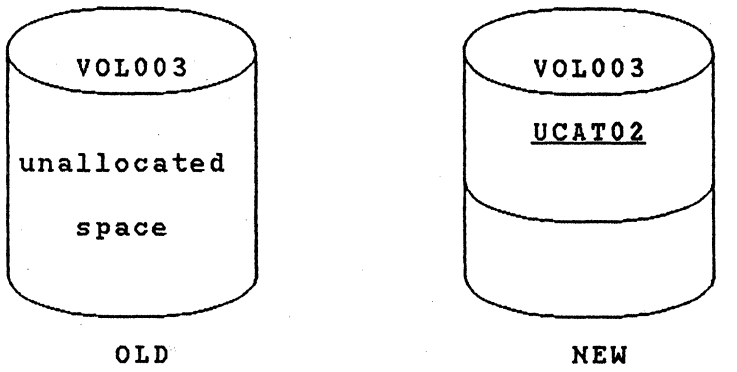
The CHKLIST command lists the tape data sets which were open at the time the checkpoint was taken, thus identifying the tape data sets which need to be mounted for restart.

4.4 DEFINE

This command is used to create catalogs, and catalog entries for alternate indexes, clusters, data spaces, paths, and nonVSAM data sets. In MVS catalog entries for ALIASes, GDGs, and PAGESPACEs can also be created. DEFINE creates the catalog entry for a VSAM object and allocates space for this object.

See the appropriate Access Method Services manual (listed on page 2) for correct coding; examples are also shown starting on page 100.

Figure 25 shows the definition of a user catalog.
UCAT02 is the new user catalog.

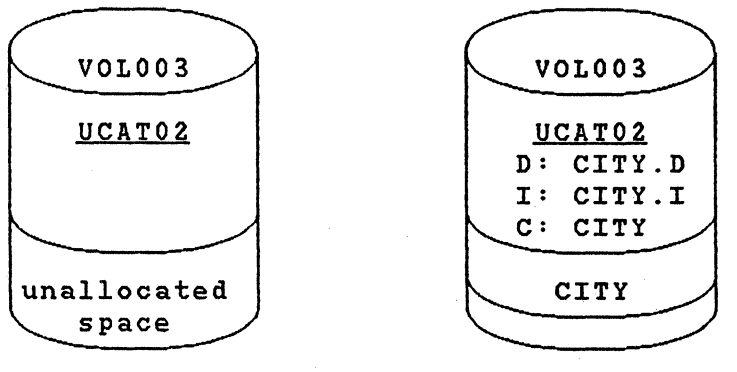


Command coding:

```
DEFINE UCAT                -
       (NAME(UCAT02)       -
       VOL(VOL003)         -
       SPACE (CYL(20 1))-
       .....
```

Figure 25. Logical function of DEFINE USERCATALOG

In figure 26 the KSDS cluster (C), data entry (D), and index entry (I) is added to UCAT02 (UCAT02 is the user catalog name) and a UNIQUE data space is built.
It is assumed the user specified the following names: CITY (cluster), CITY.D (data component), and CITY.I (index component) (see naming restrictions for MVS in section 7.2.5 on page 93). The unique space for the cluster CITY is to be allocated from the same volume.



Command coding:

```
DEFINE CLUSTER             -
        (NAME(CITY)        -
        VOL(VOL003)        -
        UNIQUE             -
        SPACE(CYL(5 1)))-
       DATA                -
        (NAME(CITY.D))     -
       INDEX               -
        (NAME(CITY.I))     -
       .....
```

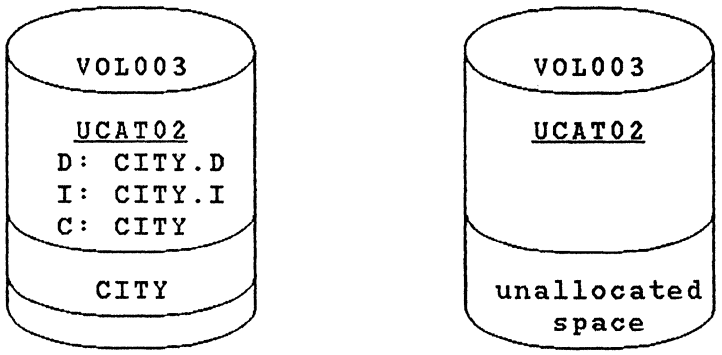
Figure 26. Logical function of DEFINE CLUSTER

4.5 DELETE

This command is used to delete catalogs and catalog entries.

See the appropriate Access Method Services manual (listed on page 2) for correct coding; examples are also shown starting on page 174.

In figure 27 the cluster entry with its component entries is deleted from UCAT02. The allocated UNIQUE space is freed to be used by the system.



Command coding:

```
DELETE CLUSTER      -
       (NAME(CITY)  -
       VOL(VOL003)  -
       UNIQUE ... )
```

Figure 27. Logical function of DELETE

4.6 EXPORT/IMPORT

These commands are used to unload/reload VSAM data sets for backup and/or transportation.

The EXPORT command has, among others, two important options: TEMPORARY|PERMANENT. EXPORT PERMANENT unloads the data set and deletes the entries from the catalog. EXPORT TEMPORARY produces an unloaded copy of the data set, and marks the catalog entry, to show that a temporary copy exists. In the unloaded format, the data set is not accessible.

EXPORT uses the VSAM catalog to copy all related information such as cluster entry, data entry, and index entry (for KSDS only) from the catalog to the target data set, a SAM VBS data set (the user allocated via JCL).

IMPORT redefines and reloads the data set.

These commands can also be used to disconnect/reconnect a user catalog from/to a VSAM master catalog (if a user catalog is disconnected from the master catalog by deleting its connector entry, it cannot be accessed from that system). No unload/reload is necessary.

See the appropriate Access Method Services manual (listed on page 2) for correct coding; examples are also shown starting on page 122.

In figure 28 the unloaded copy of CITY contains data and catalog information for the data set CITY. The unloaded copy is to be stored on disk or tape, JCL must be specified as for any other nonVSAM data set (this is a SAM VBS data set).

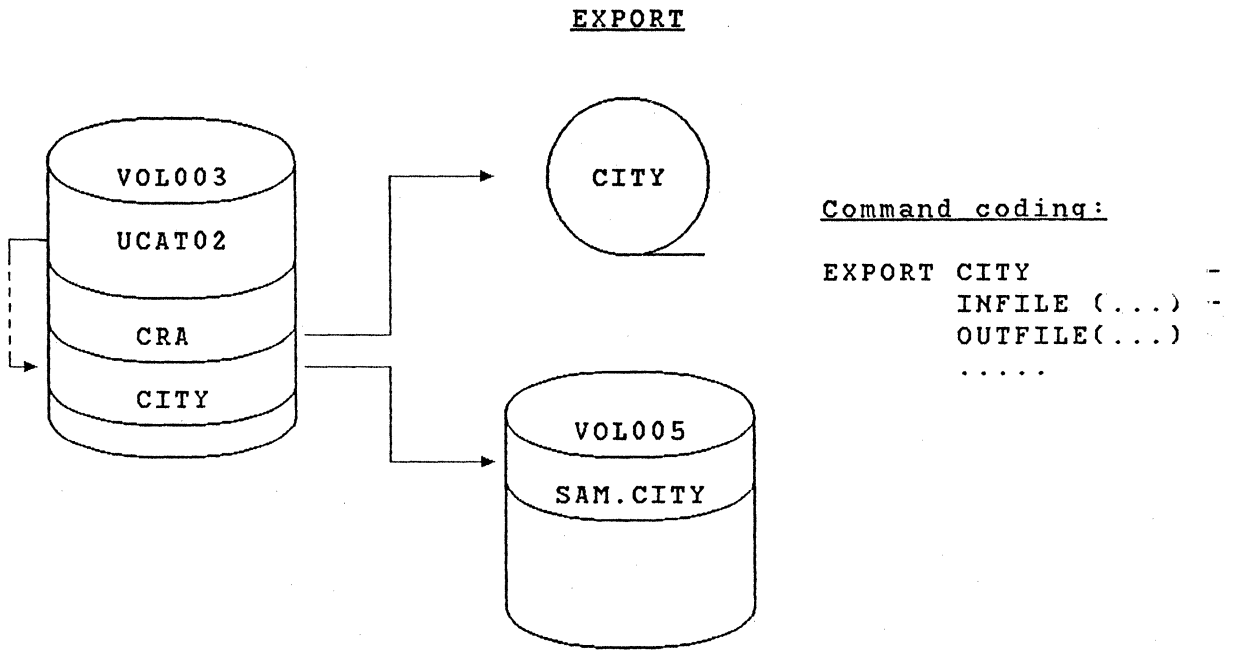


Figure 28. Logical function of EXPORT to disk or tape

In figure 29 cluster CITY is restored by using the sequential VSAM backup created by a previous EXPORT.

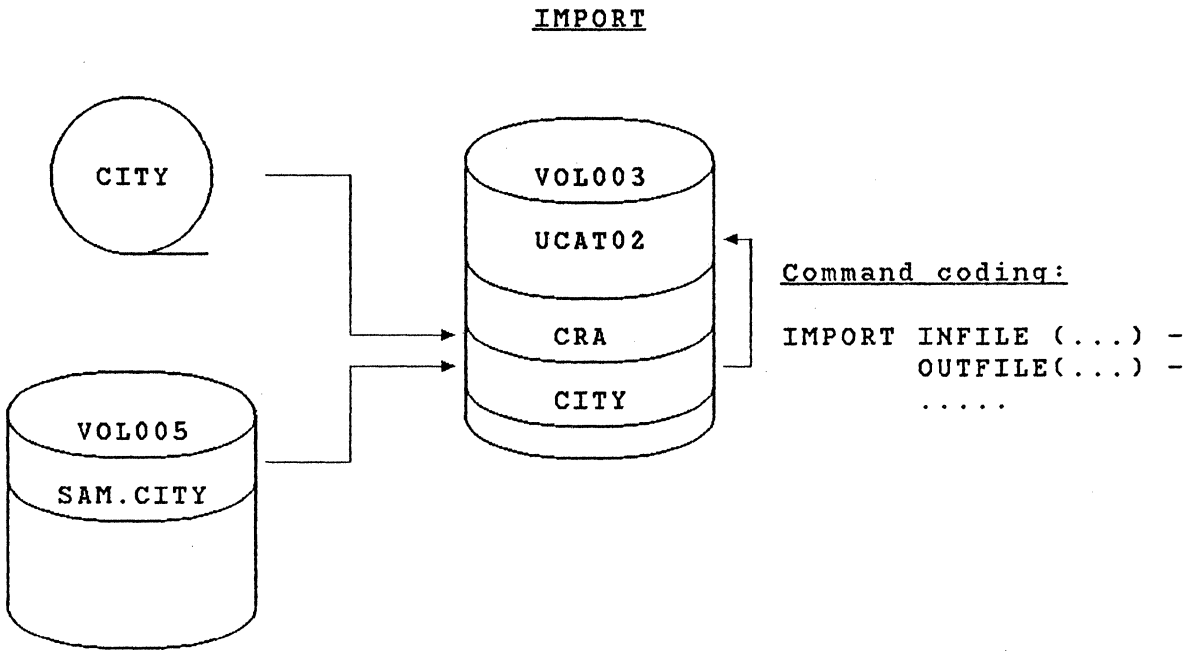


Figure 29. Logical function of IMPORT from disk or tape

4.7 EXPORTRA/IMPORTRA

If certain data sets are no longer accessible because of damage to their catalog entries, and that catalog is recoverable, these data sets can be unloaded using the EXPORTRA command.

In figure 30 EXPORTRA uses the CRA (catalog recovery area) and not the VSAM catalog to access the data set CITY to be recovered. It copies all related information such as cluster entry, data entry, and index entry from the catalog recovery area to the target data set (the target SAM data set is allocated by the user JCL, see EXPORT).

In figure 31 IMPORTRA is used to reload the data set CITY and rebuild the corresponding catalog entries. The logical function is the same as for EXPORT/IMPORT without using the catalog to find the data set.

The EXPORTRA/IMPORTRA commands are also described in section A.6.3 on page 259.

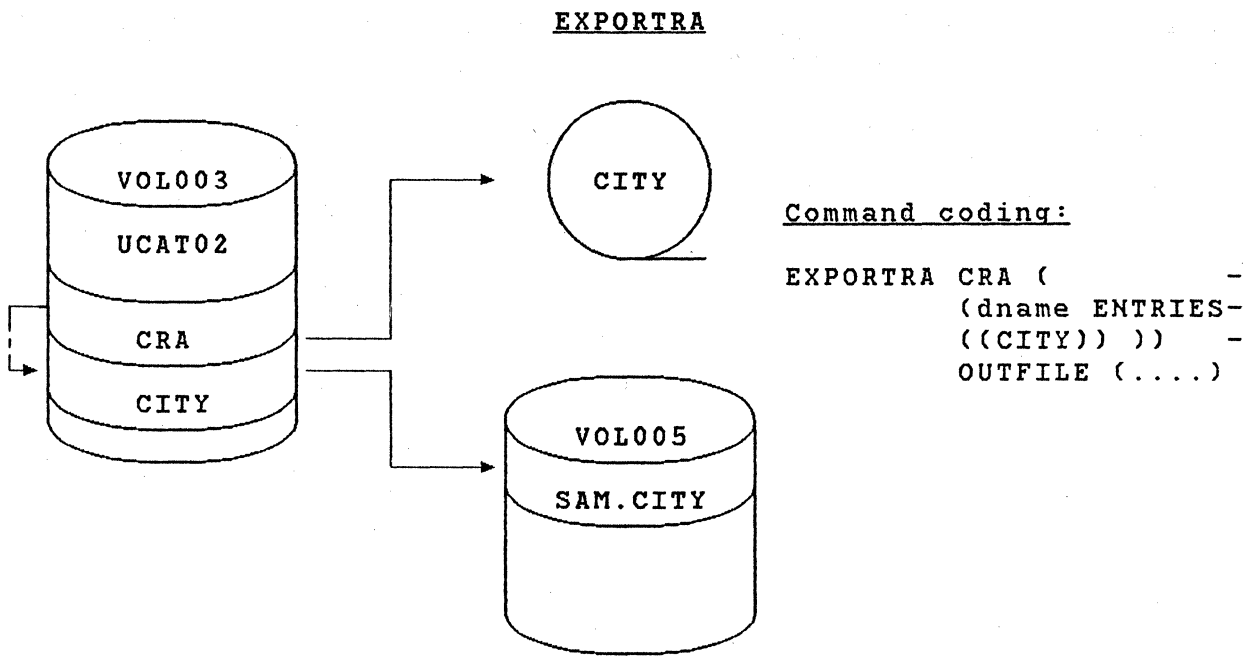


Figure 30. Logical function of EXPORTRA to disk or tape

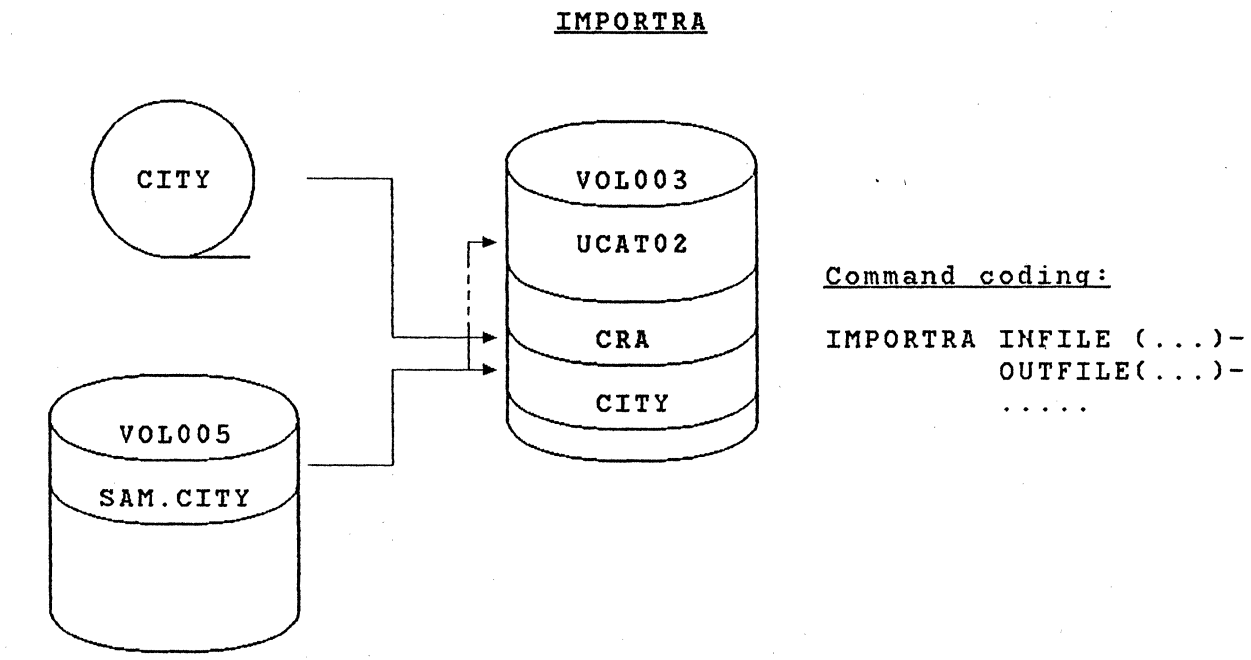


Figure 31. Logical function of IMPORTRA from disk or tape

4.8 LISTCAT

This command is used to list the VSAM catalog entries or parts of them. It is recommended to execute a LISTCAT command after any DEFINE command. Examples are shown and explained starting on page 140.

4.9 LISTCRA

This command is used to:

- List entries in the catalog recovery area (CRA)
- Indicate mismatches (if any) between the CRA and the actual catalog
- Dump the contents of the CRA(s)

Examples are shown starting on page 160.

4.10 PRINT

This command is used to print ISAM, SAM or VSAM data sets, or parts of these data sets. The printout can be obtained in hexadecimal, character or dump format.

Examples are shown starting on page 109.

4.11 REPRO

This command transfers data between 2 data sets. If data are to be transferred into an empty data set, it always reorganizes the target data set.

When copying an RRDS to a nonRRDS, empty slots are not copied. The slot numbers of the non-empty slots are lost.

When copying an RRDS to an RRDS, each non-empty slot in the source is copied to the same slot in the target. Empty slots are not copied.

The different functions are:

- Add records to the end of an ESDS
- Copy a VSAM catalog (MVS only) (move a catalog to another disk)
- Load/copy a VSAM data set

ISAM ⟶	SAM	(backup/unload an ISAM data set)
ISAM ⟶	VSAM	(convert an ISAM data set to VSAM format)
SAM ⟶	SAM	(copy, e.g. tapes)
SAM ⟶	VSAM	(load a VSAM data set)
VSAM ⟶	SAM	(backup/unload a VSAM data set
VSAM ⟶	VSAM	(copy/merge data)

- Merge records into KSDS or RRDS
- Punch or print

  ISAM data sets
  SAM data sets
  VSAM data sets

- Unload/reload a VSAM catalog (backup)

  See the appropriate Access Method Services manual (listed on page 2) for correct coding; examples are also shown starting on page 108.

Figures 32 - 34 show a few functions of REPRO:

Load a VSAM data set from disk, cards, or tape
For the following functions, different REPRO commands have to be used.

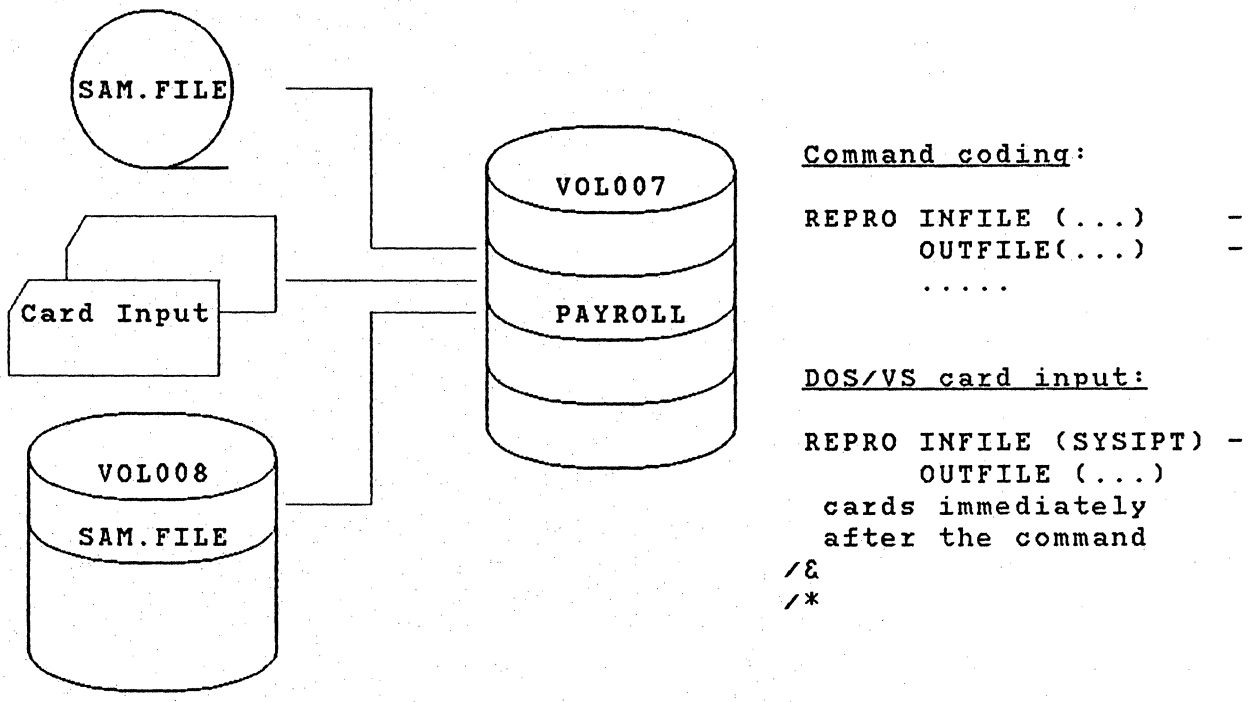


Figure 32. Logical function of REPRO (load) from disk or tape

Copy/convert data set to data set

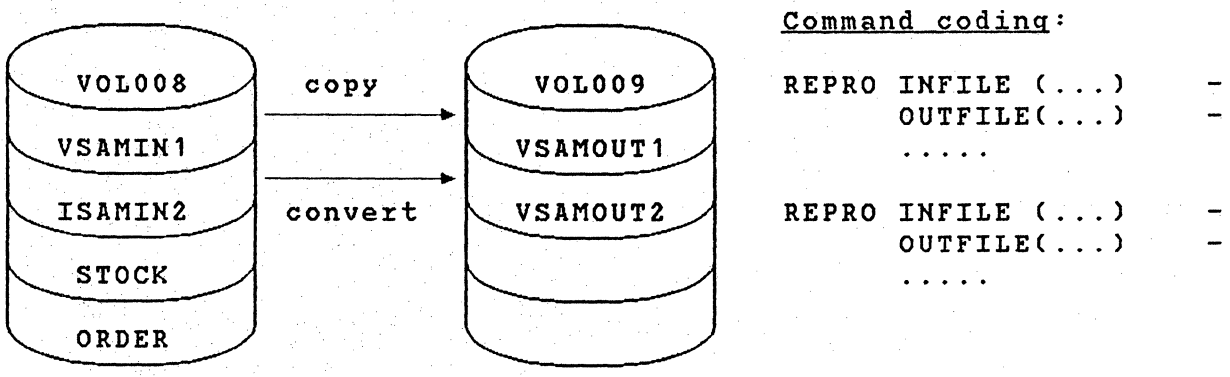
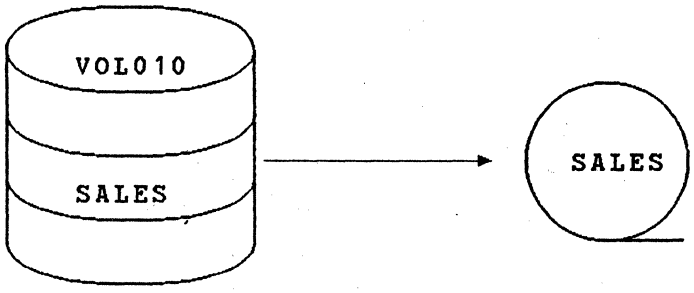


Figure 33. Logical function of REPRO (copy data)

Backup data to tape

The output tape data set contains no catalog information.



Command coding:

```
REPRO INFILE (...)  -
      OUTFILE(...)  -
       .....
```

Figure 34. Logical function of REPRO (backup data sets/catalogs)

4.12 VERIFY (ACCESS METHOD SERVICES)

This command is used to ensure a catalog reflects the correct 'high used RBA' of a data set (the 'high used RBA' points to the last byte used in the VSAM data set; see LISTCAT explanation on page 156). It should be used after an OPEN-error caused by a previous system failure or an ABEND condition while updating the data set.

VERIFY cannot be used for empty data sets (when the high-used RBA = 0 in the catalog cluster entry). This condition will also occur when an ABEND or system failure occurs during the load of the data set with the SPEED or RECOVERY option specified (see section 7.2.8 on page 97), or while reloading a reusable data set specifying the REUSE option (see page 16).

An example is shown on page 126.

For further explanation see also section 4.13, which describes the VERIFY macro.

4.13 VERIFY (MACRO)

There is the VERIFY macro itself, which only update the 'high-used RBA' in the control block structure and not in the VSAM catalog.

The Access Method Services command VERIFY opens the data set, issues the VERIFY macro to update the control blocks. Access Method Services then issues a CLOSE macro and it is this VSAM CLOSE that updates the VSAM catalog.

Section 4.12 describes the Access Method Services VERIFY command and not the VERIFY macro itself.

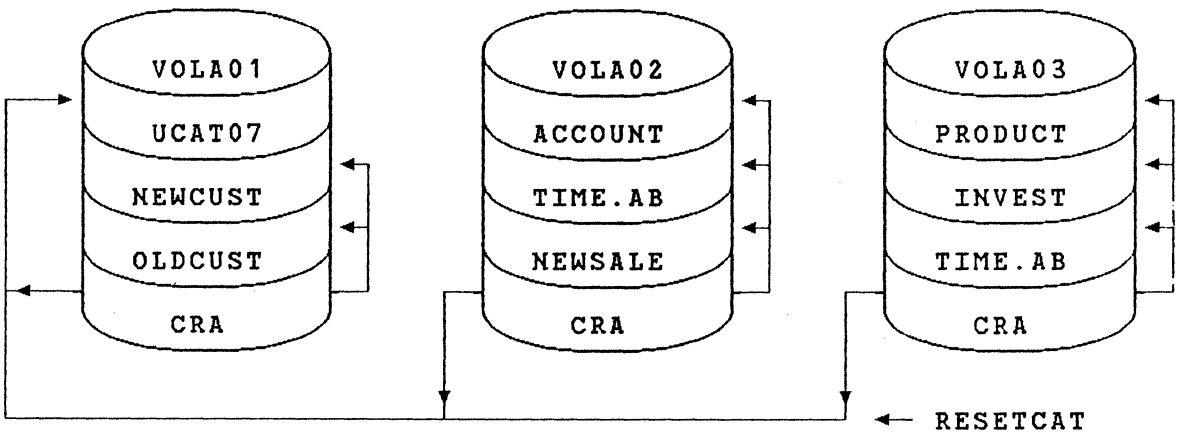
4.14 RESETCAT

This command synchronizes a recoverable VSAM catalog with its volumes. For example if a VSAM volume becomes inaccessible and a backup copy of the catalog volume is used to restore the volume to a previous level, the volume may be out of synchronization with its catalog.

RESETCAT compares the relevant catalog entries with the entries in the CRA on that volume, and resets the catalog entries where necessary, so that the catalog is synchronized with the volumes.

RESETCAT usage for unload/reload and device conversion is described on pages 254 and 259.

See the appropriate Access Method Services manual (listed on page 2) for correct coding; an example is also shown on page 164.



Command coding:

```
RESETCAT  CAT UCAT07                                          -
          CRAFILES ((VOLA01 ALL)                              -
                    (VOLA02 ALL)                              -
                    (VOLA03 ALL) )                            -
          WORKCAT   (name of a second or master catalog)      -
          WORKFILE  (workfile d-name)
```

Figure 35. Logical function of RESETCAT

5.0 RELATIONSHIP OF DATA ACCESS TO DATA ORGANIZATION

5.1 OVERVIEW OF ACCESS METHODS OTHER THAN VSAM

- BASIC DIRECT ACCESS METHOD (BDAM)

  In the Basic Direct Access Method (BDAM), records within a data set are organized on direct access volumes in any manner chosen by the programmer. Storage and retrieval of a record is by relative address within the data set or a actual disk address. This address can be that of the desired record or a starting point within the data set where a search for a record, based on a key supplied by the programmer, begins. Addresses are also used by BDAM as a starting point when seeking available space for new records.

  The READ/WRITE macro instructions cause the initiation of an input/output operation. The completion of these operations is tested by using synchronization macro instructions.

- BASIC INDEXED SEQUENTIAL ACCESS METHOD (BISAM)

  Sequential and direct processing are provided by the Indexed Sequential Access Method (ISAM). Records are maintained on DASD devices in control field sequence by key. The system maintains a multilevel index structure that allows retrieval of any record by its key. Records can be added to an existing ISAM data set without rewriting the data set.

  The Basic Indexed Sequential Access Method (BISAM) stores and retrieves records randomly into or from an indexed sequential data set. Selective reading is performed using the READ macro instruction, specifying the key of the logical record to be retrieved. Individual records can be replaced or new records can be added randomly. As new records are inserted, ISAM performance decreases, until it becomes necessary to reorganize the data set. When records are deleted, the space is not reuseable until after the next data set reorganization.

  The READ/WRITE macro instructions cause the initiation of an input/output operation. The completion of these operations is tested by using synchronization macro instructions.

- BASIC PARTITIONED ACCESS METHOD (BPAM) (OS/VS only)

  The Basic Partitioned Access Method (BPAM) is designed for efficient storage and retrieval of discrete sequences of data (members) belonging to the same data set on a direct access device. Each member of the data set has a simple name. The data set includes a directory that relates the member name with the address where the sequence begins. Members can be added to a partitioned data set as long as space is available in the directory and in the data set.

The READ/WRITE macro instructions cause the initiation of an input/output operation. The completion of these operations is tested by using synchronization macro instructions.

- BASIC SEQUENTIAL ACCESS METHOD (BSAM)

  The Basic Sequential Access Method (BSAM), sequentially organizes data and stores or retrieves physical blocks of data from tape, disk or unit record devices. The READ/WRITE macro instructions cause the initiation of an input/output operation. The completion of these operations is tested by using synchronization macro instructions. Automatic translation between EBCDIC and ASCII codes is provided for magnetic tape labels and record formats.

- DIRECT ACCESS METHOD (DAM)

  This is another name for BDAM.

- INDEXED SEQUENTIAL ACCESS METHOD (ISAM)

  The Indexed Sequential Access Method (ISAM) is comprised of the Basic Indexed Sequential Access Method (BISAM) and the Queued Indexed Sequential Access Method (QISAM). See the description of these two access methods in this section.

- QUEUED INDEXED SEQUENTIAL ACCESS METHOD (QISAM)

  The Queued Indexed Sequential Access Method (QISAM) is used to create an indexed sequential data set or to retrieve and update records sequentially from such a data set. Synchronization of the program with the completion of input/output transfer, and record blocking/deblocking are automatic. QISAM is also used to reorganize an existing data set.

  The GET/PUT macro instructions cause the initiation of an input/output operation. These operations are synchronous.

- QUEUED SEQUENTIAL ACCESS METHOD (QSAM)

  In the Queued Sequential Access Method (QSAM), logical records are retrieved or stored as requested. The access method anticipates the need for records based on their sequential order, and normally has the desired record in storage, ready for use, before the request for retrieval is issued. When writing data, the program normally continues as if the record had been written immediately, although the access method routines may block it with other logical records and defer the actual writing until the output buffer has been filled. As with BSAM, automatic translation between EBCDIC and ASCII codes is provided for magnetic tape labels and record formats.

  The GET/PUT macro instructions cause the initiation of an input/output operation. These operations are synchronous.

- SEQUENTIAL ACCESS METHOD (SAM)

  The Sequential Access Method (SAM) is comprised of the Basic Sequential Access Method (BSAM) and the Queued Sequential Access Method (QSAM). See the description of these two access methods in this section.

The following are Telecommunications Access Methods and are not discussed further in this manual.

- BASIC TELECOMMUNICATIONS ACCESS METHOD (BTAM)
- REMOTE TERMINAL ACCESS METHOD (RTAM)
- TELECOMMUNICATIONS ACCESS METHOD (TCAM)
- VIRTUAL TELECOMMUNICATIONS ACCESS METHOD (VTAM)

5.2 SAM VERSUS SEQUENTIAL VSAM

Most direct access SAM applications can be mapped into VSAM sequential applications with reasonably good results. There are a number of parameters that might affect performance significantly.

5.2.1 LOGICAL RECORD PROCESSING (ADR-ESDS)

This method is usually a good alternative to SAM if the number of records per CI is small and the number of data buffers is greater than the minimum (for more information about buffer, see section 8.1.5, starting on page 190). If the number of records per CI is large, the overhead per logical record may require more CPU time then SAM.

5.2.2 LOGICAL RECORD PROCESSING (KEY-RRDS)

One of the efficient means of doing sequential accessing is by using an RRDS. If the records are fixed length then this is a good choice. It is not possible to define an alternate index over an RRDS.

5.2.3 KEYED SEQUENTIAL RECORD PROCESSING (KEY-KSDS)

This is probably the least efficient mapping of SAM into VSAM since at each change of CI there must be a reference to the sequence set buffer to locate the next CI. If a logical consolidation of programs can occur which might make use of the insertion capabilities of VSAM then the extra overhead might be worth it. Normally in a SAM application a file update involves both an input and an updated output version of the same file containing inserted records or updated records having length changes. In this case a single KSDS can take the place of both the input and output SAM files.

5.2.4 CONTROL INTERVAL PROCESSING

The most efficient method of processing is control interval processing, but it is considered only for very special applications. The problem program must, however, do all of the logical record blocking and deblocking itself.

For applications that can deblock a control interval easily or where there is only one record per CI this is an excellent choice. In programs using COBOL and PL/I versions where CNV processing is not supported, the user can define a user block at least 10 bytes less than the CI that will be defined for the file and process his own user records. This will reduce the number of VSAM logical records per CI to as few as 1 depending upon the user blocksize and CI size chosen.

Caution should be used processing a KSDS or RRDS with CNV processing and generally should be avoided by the application programmer.

In VS1 and VS2 the ICI option (Improved CI Processing) (a subparameter of the MACRF parameter in the ACB macro instruction) may be used to further reduce CPU time for data sets meeting the requirement of ICI Processing. The data set, however, may not be extended or created using ICI. If ICI is used and the program is authorized it may also use the CFX option (a subparameter of the MACRF parameter in the ACB macro instruction) to page fix the control blocks. The user must specify UBF (a subparameter of the MACRF parameter in the ACB macro instruction) and page fix his user buffer if CFX is used.

5.3 BDAM VERSUS DIRECT ESDS AND RRDS

BDAM data sets with fixed length blocks can be mapped into VSAM with fixed length records giving excellent results.

Each fixed length block in BDAM becomes a logical record in VSAM. The dummy BDAM blocks become empty record slots, (if a VSAM RRDS is used) and the KEY and DATA portions of the BDAM block can be combined into a single record entry in an ESDS or RRDS.

Because keyed BDAM has an unblocked format and VSAM has a blocked format there is potentially much better space utilization in VSAM. As a result the number of cylinders and consequently the average seek time is generally less for an ESDS or RRDS than for BDAM. A read by key in BDAM causes the channel and the device to be busy until the block is found, resulting in high channel and device times when searching by key. In VSAM RRDS retrieval the channel and device are busy only for the duration of the data transfer. During an extended search for a record in VSAM, sequential processing can be used to read several CI's worth of records with a single EXCP.

The formatting of an RRDS is a small fraction of the time required to format a BDAM data set with dummy records. The initial random loading of a BDAM, ESDS or RRDS data set are nearly equivalent. VSAM uses more CPU time than BDAM, but is better on channel time. It is assumed that random loading will not occur frequently. Backup and recovery are done sequentially which should be much faster with VSAM.

Direct retrieval of a VSAM record is significantly better than BDAM retrieval in total elapsed time per record but uses slightly more CPU cycles.

In general all operations using VSAM are better than BDAM in elapsed time. VSAM operations tend to use more CPU cycles than BDAM whereas BDAM uses more channel time than VSAM processing.

5.4 ISAM VERSUS VSAM

All ISAM applications can be mapped into VSAM KSDS applications. Either native VSAM support or VSAM through the ISAM Interface Program (IIP) may be used. Use of the IIP avoids program conversion (see also next section and section 5.4.3).

Some general comments about the relative performance of VSAM and ISAM should be made. They are:

- OPEN and CLOSE in VSAM are significantly longer than in ISAM.
- Data set creation in VSAM takes longer than ISAM depending upon the amount of free space required, the number of buffers, and the blocking factor.
- Direct VSAM operations are usually faster than the equivalent ISAM operation.
- VSAM requires more virtual storage than ISAM.
- In sequential access, VSAM generally uses more CPU time than ISAM but in ordered direct, random batch and sorted batch, accessing VSAM generally requires less CPU time than ISAM.
- VSAM uses less channel time than ISAM overall.
- In random access, the 'path length' from the highest level index entry down to the record is always the same in VSAM, independent of the number of insertions, whereas for ISAM the 'path length' tends to vary and become longer on the average, the more records have been inserted into a data set.
- VSAM uses free space in a far better way then ISAM. KSDS requires much less reorganization then ISAM.

5.4.1 ISAM INTERFACE PROGRAM

If a program does not require functions other than those provided by the original ISAM program, the IIP is an efficient means of using VSAM. The elapsed time and CPU time differences between IIP and native VSAM is negligible.

The ISAM Interface Program consists of a few modules which are automatically available when VSAM is used. No special generating is required. The only changes are as follows:

- The ISAM data set must be converted into a VSAM data set by using the Access Method Services functions: DEFINE CLUSTER (for defining the VSAM data set), and REPRO (to transfer the data from the ISAM data set to the VSAM data set).

- The ISAM program JCL must be changed to specify the VSAM data set.

5.4.2 ISAM TO VSAM DATA SET CONVERSION

Data and Index CI sizes should be set as discussed earlier. The overall free space can be determined from the amount of overflow in the ISAM data set. Determine the number of records that will fit into all of the ISAM overflow areas of the primary extent. The number of records in the newly created ISAM data set plus the number of records that can be contained in the overflow areas is the capacity of the primary ISAM extent. The difference between the newly created ISAM file size and the capacity of the primary extent is the amount by which the ISAM file may grow before a secondary extent is allocated. Use this amount of growth (in records) to compute the VSAM free space as discussed earlier.

5.4.3 ISAM PROGRAM CONVERSION USING ISAM INTERFACE

As described in the previous sections ISAM programs and ISAM data sets can be transferred easily to VSAM.

The ISAM data must be transferred into a VSAM data set (with REPRO or a similar user program).

The definition parameters of the VSAM data set are based on the characteristics of the ISAM data set. As ISAM differentiates between variable-length and fixed length records and blocked and unblocked records, these characteristics must be taken into account when defining the VSAM data set.

The following figure shows the definition differences for fixed blocked and fixed-unblocked records (the characters A, B, C represent the key of the records):

The ISAM definitions are for DOS/VS, the OS/VS DCB parameters are similar.

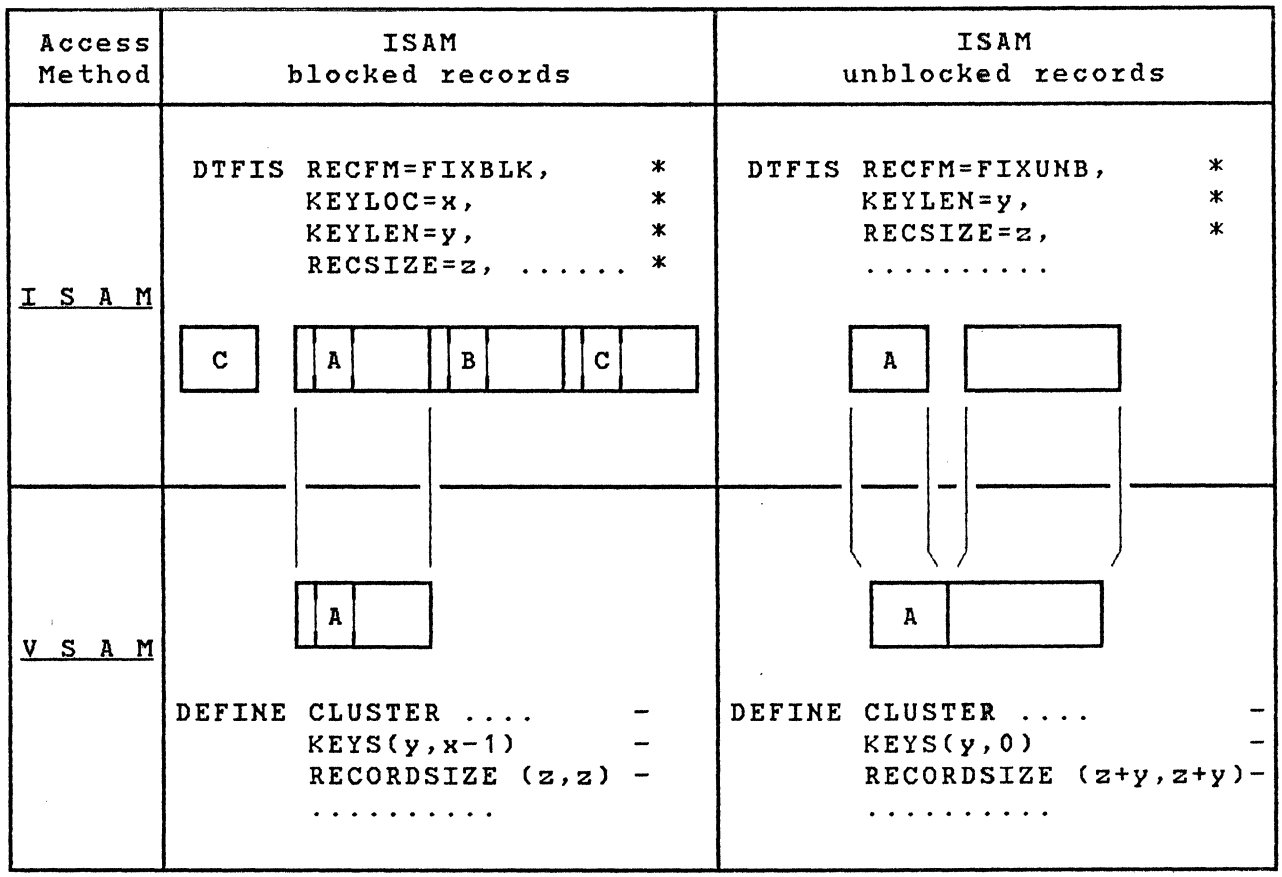


Figure 36. VSAM definition for fixed-length ISAM records

Variable-length records:

```
ISAM: KEYLEN=y,KEYLOC=x,   *   VSAM: DEFINE CLUSTER ....          -
      RECSIZE=z,           *                KEYS (y,x-4)          -
      .....                                 RECORDSIZE (avg,z-4)  -
                                            .....
```

After copying the ISAM data into the VSAM data set, the ISAM program JCL must be changed to specify the VSAM data set instead of the ISAM data set.

The ISAM program itself need not to be changed (as long as standard operations are performed).

When VSAM recognizes that ISAM macros are executed for a VSAM data set, the ISAM Interface Routines are called automatically (without any specification of the user) to translate the requests and also the return codes supplied by VSAM into ISAM return codes.

It is not suggested to convert an ISAM program into a VSAM program as the Interface routines are very fast and use only about 11K of pageable storage.

The ISAM interface support is also provided in the High Level Languages which use ISAM.

5.5 HIGH LEVEL LANGUAGE VSAM SUPPORT

COBOL/VS, PL/I, and RPGII (DOS/VS only) provide native VSAM support. They provide Sequential and Random accessing capabilities.

For further discussions see the appropriate COBOL/VS, PL/I, or RPGII manuals.

6.0 VSAM INSTALLATION AND USAGE OF VSAM CATALOGS WITH IPL

6.1 DOS/VS

6.1.1 VSAM RELATED PARAMETERS IN THE SUPERVISOR (DOS/VS)

```
FOPT    VSAM=YES,                    /
        GETVIS=YES,                  /
        RELLDR=YES,                  /
        .
        .
ALLOC   BG=nK,...,F1=n1K                 n = 170K->470k(depending
                                         on services used) for
                                         partitions where Access
                                         Method Services is
                                         supposed to run.
                                         Or user program size
                                         plus an amount equiva-
                                         lent to that of the
                                         working set indicated
                                         in section 9.2.1 on
                                         page 228.
VSTAB   .
        VSIZE=old VSIZE + nK,        /   n > 350
        SVA=(old SVA-size + nK,...), /
        BUFSIZE=old value + m            m > 40/partition
IOTAB   .
        NRES=nr,                     /   nr < 256 (number of VSAM
        .                                Resource Usage records).
        .
ASSGN   SYSCAT,X'cuu'                    Standard assignment
                                         for master catalog.
```

6.1.2 HOW TO USE A USER CATALOG AS MASTER CATALOG (DOS/VS)

If the standard assignment of the the master catalog has to be changed, it has to be specified between the 'SET' and the 'DPD' commands in the form :

```
CAT UNIT=X'cuu'
```

This is the ONLY point where a master catalog can be reassigned. An ASSGN SYSCAT is an IPL command. It is NOT a valid Job Control (JCL) nor an Attention Routine (ATTN).

6.1.3 LOAD A NEW SDL/SVA (DOS/VS)

After IPL, a new SDL/SVA has to be loaded (using for instance the procedure 'VSAMSVA'). This has to be done just once and remains valid until the VSIZE changes or the 'WARM COPY of SVA' is rejected. See SYSGEN manual.

6.2 VS2 MVS

6.2.1 HOW TO INSTALL VSAM IN VS2 MVS

As the prime system catalog is a VSAM master catalog, VSAM support is always included in the system.

The usual way to generate a MVS system is by using another MVS system as generating system.

To create a master catalog for the new system, there are two possibilities:

1. Use the SYSGEN DATASET macro alone to create the master catalog.

   If the new master catalog is to be recoverable this version cannot be used. If this version is used and a master password is to be assigned to the master catalog (strongly recommended) this master password must be added with an ALTER command (see example on page 121).

2. Use the DEFINE MASTERCATALOG command and the DATASET macro.

   This version allows specification of all attributes valid for the DEFINE MASTERCATALOG command, which includes the attribute RECOVERABLE and the specification of passwords.

Create the master catalog with the DATASET macro alone:

Specify the following:

```
DATASET  VSCATLG,VOL=(SYSRES,3330),SPACE=(CYL,(10,5)),          *
         NAME=SYS1.MASTERCT
```

Note: The master catalog need not be on the systems residence device. The resulting catalog is nonrecoverable.

This DATASET macro results in 3 major functions:

- DEFINE MASTERCATALOG (is treated as a DEFINE USERCATALOG in the generating system).
- After defining the catalog, an EXPORT DISCONNECT command is issued to detach the catalog from the generating system.
- A member with the name SYSCATLG is created in SYS1.NUCLEUS (this is the standard member pointing to the master catalog, see also the next section).

Create the master catalog with pre-allocation and DATASET macro:

Specify the following:

```
DEFINE MASTERCATALOG (NAME(SYS1.MASTERCT)          -
                      FILE (CATVOL)                -
                      RECOVERABLE                  -
                      VOLUME (SYSRES)              -
                      CYL (10 5)                   -
                      MASTERPW (MASTPROT) )
```

After executing this command (which is treated like a DEFINE USERCATALOG command) an EXPORT DISCONNECT command must be issued to prevent the usage by the generating system.

In the subsequent SYSGEN for the new system, the DATASET macro must not specify a SPACE value as follows:

```
DATASET  VSCATLG,VOL=(SYSRES,3330),                    *
         NAME=SYS1.MASTERCT
```

The DATASET macro creates a member with the name SYSCATLG in SYS1.NUCLEUS (this is the standard member pointing to the master catalog, see also the next section.

6.2.2 HOW TO USE A USER CATALOG AS MASTER CATALOG (MVS)

A member of SYS1.NUCLEUS must be defined to point to the alternate catalog (the user catalog to be used as master catalog). This member consists of a 80 byte record as follows:

Columns	Contents
1 to 6	Volume serial of volume containing the alternate catalog
7	unused
8	Device type (see following chart)
9 to 10	unused
11 to 54	Data set name of the alternate catalog left-justified and padded with blanks
55 to 80	unused

The device type codes (used in column 8) and JCL block sizes are as follows:

Device	Code (hex)	Card Punch	BLKSIZE (in JCL)
2305-1	06	12-6-9	14136
2305-2	07	12-7-9	14660
2314	08	12-8-9	7294
3330	09	12-1-8-9	13030
3330 Mod.11	0D	12-5-8-9	13030
3340/3344	0A	12-2-8-9	8368
3350	0B	12-3-8-9	19069

The following job can be used to place the member in SYS1.NUCLEUS:

```
//ADD       JOB  ...
//STEP      EXEC PGM=IEBGENER
//SYSIN     DD  DUMMY
//SYSUT2    DD  DSN=SYS1.NUCLEUS(xxxx),DISP=(MOD,KEEP),UNIT=unit,
//              DCB=(BLKSIZE=****),VOL=SER=volser
//SYSPRINT DD  SYSOUT=A
//SYSUT1    DD  *
        data card
/*
```

xxxx = name of the new member with the alternate catalog pointer
**** = see BLKSIZE in device type code table on previous page

Note: Do not use IEBUPDTE to add the member because this alters the blocksize of SYS1.NUCLEUS and makes further changes impossible.

If it is necessary to change the member, use IEHPROGM to scratch the member and then add it again.

To use the new catalog after the next IPL (cold start) the pointer in IEAVNP11 (in the NUCLEUS) pointing to SYSCATLG (SYSCATLG is the standard member containing the pointer to the VSAM master catalog) must be changed to point to the new member (see the following example).

To change the pointer in IEAVNP11 the following job can be used:

```
//ZAP       JOB  ...
//STEP      EXEC PGM=AMASPZAP
//SYSLIB    DD  DSN=SYS1.NUCLEUS,DISP=OLD
//SYSPRINT DD  SYSOUT=A
//SYSIN     DD  *
 NAME  IEAVNP11  IEAVNP11
 VER   1905  E2E8E2C3 C1E3D3C7
 REP   1905  new member name
/*
```

Note: Before applying this change, use the AMASPZAP service aid program to dump CSECT IEAVNP11. Then use the dump to determine the exact location of the data to be altered.

The new catalog is used after the next IPL (cold start).

6.3 VS1 AND VS2 SVS

6.3.1 HOW TO INSTALL VSAM IN VS1 AND VS2 SVS

VSAM is included in the system with the System Generation by default.

To use VSAM a VSAM Master catalog must be created using the VSAM utility Access Method Services (see example in section 7.3 on page 100).

This DEFINE MASTERCATALOG places a pointer into the OS/VS system catalog with the name AMASTCAT (independent of the specified master catalog name).

After having a VSAM master catalog defined, other VSAM data sets and/or user catalogs may be defined.

6.3.2 HOW TO USE A USER CATALOG AS MASTER CATALOG(SVS/VS1)

As the format of a VSAM user catalog and the VSAM master catalog is identical, any VSAM user catalog can be used as VSAM master catalog.

This is very important if the VSAM master catalog cannot be accessed due to any reason. So a user catalog can be used as a master catalog and the damaged master catalog can then be connected to the new master catalog as a user catalog to be restored.

To use a user catalog as the VSAM master catalog, the pointer in the OS/VS System catalog must be changed with the following Job:

```
//CHGCAT   JOB  ...
//STEP1    EXEC PGM=IEHPROGM
//SYSPRINT DD   SYSOUT=A
//DD1      DD   UNIT=disk,VOL=SER=mcatvolser,DISP=OLD
//DD2      DD   UNIT=disk,VOL=SER=ucatvolser,DISP=OLD
//SYSIN    DD   *
  UNCATLG DSNAME=AMASTCAT
    CATLG DSNAME=AMASTCAT,VOL=3330=ucatvolser
/*
```

After executing this job, the new catalog is used as master catalog after the next IPL (cold start).

7.0 HOW TO START USING VSAM

After VSAM is installed in your Operating System you may begin to use it.

Access Method Services commands must be used to define the VSAM catalog and your VSAM data sets (in MVS the VSAM master catalog already exists when the system is IPL-ed).

General comments on this chapter:

- To show the difference between DOS/VS and OS/VS, all the examples are shown on the same page in two columns.
- To be able to print two columns on normal size paper, some Access Method Services messages had to be abbreviated.
- To reduce the length of the output, most blank lines have been deleted.
- The DOS/VS examples were executed with DOS/VS Rel.33.

7.1 HOW TO CODE ACCESS METHOD SERVICES COMMANDS

The following is a short description of how to code an Access Method Services command. A detailed description is included in the appropriate Access Method Services manual.

Commands and their parameters can be coded in free format as shown below:

```
COMMAND sep PARM1 sep PARM2 sep PARM3 cont
                PARM4 term
```

where:

sep (separator) : 1 or more blanks, comma(,) or comments.

cont (continuation) : there are two different types of continuation:

1. A value, subparameter or parameter has to be continued.
   Plus (+) indicates the continuation of a value within a parameter. The remainder of the value has to start at the left margin on next line.

2. A command has to be continued (more parameter to follow).
   Minus (-) indicates the continuation or the same command on the next line(s).

term (terminators) : right margin or absence of continuation, or semicolon(;) or blank line.

comments format : /* any text */ . A comment line placed between 2 command lines must have a continuation sign (-).

left/right margins : the default left and right margins are at columns 2 and 72. They can be changed with the PARM MARGINS command.

Some rules and recommendations:

- Abbreviations of commands and keyword parameters can be intermixed with full naming.

- An easy check of the continuation rules within a command verb is possible if the continuation sign (-) is always placed in the same column (see following examples).

  Note: If no continuation sign (-) is coded and more parameters belonging to the same command follow, the subsequent parameters are ignored and defaulted by VSAM up to the next command (if required parameters are missing, an error message is issued). As this may lead to unexpected definition values, it is suggested to issue a LISTCAT command following each DEFINE command.

- There are two types of parameters : positional and keyword parameters. In a parameter list or sublist, positional parameters must always be specified first (positional parameters are required parameters.

  Example: INFILE (dname ENV (...)) correct
  INFILE (ENV (...) dname) wrong

- Neither commands nor keyword parameters (in long or short form) are reserved words: therefore it is dangerous and not suggested to use parameters or keywords as component names (e.g. dont use UPDATE as a data set name, as this is a parameter for Access Method Services).

- DOS/VS only : when Access Method Services is used for tape operations specific 'SYSnnn', is required, i.e.:

  SYS004 : for input tape(s)
  SYS005 : for output tape(s)

  For functions using disks, any 'SYSnnn' can be chosen.
  The default assignments are:

  SYS006 : for input disk(s)
  SYS007 : for output disk(s)

7.2 EXPLANATION OF IMPORTANT DEFINE PARAMETERS

VSAM uses catalogs as a central information point for all VSAM data sets and the direct-access volumes on which they are stored. You can define a VSAM object in a VSAM catalog only by using the Access Method Services DEFINE command. Additionally, you can use the DEFINE command to define nonVSAM objects in a VSAM catalog (in DOS/VS nonVSAM objects can only be defined in a nonrecoverable catalog; in MVS nonVSAM objects may also be defined using the JCL parameter CATLG).

When you issue the DEFINE command to catalog an object, Access Method Services builds one or more catalog entries to describe the object.

The VSAM objects you can define are:

- Master catalog (in MVS this command creates a user catalog)
- User catalog
- Data space
- Cluster, or VSAM data set.
  There are three types of cluster data organization:
  - ESDS (Entry Sequenced Data Set)
  - KSDS (Key Sequenced Data Set)
  - RRDS (Relative Record Data Set)
- Alternate index
- Path
- NonVSAM data set (in DOS/VS only in nonrecoverable catalog and for compatibility purposes)

In MVS systems additionally the following can be defined:

- Alias (also in SVS possible)
- Generation Data Group (GDG)
- Page space

Many parameters may be specified for the various DEFINE commands. The most important ones are described in detail (others are described with the example):

- FILE
- FREESPACE
- IMBED/REPLICATE
- KEYRANGES
- NAME
- RECORDSIZE
- SHAREOPTIONS
- SPEED/RECOVERY
- SUBALLOCATABLE/UNIQUE

7.2.1 FILE

The FILE parameter refers to the 'filename' of a DLBL statement (DOS/VS), or to a 'dname' of a DD statement (OS/VS).

The FILE parameter and its associated JCL statement is always required, when a volume has to be mounted to change the VTOC and/or the CRA on that volume (not MVS, see 'MVS note' below).

MVS note: FILE may be omitted and the required volume is dynamically allocated (assuming the volume has been mounted with the correct attributes).

The following commands require a FILE parameter (not MVS, see 'MVS note'):

- DEFINE MASTERCATALOG
- DEFINE USERCATALOG
- DEFINE SPACE
- DEFINE ALTERNATEINDEX/CLUSTER with the UNIQUE attribute
- VERIFY[1]

[1] FILE is always needed for VERIFY (even for nonrecoverable catalog) and is not used to identify the CRA volume, but to identify the DD statement which names the data set to be opened and verified.

Most Access Method Services commands modifying a recoverable catalog must have a FILE parameter and its associated JCL statement (in OS/VS the JCL statement may be omitted (not suggested), if the requested volume is already allocated by an other JCL statement or when the catalog itself resides on the volume).

The FILE parameter is required for the following commands when using a recoverable catalog (not MVS, see 'MVS note'):

- ALTER
- DEFINE commands
- DELETE
- IMPORT[2]
- IMPORTRA[2]

[2] FILE is only needed in a nonMVS system if the components are unique and reside on different device types (FILE must be coded separately for the data and index components); otherwise FILE is not needed.

7.2.2 FREESPACE

A KSDS can be specified to reserve space to be held free when loading data (see also section 2.5 on page 20).

The specified amount is only held free, when sequential insertions, such as 'loading' or 'mass insertion' (see section 8.8 on page 207) are performed.

Two values can be specified: FREESPACE(CI-percent CA-percent).

- 'CI-percent' specifies the amount of space to be held free per control interval (CI).

  The way VSAM treats this free space definition is: specified percentage times actual CI size (rounded down to a full byte). So VSAM does not care about record length and does not round the value up, if the amount of free space is less then one record.

  Example:

  Definition:

  Control Interval Size = 1024 bytes
  Free space = 15 % free CI space (FREESPACE (15 0)

  VSAM calculation:

  1024 * 15 % = 153,6 = 153 bytes minimum free space (excluding control fields).

- 'CA-percent' specifies the amount of control intervals to be held free per control area (CA).

If the maximum value (100%) is specified for both Ci-percent and CA-percent, each control interval will contain one record and each control area will contain one used control interval.

Section A.6.2 on page on 258 contains a description of FREESPACE considerations when issuing the IMPORT command. For a special definition using different FREESPACE values, see also section 8.4 on page 199.

7.2.3 IMBED/REPLICATE (INDEX OPTIONS)

When a key-sequenced data set is processed sequentially, the sequence set index level is used to indicate the order in which control intervals are to be accessed. To improve performance during sequential processing, the index sequence set can be separated from the rest of the index component (index set levels) and stored with the logical records in the data component (IMBED). When this option is chosen, the index records for a control area are placed on the first track of the control area that both index and logical records can be accessed without moving the disk arm (similar to the location of the track index within the prime area in an ISAM data set).

When the index sequence set is stored within the data component (IMBED), sequence set records are also replicated. That is, each sequence set index record is allocated one track at the beginning of the control area. The index record is duplicated on the track as many times as it will fit. This technique significantly minimizes the rotational delay involved in arriving at the beginning of an index record. If there is only one control area in a cylinder, index sequence set records will be replicated beginning with track 0. If there are two control areas in a cylinder, initial tracks of the first area will contain replicated index records for the first control area, while initial tracks of the second area will contain replicated index records for the second control area.

Index set records, like index sequence set records, contain compressed index entries. The index entries in each level of the index set point to index records of the next lower index level. An index entry within the index set contains a pointer to an index record, the highest key in that index record, and control information. Index set levels can also be replicated (REPLICATE). When this option is chosen, one track is required for each index record in the entire index set. An index record is duplicated on its assigned track as many times as it will fit.

The index set may not be replicated when the index set and the sequence set of the primary index are physically separate (sequence set stored with logical records). However, when the index set and the sequence set are stored together, both are replicated or neither is replicated.

The following combinations are possible:

- NOIMBED NOREPLICATE:
  All index levels are stored together and are not replicated.

- NOIMBED REPLICATE:
  All index levels are stored together, each index record occupies an own track and is repeated on this track as often as possible.

- IMBED NOREPLICATE:
  The index sequence set level is stored with the data component. Each index record in the index sequence set is stored with its data control interval, occupies an own track, and is repeated on this track as often as possible. In most cases this is the best choice. The higher index levels are not repeated on a track.

- IMBED REPLICATE:
  The index sequence set level is stored with the data component. Each index record in the index sequence set is stored with its data control interval. Each index sequence set record occupies its own track, and is repeated on this track as often as possible.

7.2.4 KEYRANGES

KEYRANGES is a DEFINE parameter and specifies that in a KSDS specific key ranges are to be stored on specific volumes.

As the user knows (or should know) which volume is to be used for a specific key range, he can mount the volume he needs with a so called subset mount (see also section 8.10 on page 216).

The allocation algorithms (amount of primary and secondary space) and the rules for multivolume data sets are explained in the section 8.1.3.3 on page 188.

When using key ranges, data on a specific volume can be restored from a backup tape (with full volume restore, which may lead to a data set 'out-of-synch' condition), as usually all keys of a key range are stored on one volume (see following examples).

- Example 1 (3 key ranges, 3 volumes)

```
DEFINE CLUSTER ... VOLUMES (V1,V2,V3)      -
       KEYRANGES ((A,F) (G,P) (Q,Z))       -
       CYL (100,50)
```

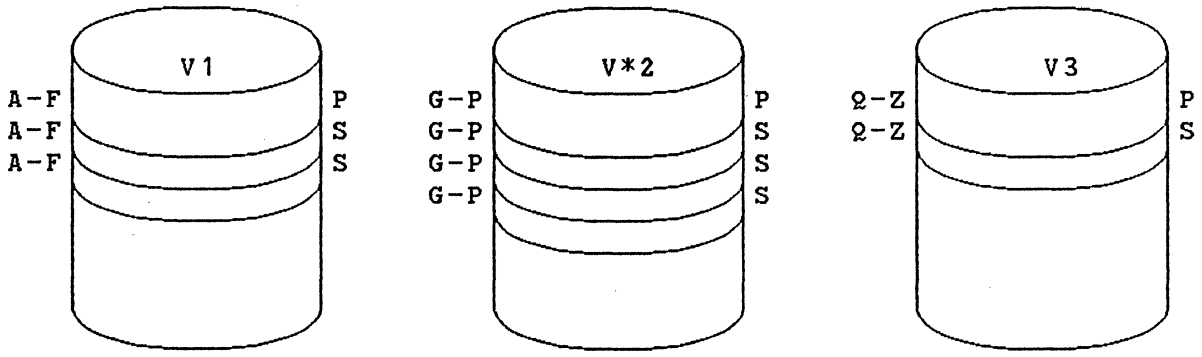
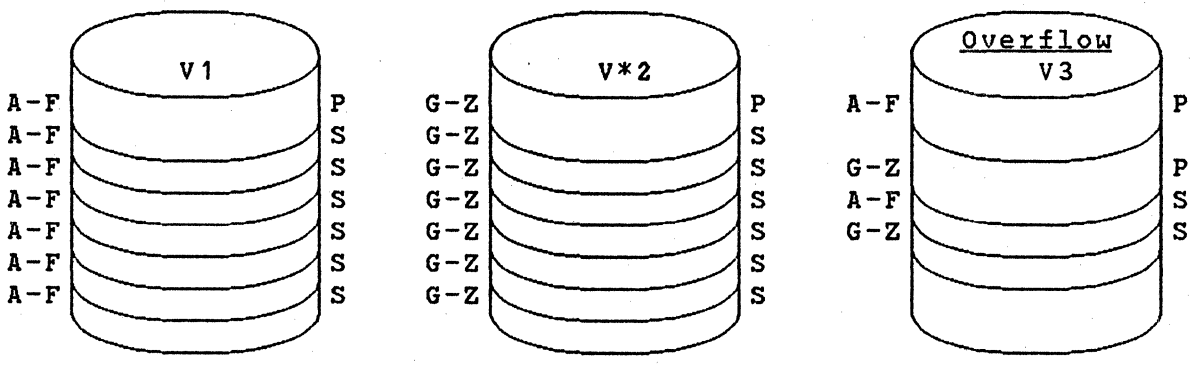


Figure 37. Keyranges (example 1).

- Example 2 (2 key ranges, 3 volumes)

```
DEFINE CLUSTER ... VOLUMES (V1,V2,V3)      -
       KEYRANGES ((A,F) (G,Z))             -
       CYL (100,50)
```

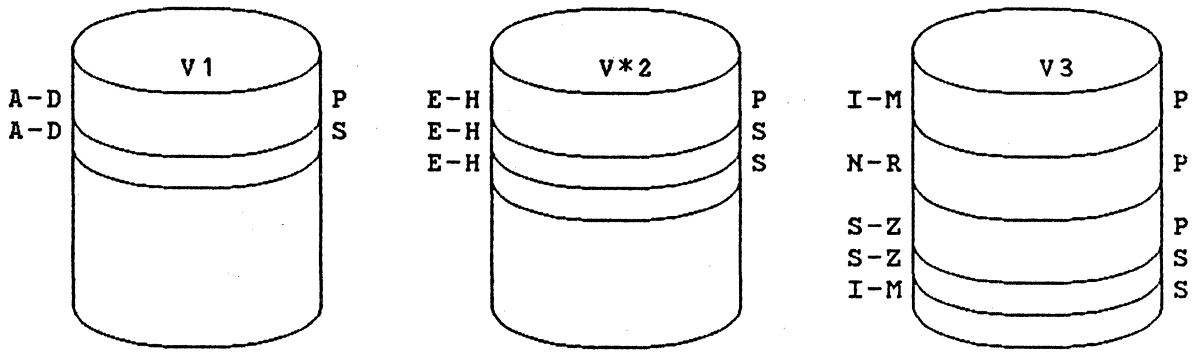


P = Primary allocation
S = Secondary allocation

Figure 38. Keyranges (example 2).

- Example 3 (5 key ranges, 3 volumes)

```
DEFINE CLUSTER ... VOLUMES (V1,V2,V3)            -
       KEYRANGES ((A,D) (E,H) (I,M) (N,R) (S,Z)) -
       CYL (100,50)
```

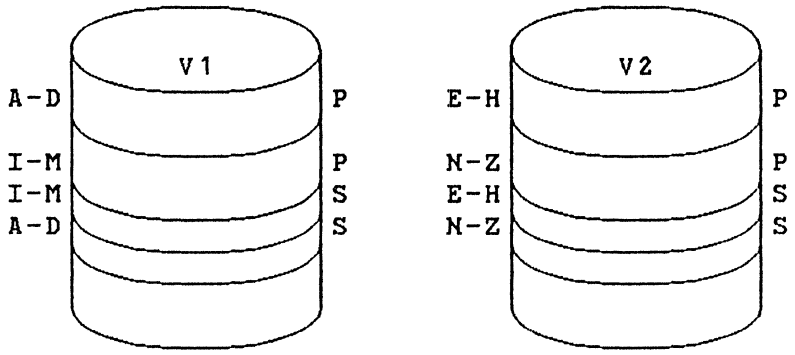


P = Primary allocation
S = Secondary allocation

Figure 39. Keyranges (example 3).

- Example 4 (4 Keyranges, 2 volumes (twice specified))

```
DEFINE CLUSTER ... VOLUMES (V1,V2,V1,V2)  -
       KEYRANGES ((A,D) (E,H) (I,M) (N,Z))-
       CYL (100,50)
```



P = Primary allocation
S = Secondary allocation

Figure 40. Keyranges (example 4).

7.2.5 NAME

The name specified for a VSAM data set may contain 1 - 44 alphanumeric characters, national characters (@, # and $) and two special characters (hyphen and 12-0 overpunch).

Names containing more than eight characters must be segmented by periods; one to eight characters may be specified between periods.

The first character of a name or name segment must be either an alphabetic or national character.

With multiple catalogs you should take care that a data set name in one catalog is not duplicated in another catalog.

It is possible to have the same data set name in more than one catalog.

Access Method Services prevents you from cataloging two objects with the same name in the same catalog, and from altering the name of an object that its new name duplicates the name of another object in the same catalog.

Duplication is not checked or prevented when a user catalog is imported into a system (see IMPORT CONNECT command on page 166).

No check is made to determine whether the imported catalog contains a name that another catalog already in the system contains. For cluster naming conventions see also section 2.9.2 on page 43.

In MVS systems, the data set name can also be used to identify the catalog where the data set is cataloged. A name segmented with periods is called a qualified name. The first qualifier (the part of the name up to the first period) is used to find the user catalog if no STEPCAT or JOBCAT catalog is specified (the Access Method Services manual contains a section 'Order of catalog use' for each command, where the catalog search sequence is described).

Note (MVS only): As the data set name (qualified name) is used for catalog allocation a qualified name and an unqualified name cannot exist in the same catalog if the first qualifier of the qualified name is the same as the unqualified name.

The example in section 4.4 on page 60 does not work in a MVS system and is rejected with a message 'DUPLICATE DATA SET NAME', because the cluster name is CITY and the data component name is 'CITY.D' which is then a violation of the restriction described above.

To prevent these problems, it is suggested always to use qualified names in an MVS system and always use the same first qualifier in one catalog.

This first qualifier should be identical with the ALIAS or the catalog name defined for the user catalog.

7.2.6 RECORDSIZE

As described on page 8 there is no special parameter to define that fixed-length records or variable-length records are to be used.

The RECORDSIZE parameter indicates, however, if fixed-length records are to be used or not.

The format of RECORDSIZE is:

```
RECORDSIZE (avg.length  max.length)
```

If fixed-length records with a length of 80 bytes are to be used, the definition would be RECORDSIZE (80 80).

The definition of RECORDSIZE (60 80) would, however, also enable the user to use fixed-length records with 80 bytes, because the 'avg.length' parameter is only used by VSAM to calculate the minimum control interval size for the data component (or to check the size the user has defined).

The 'avg.length' parameter is also checked when an RRDS is being defined (in this case 'avg. length' and 'max. length' must have the same value, since fixed-length records are required for RRDS).

The 'max.length' parameter is a limiting size for VSAM. Records with a larger size will force an error message, but if RECORDSIZE (100 100) has been specified and only records with a length of 80 bytes are used, neither error messages are written nor is space wasted on disk (except for an RRDS data set whose slots are preformatted depending on the 'max.length' parameter), as VSAM checks the actual length of the record and stores this information in the RDF (Record Definition Field).

7.2.7 SHAREOPTIONS AND DATA SET SHARING

A VSAM data set can be accessed concurrently by two or more subtasks within the same partition and two or more job steps (partitions) (in OS/VS DISP=SHR must be specified for the VSAM data set by each job step). Both types of sharing can be used for a VSAM data set at the same time. The type of data set sharing permitted for two or more partitions is controlled by using the SHAREOPTIONS parameter of the DEFINE command when the VSAM data set is defined.

The format of the SHAREOPTIONS command is as follows:

```
SHAREOPTIONS (cross-region [cross-system] )
```

7.2.7.1 CROSS-PARTITION/REGION SHARING

The following types of options are supported:

- SHAREOPTIONS (1):

  The data set can be opened by one user for output processing (to update or add records) or the data set can be opened by multiple users for read operations only. Full read and write integrity is provided by this option.

- SHAREOPTIONS (2):

  The data set can be opened by one user for updating or record addition (output operations) and by multiple users for read-only processing. Since only one user can perform write operations, write integrity is provided by this option. However, read integrity must be provided by each user since users can read a record that is in the process of being updated.

- SHAREOPTIONS (3):

  The data set can be opened by any number of users for both read and write operations. Data set integrity must be maintained by the user. No integrity (read or write) is provided by VSAM.

- SHAREOPTIONS (4):

  The data set can be opened by any number of users for both read and write operations. For direct processing operations, VSAM provides a new buffer for each request. Control interval splitting should be avoided when this option is used (to prevent a possible CA split).

  In DOS/VS the 'trackhold facility' is used to insure the appropriate integrity.

  In OS/VS the ENQ and DEQ macros must be issued by users to maintain data integrity.

  VSAM will not allow a control area split (and no CI split of the 'high key CI' for SHAREOPTION 4 sharing of a key-sequenced data set. VSAM indicates 'NO SPACE AVAILABLE' if an attempt is made to add or change the size of a record and a control area split is required to perform the operation.

  OS/VS Notes: Data set sharing by subtasks within the same partition can be accomplished using one DD statement for the VSAM data set or multiple DD statements. When a single DD statement is used, multiple subtasks in the same partition can perform read and update operations on the VSAM data set. VSAM uses the exclusive control facility to maintain integrity during update operations. The SHR disposition parameter need not be specified in order to share a VSAM data set when one DD statement is used. However, if DISP=SHR is specified when one DD statement is used, both subtask sharing and cross-partition sharing (as described above) can be used concurrently.

  When multiple DD statements are used, multiple subtasks within a partition can share a VSAM data set using the same options as are supported for cross-partition sharing. The DISP=SHR parameter must be specified on the DD statements.

  Note the following restriction: When DISP=SHR and SHAREOPTION 4 is specified for cross- partition or cross-system sharing, VSAM provides a new buffer for each direct processing request, and users should be issuing ENQ/DEQ or RESERVE/RELEASE macros to ensure data set integrity.

7.2.7.2 CROSS-SYSTEM SHARING (OS/VS ONLY)

The following options are supported in OS/VS (they may be specified in DOS/VS but are meaningless):

- SHAREOPTIONS (x 3):

  The data set can be opened by any number of users for both read and write operations. Data set integrity is a user responsibil-

ity as VSAM does not provide any.

- SHAREOPTIONS (x 4):

  The data set can be opened by any number of users for both read and write operations. VSAM provides a new buffer for each direct processing request and RESERVE and RELEASE macros must be issued by users to maintain data set integrity. All job steps that are accessing a VSAM data set concurrently must specify DISP=SHR if data set integrity is to be maintained. Control interval splitting should be avoided.

  Note the following OS/VS restriction: When DISP=SHR and SHAREOPTION 4 is specified for cross- partition or cross-system sharing, VSAM provides a new buffer for each direct processing request, and users should be issuing ENQ/DEQ or RESERVE/RELEASE macros to ensure data set integrity.

  VSAM will not allow a control area split (and no CI split of the 'high key CI' for SHAREOPTION 4 sharing of a key-sequenced data set. VSAM indicates 'NO SPACE AVAILABLE' if an attempt is made to add or change the size of a record and a control area split is required to perform the operation.

7.2.8 SPEED/RECOVERY (LOADING OPTIONS)

When a VSAM data set is loaded, VSAM does or does not preformat control areas, depending on the attribute specified when the data set is defined, RECOVERY or SPEED, respectively.

When RECOVERY (the default) is specified, during loading VSAM preformats each control area immediately before loading any records into it. Preformatting for a key-sequenced data set consists of putting the appropriate control information in each control interval and an end-of-file indication in the first control interval in the next control area after the control area just preformatted. All zeros in the control interval definition field indicates end of file or end of key range for a key-sequenced data set.

For an entry-sequenced or relative record data set, control information and an end-of-file indication are placed in each control interval of the control area during preformatting.

The RECOVERY option (not suggested) ensures that if an error occurs, that prevents further processing while a control area is being loaded, the previously loaded control areas are not lost. Loading can resume from the first or only end-of-file indicator. Preformatting in RECOVERY mode is always done when records are added to an existing VSAM data set (even if SPEED was specified).

When SPEED is specified (suggested), records are loaded (i.e. between first OPEN and CLOSE) without preformatting each control

area before loading and the end-of-file indicator is not written until the data set is closed. When this option is chosen, loading proceeds more rapidly, but if an error that prevents further processing occurs, all the records loaded up to that point may be lost and loading would have to resume at the beginning of the data set.

The following should be considered when using RECOVERY:

- Up to now when an empty data set is loaded, the 'high used RBA' (see page 156) is only updated if the data set has been closed (by the user) after at least one record written into it, or if a second extent has to be allocated.

- Since VERIFY does not work for a data set with a 'high used RBA = 0', which usually indicates an empty data set, loading could not be resumed, even when RECOVERY is specified.

- If the user-written load program had issued a CLOSE while loading the data set (while REPRO does not), VERIFY may be used after a system malfunction to adjust the 'high used RBA' and the loading may be resumed (the user has to check which record was entered last and he has to change the load program to resume loading with the next record).

- It is suggested to always specify SPEED which is not the default.

7.2.9 SUBALLOCATION/UNIQUE

As described in section 2.9 on page 40 VSAM uses two different types of data spaces/data sets in terms of allocation.

The SUBALLOCATION parameter specifies that a data set shares a predefined VSAM data space with other VSAM data sets.

VSAM does all the space management and no JCL specification for the location of the data set must be given (DOS/VS).

SUBALLOCATION is the way of allocation VSAM is designed for.

For DOS/VS suballocated data sets offer a similar DASD management as DADSM provides for OS/VS.

For OS/VS the advantage (if using suballocated data sets) is that the 16 extent restriction per volume does not apply and therefore the disk space can be used more flexibly.

The UNIQUE parameter requires a separate allocation for the data set.

DOS/VS user must specify the number of tracks and the offset in the DLBL EXTENT statement when a UNIQUE data set is to be defined.

One F1-DSCB (Label) (or two for a KSDS) are written into the VTOC for the UNIQUE data set.

The name in the F1 DSCB is the data component name (and for a KSDS the index component name).

The cluster name specified in the DEFINE command is not used in the VTOC. Therefore it is recommended to specify also a name for the data component (and in case of a KSDS, also a name for the index component), since otherwise VSAM would generate its own 44-Byte name which is meaningless to the user when listing the VTOC or the catalog (LISTCAT).

UNIQUE data sets should be restricted for special applications.

DOS/VS users should also read the restrictions for UNIQUE data sets when using recoverable catalogs (section A.4 starting on page 250).

7.3 DEFINE MASTERCATALOG

As described in chapter 3.0 starting on page 47, the first step of creating VSAM objects, is to define the VSAM master catalog (see also the appropriate sections in chapter 6.0 starting on page 79). In MVS systems, a VSAM master catalog cannot be defined for the active system; each DEFINE MASTERCATALOG is treated as a DEFINE USERCATALOG.

Since any user catalog, however, can be used as master catalog (see the appropriate section in chapter 6.0), this discussion is also valid for MVS systems.

1. Since it is advisable that the master catalog (MCAT) should contain only the UCAT's pointers (plus CVOL pointers and ALIAS entries for MVS systems) it can be defined with CYL (1 1), if it is a nonrecoverable catalog (see also section A.4 on page 250).
2. As described in section A.4.3.1 on page 252 the master catalog need not be RECOVERABLE.
3. Even when the original space definition was large enough to hold all future entries, a secondary allocation value should be specified, as it is a very time consuming process to recover from a 'catalog full' condition.
4. The master catalog should be made password protected in order to prevent its accidental (or unauthorized) deletion or update. If the master catalog is password protected, severe delete functions, such as DELETE USERCATLOG FORCE (OS/VS only) or ALTER REMOVEVOLUMES (a special OS/VS function to delete all VSAM information on a disk, see example on page 181) are prevented as long as the master password is not specified.

DOS/VS example (input):

```
// JOB DEFINE MASTER CATALOG
// DLBL IJSYSCT,'PRIMER.MCAT',99/365,VSAM
// EXTENT SYSCAT,DOS30Z,,,4560,38
// EXEC IDCAMS,SIZE=AUTO

        DEF MCAT  (                     -
                  NAME(PRIMER.MCAT)     -
                  FILE (IJSYSCT)        -
                  VOLUME (DOS30Z)       -
                  CYL (1 1)             -
                  MASTERPW (MCATMRPW)   -
                  )
/&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0012 EXEC PGM=IDCAMS
//VOL1     DD VOL=SER=WTVSMC,DISP=OLD,UNIT=3330
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
        DEF MCAT  (                     -
                  NAME (PRIMER.MCAT)    -
                  FILE (VOL1)           -
                  VOL (WTVSMC)          -
                  CYL (1 1)             -
                  MASTERPW (MCATMRPW)   -
                  )
/*
```

DOS/VS example (output):

```
IDC0510I CAT. ALLOC. STAT. FOR VOLUME DOS30Z IS 0
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
IDC0510I CAT. ALLOC. STAT. FOR VOLUME WTVSMC IS 0
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Notes:

- The NAME parameter specifies the name of the master catalog.
- The FILE parameter is required (not for MVS, see also section 7.2.1 on page 88).
- The VOL or VOLUME parameter specifies the volume the catalog is to be allocated.
- The CYL or CYLINDERS parameter specifies the amount of primary and secondary space to be allocated for the VSAM data space, which is created with this command (see also section 2.11 on page 45, 189).

  Since no DATA and INDEX parameters are specified the master catalog occupies the whole data space (see also begin of next section DEFINE USERCATALOG).
- The MASTERPW parameter specifies the master password (MCATMRPW) for the master catalog (see also section 3.3 on page 52).

DOS/VS notes:

- The 'dname' must be IJSYSCT in both the DLBL and the FILE statements.
- The EXTENT must begin on a cylinder boundary. Since the primary allocation is one cylinder and the catalog is not RECOVERABLE, an EXTENT specification of two cylinders (38 tracks on 3330) leaves one cylinder for catalog extension or suballocatable space since the master catalog will contain only user catalog pointers.
- No ASSGN is necessary since it has been specified either in the supervisor or at IPL-time.
- The DLBL/EXTENT information will be stored in the STDLABEL area, so they have been omitted in the following examples.
- The line ' //EXEC IDCAMS SIZE=AUTO ' is followed by one or more Access Method Services commands as printed in the output.
- A '/*' signals end of input to Access Method Services.

OS/VS notes:

- The line ' //SYSIN DD * ' is followed by one or more Access Method Services commands as printed in the output.
- A '/*' or '//' (next step) signals end of input.

7.4 DEFINE USERCATALOG

As discussed in chapter 3.0 starting on page 47 user catalogs should be used to ease transportation of VSAM data sets and catalog recovery.

DEFINE USERCATALOG creates a suballocatable data space on a volume not owned by any other VSAM catalog. If the catalog entries are changed often by deleting or adding of VSAM objects, it might be advisable to reserve for the catalog its own data space to prevent secondary allocation being built for form the catalogs prime allocation.

To do so, do not specify DATA or INDEX parameters in the DEFINE USERCATALOG command. Thus the whole space specified with the SPACE parameter will be assigned to the user catalog (the UNIQUE attribute is not allowed in this command).

In DOS/VS the size-definition in the EXTENT parameter determines, only if the catalog resides in this space.

To define the user catalog the following assumptions were made:

- The user catalog occupies its own data space (therefore neither DATA nor INDEX parameters were specified).
- The update password allows definitions of VSAM objects without knowing the master password. The master password is required for all other functions (like deleting entries, alter passwords etc.) but the read-only ones. It is also required for the function LISTCAT ALL (when the passwords are to be listed).
- The user catalog is recoverable, therefore a CRA (catalog recovery area) with a size of one cylinder is allocated in addition to the specified catalog space, either by OS-DADSM or by DOS/VS using space specified in the EXTENT statement.
- In our example, the UCAT has to be large enough to contain (long range planning):

  - 15 KSDS clusters
  - 6 ESDS clusters
  - 2 RRDS clusters
  - 5 Alternate indexes
  - 5 Paths
  - 2 Volumes owned by the catalog.

The number of tracks required can be calculated using the worksheet shown in the section 'Estimating the Catalog's Space Requirements' of the Access Method Services manual (see page 2).

Based upon the previous assumptions, the worksheet result is a minimum of 99 entries or 15 tracks for the primary allocation. This value was rounded up in the example to 2 cylinders.

DOS/VS example (input):

```
// JOB DEFINE USER CATALOG
// DLBL IJSYSUC,'PRIMER.UCAT1',,VSAM
// EXTENT SYS010,WTVSAM,,,19,57
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
        DEF UCAT  (                      -
                   NAME (PRIMER.UCAT1)   -
                   FILE (IJSYSUC)        -
                   VOL (WTVSAM)          -
                   CYL (2 1)             -
                   MASTERPW (UCATMRPW)   -
                   UPDATEPW (UCATUPDT)   -
                   RECOVERABLE           -
                   )
/&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0022 EXEC PGM=IDCAMS
//VOL1     DD VOL=SER=WTVSAM,DISP=OLD,UNIT=3330
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
         DEF UCAT  (                     -
                    NAME (PRIMER.UCAT1)  -
                    FILE (VOL1)          -
                    VOL (WTVSAM)         -
                    CYL (3 1)            -
                    MASTERPW (UCATMRPW)  -
                    UPDATEPW (UCATUPDT)  -
                    RECOVERABLE          -
                    )
/*
```

DOS/VS example (output):

```
IDC0510I CAT. ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
IDC0510I CAT. ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

LISTCAT output of this object is shown later in section 7.9.2

Notes:

- The FILE parameter is required (not for MVS, see also section 7.2.1 on page 88), as the VTOC of the volume to contain the catalog has to be modified (build F1-DSCB (Label), set VSAM 'ownership-bit' ON).

The notes are continued on the next page.

DOS/VS notes:

- The 'dname' may be any name, but choosing IJSYSUC allows the DLBL/EXTENT to be cataloged in the PARSTD area and makes this UCAT default to the user catalog.

- The EXTENT of 3 cylinders contains :

  - The UCAT primary allocation (2 cylinder)
  - The CRA (1 cylinder)

  In case of a catalog overflow, any free cylinder in the suballocatable space(s) of this catalog (not yet defined) will be used for secondary allocations.

OS/VS notes:

- The value 'CYL (3 1)' specifies, that 2 cylinders are to be used for the catalog (no DATA and INDEX parameter specified (see description for DEFINE MASTERCATALOG)) and 1 cylinder is to be used as CRA (catalog recovery area), since the RECOVERABLE parameter is specified.

  Up to a total of 15 extents can be allocated if necessary (in extents of 1 cylinder).

7.5 DEFINE SPACE

This command creates a suballocatable data space with a primary allocation of 100 cylinders.

DOS/VS example (input):

```
// JOB DEFINE SPACE
// DLBL SPACE,,,VSAM
// EXTENT SYS010,WTVSAM,,,76,1900
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
        DEF SPACE (                    -
                   FILE (SPACE)        -
                   VOL (WTVSAM)        -
                   CYL (100)           -
                   )                   -
              CAT (PRIMER.UCAT1/UCATUPDT)
/&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0032 EXEC PGM=IDCAMS
//SPACE    DD VOL=SER=WTVSAM,DISP=OLD,UNIT=3330
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
         DEF SPACE (                   -
                    FILE (SPACE)       -
                    VOL (WTVSAM)       -
                    CYL (100,2)        -
                    )                  -
               CAT (PRIMER.UCAT1/UCATUPDT)
/*
```

DOS/VS example (output):

```
IDC0511I SPACE ALLOC.STAT. FOR VOLUME WTVSAM IS 0
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
IDC0511I SPACE ALLOC.STAT. FOR VOLUME WTVSAM IS 0
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

LISTCAT output of this object is shown later in section 7.9.2

Notes:

- The FILE parameter is required (not for MVS, also see section 7.2.1 on page 88), as the VTOC of the volume to contain the data space has to be modified (build F1-DSCB (Label)), and a copy of the data space entry has to be written into the CRA.
- The CAT parameter must be coded since the UCAT is password protected.

DOS/VS notes:

- Defining a SPACE (as well as any UNIQUE cluster) requires an EXTENT specification where the object is to be placed.
- The secondary allocation for SPACE is not used in DOS/VS. Such a specification would , if indicated, be recorded in the catalog for OS/VS compatibility only.
- The FILE parameter also indicates to the DOS/VS system routines the amount fo space controlled by VSAM.

7.6 KSDS (KEY-SEQUENCED DATA SET)

7.6.1 DEFINE KSDS

DOS/VS example (input):

```
// JOB DEFINE KSDS IN VSAM SPACE
// DLBL KSDS,'CUSTOMER.K',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
       DEF CLUSTER                          -
                 (                          -
                 NAME (CUSTOMER.K)          -
                 FILE (KSDS)                -
                 VOL (WTVSAM)               -
                 CYL (5 2)                  -
                 INDEXED                    -
                 IMBD                       -
                 )                          -
           DATA  (                          -
                 NAME (CUSTOMER.K.D)        -
                 RECORDSIZE (200 200)       -
                 KEYS (5 0)                 -
                 FSPC (20 10)               -
                 CISZ (1024)                -
                 )                          -
           INDEX (                          -
                 NAME (CUSTOMER.K.I)        -
                 CISZ (2048)                -
                 )                          -
                 CAT (PRIMER.UCAT1/UCATUPDT)
/&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0042 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//SYSPRINT DD SYSOUT=A
//KSDS     DD VOL=SER=WTVSAM,DISP=OLD,UNIT=3330
//SYSIN    DD *
       DEF CLUSTER                          -
                 (                          -
                 NAME (CUSTOMER.K)          -
                 FILE (KSDS)                -
                 VOL (WTVSAM)               -
                 CYL (5 2)                  -
                 INDEXED                    -
                 IMBD                       -
                 )                          -
           DATA  (                          -
                 NAME (CUSTOMER.K.D)        -
                 RECORDSIZE (200 200)       -
                 KEYS (5 0)                 -
                 FSPC (20 10)               -
                 CISZ (1024)                -
                 )                          -
           INDEX (                          -
                 NAME (CUSTOMER.K.I)        -
                 CISZ (2048)                -
                 )                          -
                 CAT (PRIMER.UCAT1/UCATUPDT)
/*
```

DOS/VS example (output):

```
IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0509I INDEX ALLOC.STAT. FOR VOLUME WTVSAM IS 0
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0509I INDEX ALLOC.STAT. FOR VOLUME WTVSAM IS 0
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

LISTCAT output of this object is shown later in section 7.9.2

See notes on next page.

Notes:

- INDEXED specifies that the cluster is a KSDS.
- The sequence set will be IMBEDded within the data i.e. every sequence set CI will be replicated to fill the first logical track of the corresponding CA. NOREPLICATE is assumed per default for the index set.
- As DATA- and INDEX-names are specified, VSAM does not have to generate names itself.
- According to the calculation made in section 8.3 on page 197, the minimum INDEX-CISZ is 2 K. If this parameter specification is too small, a higher value will be used by VSAM without the user being notified by any kind of message. It is therefore recommended to look carefully in the LISTCAT output for any difference between what has been specified and what VSAM stated.

DOS/VS notes:

- IJSYSUC is stored in the STDLABEL area.
- The VSAM data set name in the DLBL statement is optional for DEFINE.
- The defined KSDS is suballocated (default), i.e. it occupies a certain amount of space anywhere in a pre-defined VSAM space on the volume specified in the EXTENT 'volid'. Therefore only one set of DLBL and EXTENT (short form) is required. For a UNIQUE KSDS cluster two pairs of DLBL/EXTENT (full form) would be required, one for the DATA part of the cluster, the other for the INDEX.
- Since the user catalog is recoverable, the FILE parameter is needed to address the CRA through DLBL/EXTENT and ASSGN statements.

7.6.2 REPRO (LOAD A KSDS)

DOS/VS example (input):

```
// JOB LOAD KSDS FROM DISK-SAM
// DLBL SQDSK,'KSDS.LARGE.DATA'
// EXTENT SYS000,,,,7562,57
// ASSGN SYS000,DISK,VOL=DOS30Z,SHR
// DLBL KSDS,'CUSTOMER.K',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
        REPRO                          -
              INFILE (SQDSK            -
                  ENV (RECFM (FB)      -
                       BLKSZ (1600)    -
                       RECSZ (200)     -
                       PDEV (3330))    -
                     )                 -
              OUTFILE (KSDS)           -
                  REPLACE
/&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0052 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//SQDSK    DD VOL=SER=WTVS1R,UNIT=3330,
//         DSN=LOADKSDS,DISP=(OLD,KEEP),
//         DCB=(RECFM=FB,LRECL=200,BLKSIZE=1600)
//KSDS     DD DSN=CUSTOMER.K,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
         REPRO                         -
             INFILE (SQDSK)            -
             OUTFILE (KSDS)
/*
```

DOS/VS example (output):

```
IDC0005I NUMBER OF RECORDS PROCESSED WAS 3000
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
IDC0005I NUMBER OF RECORDS PROCESSED WAS 3000
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

LISTCAT output of this object is shown later in section 7.9.2

DOS/VS notes:

- Access Method Services needs complete specifications of any nonVSAM data set.
- REPLACE means that in case the output KSDS contains records, those input records with matching keys will REPLACE the records in the KSDS (could have been omitted like in OS/VS).

7.6.3 PRINT KSDS

This example shows a partial printout of the KSDS cluster.

DOS/VS example (input):

```
// JOB PRINT KSDS (PARTIAL)
// DLBL KSDS,'CUSTOMER.K',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
         PRINT  INFILE (KSDS)        -
                CHAR                 -
                COUNT (5)
 /&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0062 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//KSDS     DD DSN=CUSTOMER.K,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
          PRINT  INFILE (KSDS)          -
                 CHAR                   -
                 COUNT (5)
/*
```

DOS/VS example (output):

```
LISTING OF DS -CUSTOMER.K
KEY OF RECORD -    10
   10                    PHOENIX         JOHNSON
KEY OF RECORD -    20
   20                    NEW YORK        ROBERTS
KEY OF RECORD -    30
   30                    SAN JOSE        NIXON
KEY OF RECORD -    40
   40                    SAN FRANCISCO   BURTON
KEY OF RECORD -    50
   50                    RENO            SMITH
IDC0005I NUMBER OF RECORDS PROCESSED WAS 5
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
LISTING OF DS -CUSTOMER.K
KEY OF RECORD -    10
   10                    PHOENIX         JOHNSON
KEY OF RECORD -    20
   20                    NEW YORK        ROBERTS
KEY OF RECORD -    30
   30                    SAN JOSE        NIXON
KEY OF RECORD -    40
   40                    SAN FRANCISCO   BURTON
KEY OF RECORD -    50
   50                    RENO            SMITH
IDC0005I NUMBER OF RECORDS PROCESSED WAS 5
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

7.6.4 ALTERNATE INDEXES AND PATHS TO KSDS

7.6.4.1 EXPLANATION OF THE MOST IMPORTANT PARAMETERS

- **KEYS** — specifies the size and offset of the alternate key field, which must be included in each base cluster record (and in case of a spanned record, in the first segment).

  The alternate key can overlap or be contained entirely with another (alternate or prime) key field. It must consist of a contiguous field.

- **RELATE** — specifies the base cluster this alternate index belongs to. The base cluster can be a nonreusable ESDS or KSDS cluster.

- **UNIQUEKEY/NONUNIQUEKEY** — specifies, whether an alternate key may occur only once in a base cluster record (UNIQUE) or whether more than one occurrences are allowed (NONUNIQUEKEY).

  If NONUNIQUEKEY is used (see figure 13 on page 33) it has to be taken into consideration (if the base cluster is a KSDS), that for each occurrence the prime key is stored in full length in the alternate index data record adjacent to the alternate key, so the alternate index RECORDSIZE should be specified large enough.

  For an ESDS base cluster 4 bytes RBA are stored for each occurrence.

- **UPDATE/NOUPDATE** — is a PATH parameter and specifies, whether an alternate index belonging to the upgrade set (see UPGRADE) is to be updated (when necessary) or not (further details see section 2.7.4 on page 35).

- **UPGRADE/NOUPGRADE** — specifies whether an alternate index is automatically updated by VSAM when base cluster records are inserted, deleted or when the alternate key field is changed (provided the base cluster is not accessed via a path with the NOUPDATE attribute.

7.6.4.2 DEFINE AIX, DEFINE PATH

DOS/VS example (input):

```
// JOB DEF AIX 'SALESMAN' + PATH 'SALES.CUST'
// DLBL KSDS,'CUSTOMER.K',,VSAM
// EXTENT SYS010,WTVSAM
// DLBL AIXK1,'SALESMAN.K',,VSAM
// EXTENT SYS010,WTVSAM
// DLBL PATHK1,'SALES.CUST.K',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
        DEF AIX   (                      -
                  NAME (SALESMAN.K)      -
                  FILE (AIXK1)           -
                  RELATE (CUSTOMER.K)    -
                  KEYS (12 40)           -
                  VOL (WTVSAM)           -
                  CYL (1 1)              -
                  IMBD                   -
                  NONUNIQUEKEY           -
                  UPGRADE                -
                  )                      -
           DATA   (                      -
                  NAME (SALESMAN.K.D)    -
                  CISZ (4096)            -
                  FSPC (20 10)           -
                  )                      -
           INDEX  (                      -
                  NAME (SALESMAN.K.I)    -
                  )                      -
                  CAT (PRIMER.UCAT1/UCATUPDT)
        DEF PATH  (                      -
                  NAME (SALES.CUST.K)    -
                  FILE (PATHK1)          -
                  PATHENTRY (SALESMAN.K) -
                  UPDATE                 -
                  )                      -
                  CAT (PRIMER.UCAT1/UCATUPDT)
/&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0072 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//AIXK1    DD VOL=SER=WTVSAM,DISP=OLD,UNIT=3330
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
        DEF AIX   (                      -
                  NAME (SALESMAN.K)      -
                  FILE (AIXK1)           -
                  RELATE (CUSTOMER.K)    -
                  KEYS (12 40)           -
                  VOL (WTVSAM)           -
                  CYL (1 1)              -
                  IMBD                   -
                  NONUNIQUEKEY           -
                  UPGRADE                -
                  )                      -
           DATA   (                      -
                  NAME (SALESMAN.K.D)    -
                  CISZ (4096)            -
                  FSPC (20 10)           -
                  )                      -
           INDEX  (                      -
                  NAME (SALESMAN.K.I)    -
                  )                      -
                  CAT (PRIMER.UCAT1/UCATUPDT)
        DEF PATH  (                      -
                  NAME (SALES.CUST.K)    -
                  FILE (AIXK1)           -
                  UPDATE                 -
                  PATHENTRY (SALESMAN.K) -
                  )                      -
                  CAT (PRIMER.UCAT1/UCATUPDT)
/*
```

Output listing is shown on next page

DOS/VS example (output):

```
*****  THIS IS THE OUTPUT OF DEFINE AIX  *****

IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0509I INDEX ALLOC.STAT. FOR VOLUME WTVSAM IS 0
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0


*****  THIS IS THE OUTPUT OF DEFINE PATH  *****

IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
*****  THIS IS THE OUTPUT OF DEFINE AIX  *****

IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0509I INDEX ALLOC.STAT. FOR VOLUME WTVSAM IS 0
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0


*****  THIS IS THE OUTPUT OF DEFINE PATH  *****

IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

LISTCAT output of this object is shown later in section 7.9.2

Notes:

- KEY specifies the length and the offset of the key in the record. That is the length and position (relative to zero) of the alternate key in the base cluster record.
- NONUNIQUEKEY means that a secondary key may point to more than one primary record (synonyms).
- This AIX will be in the UPGRADE set of the base cluster (UPGRADE is default), i.e. it will be automatically updated whenever records in the base cluster are added/deleted or a secondary key field is modified, and if the PATH accessing the base cluster has the UPDATE option (default).
- The DATA CISZ is difficult to determine because of the NONUNIQUEKEY option. But even if specified too small, it would be accepted since the data records of an AIX are in spanned format (SPANNED is default).
- PATHENTRY indicates the name of the AIX this path refers to.
- The DATA RECSZ can be omitted (default is 4086 32600) for average and maximum. The actual size can be computed considering that a data record of any AIX must account for five bytes (header) plus the length of the alternate key plus one (UNIQUEKEY) to 'n' (NONUNIQUEKEY) times the length of the primary key (KSDS base cluster) or times four bytes (RBA length for an ESDS base cluster).
- The FILE parameter is required (not for MVS, also see section 7.2.1 on page 88). The same DD-statement can be used for DEFINE AIX and DEFINE PATH.

The notes are continued on the next page.

DOS/VS notes:

- IJSYSUC is stored in the STDLABEL area.
- The FILE parameters are required to address the CRA of the recoverable user catalog. The same set of DLBL, EXTENT, and ASSGN statements can be used in further jobs to access the data sets since they have the file-ID parameter already coded.
- The 'volid' in the path's EXTENT refers to the first volume of the index component of the KSDS base cluster, or for an ESDS, to the first volume containing the data component of the ESDS base cluster, to allocate the volume with the CRA containing a copy of the catalog entries for this object.

7.6.4.3 BLDINDEX AIX ON KSDS

DOS/VS example (input):

```
// JOB BUILD AIX SALESMAN-CUSTOMER.K ON KSDS
// DLBL KSDS,'CUSTOMER.K',,VSAM
// EXTENT SYS010,WTVSAM
// DLBL AIXK1,'SALESMAN.K',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
        BLDINDEX  INFILE (KSDS)        -
                  OUTFILE (AIXK1)
 /&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0082 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//KSDS     DD DSN=CUSTOMER.K,DISP=OLD
//AIXK1    DD DSN=SALESMAN.K,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
         BLDINDEX  INFILE (KSDS)
                   OUTFILE (AIXK1)
/*
```

DOS/VS example (output):

```
IDC0652I SALESMAN.K  BUILT
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
IDC0652I SALESMAN.K  BUILT
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Notes:

- To create the AIXs, the BLDINDEX function uses a sort which will normally be executed in virtual storage, but if Access Method Services cannot obtain enough virtual storage (or if EXTERNALSORT is specified), BLDINDEX automatically defines two ESDS files in a VSAM space controlled by the same or of a different catalog, uses them as work files for the external sort and deletes them at the end of sort. The default 'dnames' are IDCUT1 and IDCUT2.

DOS/VS notes:

- If external sort is used, two sets of DLBL/EXTENTs (no EXTENT, only volume information needed) must be provided. They can be stored in the label area. The default 'dnames' are IDCUT1 and IDCUT2.

7.6.4.4 PRINT AIX

DOS/VS example (input):

```
// JOB PRINT AIXK1 (PARTIAL)
// DLBL AIXK1,'SALESMAN.K',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
          PRINT  INFILE (AIXK1)       -
                 CHAR                 -
                 COUNT (5)
 /&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0092 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//AIXK1    DD DSN=SALESMAN.K,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
             PRINT INFILE(AIXK1)          -
                   CHAR                   -
                   COUNT (5)
/*
```

Output of Access Method Services (DOS/VS and OS/VS are similar):

```
LISTING OF DS -SALESMAN.K

KEY OF RECORD - ASH
.....ASH            70 1120 1330 1610 1990 2130 3180 3390 3670 4050 4190 5240 5450 5730 6110 6250 7//
10 9360 9570 985010070111201133011610119901213013180133901367014050141901524015450157301611016250 17//
101936019570198502007021120213302161021990221302318023390236702405024190252402545025730261102625027//
10293602957029850
KEY OF RECORD - ASPINALL
.....ASPINALL       300  930 1710 2040 2360 2990 3770 4100 4420 5050 5830 6160 6480 7110 7890 8220 8//
30117101204012360129901377014100144201505015830161601648017110178901822018540191701995020300209302 1//
702410024420250502583026160264802711027890282202854029170299 50
KEY OF RECORD - BAXTER
.....BAXTER         350 1210 1650 1980 2410 3270 3710 4040 4470 5330 5770 6100 6530 7390 7830 8160 8//
10116501198012410132701371014040144701533015770161001653017390178301816018590194501989020350212102 1//
10240402447025330257702610026530273902783028160285902945029890
KEY OF RECORD - BOOTH
.....BOOTH          530  920 1220 1600 1800 2020 2590 2980 3280 3660 3860 4080 4650 5040 5340 5720 5//
00 7780 7980 8200 8770 9160 9460 984010530109201122011600118001202012590129801328013660138601408014//
201614016710171001740017780179801820018770191601946019840205302092021220216002180022020225902298023//
502504025340257202592026140267102710027400277802798028200287702916029460298 40
KEY OF RECORD - BURTON
.....BURTON          40  740 1050 1760 2100 2800 3110 3820 4160 4860 5170 5880 6220 6920 7230 7940 8//
401074011050117601210012800131101382014160148601517015880162201692017230179401828018980192902000020//
00228002311023820241602486025170258802622026920272302794028280289802929030000
IDC0005I NUMBER OF RECORDS PROCESSED WAS 5
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Note: The output lines were truncated on the right side due to limited page size.

7.6.4.5 PRINT PATH

DOS/VS example (input):

```
// JOB PRINT PATHK1 (PARTIAL)
// DLBL PATHK1,'SALES.CUST.K',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
          PRINT  INFILE (PATHK1)      -
                 CHAR                 -
                 COUNT (5)
 /&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0102 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//PATHK1   DD DSN=SALES.CUST.K,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
          PRINT  INFILE (PATHK1)       -
                 CHAR                  -
                 COUNT (5)
/*
```

DOS/VS example (output):

```
LISTING OF DS -SALES.CUST.K
KEY OF RECORD - ASH
   70                    WICHITA        ASH
KEY OF RECORD - ASH
 1120                    ALBANY         ASH
KEY OF RECORD - ASH
 1330                    SEATTLE        ASH
KEY OF RECORD - ASH
 1610                    OAKLAND        ASH
KEY OF RECORD - ASH
 1990                    NEW ORLEANS    ASH
IDC0005I NUMBER OF RECORDS PROCESSED WAS 5
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
LISTING OF DS -SALES.CUST.K
KEY OF RECORD - ASH
   70                    WICHITA        ASH
KEY OF RECORD - ASH
 1120                    ALBANY         ASH
KEY OF RECORD - ASH
 1330                    SEATTLE        ASH
KEY OF RECORD - ASH
 1610                    OAKLAND        ASH
KEY OF RECORD - ASH
 1990                    NEW ORLEANS    ASH
IDC0005I NUMBER OF RECORDS PROCESSED WAS 5
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

7.6.4.6 DEFINE AIX, DEFINE PATH

DOS/VS example (input):

```
// JOB DEF AIX 'CITY' + PATH 'CITY.CUST'
// DLBL KSDS,'CUSTOMER.K',,VSAM
// EXTENT SYS010,WTVSAM
// DLBL AIXK2,'CITY.K',,VSAM
// EXTENT SYS010,WTVSAM
// DLBL PATHK2,'CITY.CUST.K',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
        DEF AIX   (                    -
                  NAME (CITY.K)        -
                  FILE (AIXK2)         -
                  RELATE (CUSTOMER.K)  -
                  KEYS (15 25)         -
                  VOL (WTVSAM)         -
                  CYL (1 1)            -
                  IMBD                 -
                  NONUNIQUEKEY         -
                  UPGRADE              -
                  )                    -
           DATA   (                    -
                  NAME (CITY.K.D)      -
                  CISZ (4096)          -
                  FSPC (20 10)         -
                  )                    -
           INDEX  (                    -
                  NAME (CITY.K.I)      -
                  )                    -
                  CAT (PRIMER.UCAT1/UCATUPDT)
        DEF PATH  (                    -
                  NAME (CITY.CUST.K)   -
                  FILE (PATHK2)        -
                  PATHENTRY (CITY.K)   -
                  UPDATE               -
                  )                    -
                  CAT (PRIMER.UCAT1/UCATUPDT)
/&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0122 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//AIXK2    DD VOL=SER=WTVSAM,DISP=OLD,UNIT=3330
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
        DEF AIX   (                    -
                  NAME (CITY.K)        -
                  FILE (AIXK2)         -
                  RELATE (CUSTOMER.K)  -
                  KEYS (15 25)         -
                  VOL (WTVSAM)         -
                  CYL (1 1)            -
                  IMBD                 -
                  NONUNIQUEKEY         -
                  UPGRADE              -
                  )                    -
           DATA   (                    -
                  NAME (CITY.K.D)      -
                  CISZ (4096)          -
                  FSPC (20 10)         -
                  )                    -
           INDEX  (                    -
                  NAME (CITY.K.I)      -
                  )                    -
                  CAT (PRIMER.UCAT1/UCATUPDT)
        DEF PATH  (                    -
                  NAME (CITY.CUST.K)   -
                  FILE (AIXK2)         -
                  UPDATE               -
                  PATHENTRY (CITY.K)   -
                  )                    -
                  CAT (PRIMER.UCAT1/UCATUPDT)
/*
```

Output listing is shown on next page

DOS/VS example (output):

```
*****  THIS IS THE OUTPUT OF DEFINE AIX  *****

IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0509I INDEX ALLOC.STAT. FOR VOLUME WTVSAM IS 0
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0


*****  THIS IS THE OUTPUT OF DEFINE PATH  *****

IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
*****  THIS IS THE OUTPUT OF DEFINE AIX  *****

IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0509I INDEX ALLOC.STAT. FOR VOLUME WTVSAM IS 0
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0


*****  THIS IS THE OUTPUT OF DEFINE PATH  *****

IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

LISTCAT output of this object is shown later in section 7.9.2

Note: see notes in section 7.6.4.2 on page 111.

7.6.4.7 BLDINDEX AIX ON KSDS

DOS/VS example (input):

```
// JOB BUILD AIX CITY-CUSTOMER.K ON KSDS
// DLBL KSDS,'CUSTOMER.K',,VSAM
// EXTENT SYS010,WTVSAM
// DLBL AIXK2,'CITY.K',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
        BLDINDEX  INFILE (KSDS)          -
                  OUTFILE (AIXK2)
 /&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0132 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//KSDS     DD DSN=CUSTOMER.K,DISP=OLD
//AIXK2    DD DSN=CITY.K,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
         BLDINDEX  INFILE (KSDS)
                   OUTFILE (AIXK2)
/*
```

DOS/VS example (output):

```
IDC0652I CITY.K  BUILT
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
IDC0652I CITY.K  BUILT
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Note: see notes in section 7.6.4.3 on page 113.

7.6.4.8 PRINT PART OF AN AIX

DOS/VS example (input):

```
// JOB PRINT AIXK2 (PARTIAL)
// DLBL AIXK2,'CITY.K',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
          PRINT  INFILE (AIXK2)       -
                 CHAR                 -
                 COUNT (3)
 /&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0142 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//AIXK2    DD DSN=CITY.K,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
           PRINT     INFILE (AIXK2)       -
                     CHAR                 -
                     COUNT (3)
/*
```

Output of Access Method Services (DOS/VS and OS/VS are similar):

```
LISTING OF DS -CITY.K

KEY OF RECORD - ALBANY
.....ALBANY          260  770 1120 2320 2830 3180 4380 4890 5240 6440 6950 7300 8500 9010 93601026//
131801438014890152401644016950173001850019010193602026020770211202232022830231802438024890252402644//
29360
KEY OF RECORD - ATLANTA
.....ATLANTA         520 1840 2580 3900 4640 5960 6700 8020 876010520118401258013900146401596016 70//
225802390024640259602670028020287 60
KEY OF RECORD - AUSTIN
.....AUSTIN          280  620  790 1420 1670 2060 2340 2680 2850 3480 3730 4120 4400 4740 4910 554//
 6970 7600 7850 8240 8520 8860 9030 9660 991010280106201079011420116701206012340126801285013480 1373//
155401579016180164601680016970176001785018240185201886019030196601991020280206202079021420216702206//
237302412024400247402491025540257902618026460268002697027600278502824028520288602903029660299 10
IDC0005I NUMBER OF RECORDS PROCESSED WAS 3
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Note: The output lines were truncated on the right side due to limited page size.

7.6.4.9 PRINT PART OF A BASE CLUSTER VIA PATH

DOS/VS example (input):

```
// JOB PRINT PATHK2 (PARTIAL)
// DLBL PATHK2,'CITY.CUST.K',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
          PRINT  INFILE (PATHK2)      -
                 CHAR                 -
                 COUNT (5)
 /&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0152 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//PATHK2   DD DSN=CITY.CUST.K,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
         PRINT     INFILE (PATHK2)      -
                   CHAR                 -
                   COUNT (5)
/*
```

DOS/VS example (output):

```
LISTING OF DS -CITY.CUST.K
KEY OF RECORD - ALBANY
  260                    ALBANY          SMITH
KEY OF RECORD - ALBANY
  770                    ALBANY          REYNOLDS
KEY OF RECORD - ALBANY
 1120                    ALBANY          ASH
KEY OF RECORD - ALBANY
 2320                    ALBANY          SMITH
KEY OF RECORD - ALBANY
 2830                    ALBANY          REYNOLDS
IDC0005I NUMBER OF RECORDS PROCESSED WAS 5
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
LISTING OF DS -CITY.CUST.K
KEY OF RECORD - ALBANY
  260                    ALBANY          SMITH
KEY OF RECORD - ALBANY
  770                    ALBANY          REYNOLDS
KEY OF RECORD - ALBANY
 1120                    ALBANY          ASH
KEY OF RECORD - ALBANY
 2320                    ALBANY          SMITH
KEY OF RECORD - ALBANY
 2830                    ALBANY          REYNOLDS
IDC0005I NUMBER OF RECORDS PROCESSED WAS 5
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

7.6.4.10 REPRO (COPY DATA OF A KSDS TO A SAM-FILE ON DISK)

DOS/VS example (input):

```
// JOB REPRO KSDS TO SAM-DISK
// DLBL KSDS,'CUSTOMER.K',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// DLBL REPKSDS,'REPRO.KSDS'
// EXTENT SYS000,DOS30Z,1,0,6707,95
// ASSGN SYS000,DISK,VOL=DOS30Z,SHR
// EXEC IDCAMS,SIZE=AUTO
        REPRO     INFILE (KSDS)          -
                  OUTFILE (REPKSDS       -
                  ENV (BLKSZ (4000)      -
                       RECSZ (200)       -
                       RECFM (FB)        -
                       PDEV (3330))      -
                       )
/&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0162 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//KSDS     DD DSN=CUSTOMER.K,DISP=OLD
//REPKSDS  DD VOL=SER=WTVS1R,UNIT=3330,
//            SPACE=(CYL,(5,1)),
//            DSN=REPRO.KSDS,DISP=(NEW,KEEP),
//            DCB=(RECFM=FB,LRECL=200,BLKSIZE=4000)
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
         REPRO     INFILE (KSDS)          -
                   OUTFILE (REPKSDS)
/*
```

DOS/VS example (output):

```
IDC0005I NUMBER OF RECORDS PROCESSED WAS 3000
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
IDC0005I NUMBER OF RECORDS PROCESSED WAS 3000
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

DOS/VS note:

- Access Method Services needs complete specifications of any nonVSAM data set.

7.6.4.11 ALTER (FREE SPACE, PASSWORD)

The three examples given below would probably not be used in a real application. They show three different goals which can be achieved with ALTER.

- Modify the FREESPACE definition in the data component of a cluster after it has been loaded (also see section 8.4 on page 199).
- Protect a cluster with a master password.
- Nullify password protection.
- Since a master password is established by the second ALTER command, the following one has to specify that password to alter the data set attributes.

DOS/VS example (input):

```
// JOB ALTER KSDS
// DLBL KSDS,'CUSTOMER.K',,VSAM
// EXTENT SYS010,WTVSAM
// DLBL KSDSD,'CUSTOMER.K.D',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
         ALTER  CUSTOMER.K.D            -
                FILE (KSDSD)            -
                FREESPACE (20 20)
         ALTER  CUSTOMER.K              -
                FILE (KSDS)             -
                MASTERPW (KSDSMRPW)
         ALTER  CUSTOMER.K/KSDSMRPW     -
                FILE (KSDS)             -
                NULLIFY (MRPW)
/&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIMO172 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//KSDS     DD VOL=SER=WTVSAM,DISP=OLD,UNIT=3330
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
         ALTER  CUSTOMER.K.D            -
                FILE (KSDS)             -
                FREESPACE (20 20)
         ALTER  CUSTOMER.K              -
                FILE (KSDS)             -
                MASTERPW (KSDSMRPW)
         ALTER  CUSTOMER.K/KSDSMRPW     -
                FILE (KSDS)             -
                NULLIFY (MRPW)
/*
```

DOS/VS example (output):

```
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
IDC0531I ENTRY CUSTOMER.K.D ALTERED
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0
IDC0531I ENTRY CUSTOMER.K ALTERED
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0
IDC0531I ENTRY CUSTOMER.K ALTERED
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Note: the FILE parameter is required (not for MVS, also see section 7.2.1 on page 88).

7.6.4.12 EXPORT A KSDS AND ITS AIX'S

When EXPORT PERMANENT is specified, the exported clusters and their subordinate objects will be deleted from the VSAM-catalog after the EXPORT. For this reason, all AIX's must be exported before the base cluster itself.

DOS/VS example (input):

```
// JOB EXPORT AIXK1 + AIXK2 + KSDS
// DLBL KSDS,'CUSTOMER.K',,VSAM
// EXTENT SYS010,WTVSAM
// DLBL AIXK1,'SALESMAN.K',,VSAM
// EXTENT SYS010,WTVSAM
// DLBL AIXK2,'CITY.K',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// TLBL TAPEOUT,'KSDS.AIXK1.AIXK2'
// ASSGN SYS005,TAPE,VOL=057454
// MTC REW,SYS005
// EXEC IDCAMS,SIZE=AUTO
        EXPORT    SALESMAN.K                -
                  INFILE (AIXK1)            -
                  OUTFILE (TAPEOUT          -
                  ENV (BLKSZ (4096)         -
                       PDEV (2400))         -
                       )                    -
                  PERMANENT
        EXPORT    CITY.K                    -
                  INFILE (AIXK2)            -
                  OUTFILE (TAPEOUT          -
                  ENV (BLKSZ (4096)         -
                       PDEV (2400))         -
                       )                    -
                  PERMANENT
        EXPORT    CUSTOMER.K                -
                  INFILE (KSDS)             -
                  OUTFILE (TAPEOUT          -
                  ENV (BLKSZ (4096)         -
                       PDEV (2400))         -
                       )                    -
                  PERMANENT
/&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0182 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//AIXK1    DD DSN=SALESMAN.K,DISP=OLD
//AIXK2    DD DSN=CITY.K,DISP=OLD
//KSDS     DD DSN=CUSTOMER.K,DISP=OLD
//EXPAXK1  DD VOL=(,RETAIN,,,SER=057454),
//         UNIT=3400-6,LABEL=(2,SL),
//         DSN=EXPORTED.AIXK1,DISP=(NEW,PASS)
//EXPAXK2  DD VOL=(,RETAIN,,,SER=057454),
//         UNIT=3400-6,LABEL=(3,SL),
//         DSN=EXPORTED.AIXK2,DISP=(NEW,PASS)
//EXPKSDS  DD VOL=(,RETAIN,,,SER=057454),
//         UNIT=3400-6,LABEL=(4,SL),
//         DSN=EXPORTED.KSDS,DISP=(NEW,PASS)
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
           EXPORT    SALESMAN.K                -
                     INFILE (AIXK1)            -
                     OUTFILE (EXPAXK1)         -
                     PERMANENT
           EXPORT    CITY.K                    -
                     INFILE (AIXK2)            -
                     OUTFILE (EXPAXK2)         -
                     PERMANENT
           EXPORT    CUSTOMER.K                -
                     INFILE (KSDS)             -
                     OUTFILE (EXPKSDS)         -
                     PERMANENT
/*
```

Output listing is shown on next page

DOS/VS example (output):

```
**** THIS IS THE OUTPUT OF THE FIRST EXPORT ****

IDC0594I PORT. DS CREATED ON 10/20/77 AT 15:57:18
IDC0550I ENTRY (R) SALES.CUST.K DELETED
IDC0550I ENTRY (G) SALESMAN.K DELETED
IDC0550I ENTRY (D) SALESMAN.K.D DELETED
IDC0550I ENTRY (I) SALESMAN.K.I DELETED
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

**** THIS IS THE OUTPUT OF THE SECOND EXPORT ****

IDC0594I PORT. DS CREATED ON 10/20/77 AT 15:57:28
IDC0550I ENTRY (R) CITY.CUST.K DELETED
IDC0550I ENTRY (G) CITY.K DELETED
IDC0550I ENTRY (D) CITY.K.D DELETED
IDC0550I ENTRY (I) CITY.K.I DELETED
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

**** THIS IS THE OUTPUT OF THE THIRD EXPORT ****

IDC0594I PORT. DS CREATED ON 10/20/77 AT 15:57:37
IDC0550I ENTRY (C) CUSTOMER.K DELETED
IDC0550I ENTRY (D) CUSTOMER.K.D DELETED
IDC0550I ENTRY (I) CUSTOMER.K.I DELETED
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
**** THIS IS THE OUTPUT OF THE FIRST EXPORT ****

IDC0594I PORT. DS CREATED ON 12/21/77 AT 21:32:52
IDC0550I ENTRY (R) SALES.CUST.K DELETED
IDC0550I ENTRY (D) SALESMAN.K.D DELETED
IDC0550I ENTRY (I) SALESMAN.K.I DELETED
IDC0550I ENTRY (G) SALESMAN.K DELETED
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

**** THIS IS THE OUTPUT OF THE SECOND EXPORT ****

IDC0594I PORT. DS CREATED ON 12/21/77 AT 21:32:57
IDC0550I ENTRY (R) CITY.CUST.K DELETED
IDC0550I ENTRY (D) CITY.K.D DELETED
IDC0550I ENTRY (I) CITY.K.I DELETED
IDC0550I ENTRY (G) CITY.K DELETED
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

**** THIS IS THE OUTPUT OF THE THIRD EXPORT ****

IDC0594I PORT. DS CREATED ON 12/21/77 AT 21:33:03
IDC0550I ENTRY (D) CUSTOMER.K.D DELETED
IDC0550I ENTRY (I) CUSTOMER.K.I DELETED
IDC0550I ENTRY (C) CUSTOMER.K DELETED
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

DOS/VS notes:

- Since standard tape label processing is used (no unlabeled tapes allowed prior to DOS/VS Rel. 34), the output tape contains these files :

  1. VOL + HDR-label
  2. AIXK1 data
  3. EOF label
  4. HDR label
  5. AIXK2 data
  6. EOF label
  7. HDR label
  8. KSDS data
  9. EOF label

- At EOJ the tape will not be rewound unless REWIND/UNLOAD is specified in the ENV-parameter or a MTC REW/RUN is used at end of step.

7.6.4.13 IMPORT A KSDS AND ITS AIX'S

DOS/VS example (input):

```
// JOB IMPORT KSDS + AIXK1 + AIXK2
// DLBL KSDS,'CUSTOMER.K',,VSAM
// EXTENT SYS010,WTVSAM
// DLBL AIXK1,'SALESMAN.K',,VSAM
// EXTENT SYS010,WTVSAM
// DLBL AIXK2,'CITY.K',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// TLBL TAPEIN,'KSDS.AIXK1.AIXK2'
// ASSGN SYS004,TAPE,VOL=057454
// MTC REW,SYS004
// MTC FSF,SYS004,6
// EXEC IDCAMS,SIZE=AUTO
         IMPORT  INFILE (TAPEIN          -
                    ENV (BLKSZ (4096)    -
                         PDEV (2400))    -
                         )               -
                 OUTFILE (KSDS)          -
                 CAT (PRIMER.UCAT1/UCATUPDT)
/&
// MTC REW,SYS004
// EXEC IDCAMS,SIZE=AUTO
         IMPORT  INFILE (TAPEIN          -
                    ENV (BLKSZ (4096)    -
                         PDEV (2400))    -
                         )               -
                 OUTFILE (AIXK1)         -
                 CAT (PRIMER.UCAT1/UCATUPDT)
         IMPORT  INFILE (TAPEIN          -
                    ENV (BLKSZ (4096)    -
                         PDEV (2400))    -
                         )               -
                 OUTFILE (AIXK2)         -
                 CAT (PRIMER.UCAT1/UCATUPDT)
/&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0192 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//AIXK1    DD DSN=SALESMAN.K,AMP='AMORG',
//         VOL=SER=WTVSAM,DISP=OLD,UNIT=3330
//AIXK2    DD DSN=CITY.K,AMP='AMORG',
//         VOL=SER=WTVSAM,DISP=OLD,UNIT=3330
//KSDS     DD DSN=CUSTOMER.K,AMP='AMORG',
//         VOL=SER=WTVSAM,DISP=OLD,UNIT=3330
//EXPAXK1  DD VOL=(,RETAIN,,,SER=057454),
//         UNIT=3400-6,LABEL=(2,SL),
//         DSN=EXPORTED.AIXK1,DISP=(OLD,PASS)
//EXPAXK2  DD VOL=(,RETAIN,,,SER=057454),
//         UNIT=3400-6,LABEL=(3,SL),
//         DSN=EXPORTED.AIXK2,DISP=(OLD,PASS)
//EXPKSDS  DD VOL=(,RETAIN,,,SER=057454),
//         UNIT=3400-6,LABEL=(4,SL),
//         DSN=EXPORTED.KSDS,DISP=(OLD,PASS)
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
         IMPORT  INFILE (EXPKSDS)        -
                 OUTFILE (KSDS)          -
                 CAT (PRIMER.UCAT1/UCATUPDT)
         IMPORT  INFILE (EXPAXK1)        -
                 OUTFILE (AIXK1)         -
                 CAT (PRIMER.UCAT1/UCATUPDT)
         IMPORT  INFILE (EXPAXK2)        -
                 OUTFILE (AIXK2)         -
                 CAT (PRIMER.UCAT1/UCATUPDT)
/*
```

Output listing is shown on next page

DOS/VS example (output):

```
***  THIS IS THE OUTPUT OF THE SECOND IMPORT  ***

IDC0604I DS BEING IMP. WAS EXP. 10/20/77 AT 15:57
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0509I INDEX ALLOC.STAT. FOR VOLUME WTVSAM IS 0
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

***  THIS IS THE OUTPUT OF THE SECOND IMPORT  ***

IDC0604I DS BEING IMP. WAS EXP. 10/20/77 AT 15:57
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0509I INDEX ALLOC.STAT. FOR VOLUME WTVSAM IS 0
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

***  THIS IS THE OUTPUT OF THE THIRD IMPORT  ***

IDC0604I DS BEING IMP. WAS EXP. 10/20/77 AT 15:57
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0509I INDEX ALLOC.STAT. FOR VOLUME WTVSAM IS 0
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
****  THIS IS THE OUTPUT OF THE FIRST IMPORT  ****

IDC0604I DS BEING IMP. WAS EXP. 12/21/77 AT 21:33
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0509I INDEX ALLOC.STAT. FOR VOLUME WTVSAM IS 0
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

****  THIS IS THE OUTPUT OF THE SECOND IMPORT  ****

IDC0604I DS BEING IMP. WAS EXP. 12/21/77 AT 21:32
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0509I INDEX ALLOC.STAT. FOR VOLUME WTVSAM IS 0
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

****  THIS IS THE OUTPUT OF THE THIRD IMPORT  ****

IDC0604I DS BEING IMP. WAS EXP. 12/21/77 AT 21:32
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0509I INDEX ALLOC.STAT. FOR VOLUME WTVSAM IS 0
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Notes:

- The PURGE parameter is not required because no expiration date was specified in the DEFINE command.
- Read also section A.6.2 on page 258 where IMPORT considerations are described.
- Since an AIX cannot be built or imported before its base cluster is loaded or imported, the jobstream has to start with the KSDS IMPORT.

DOS/VS notes:

- To import the KSDS, the input tape must be precisely positioned at the beginning of the KSDS HDR1-label with a MTC FSF command (see DOS/VS notes in the previous section for the tape layout).
- After the KSDS has been imported, the tape must be rewound to read the AIX's data. This can only be done by a MTC REW command between the two jobsteps. The REWIND option of the ENV parameter (available for DOS/VS Rel.34 and up) cannot be used since it would rewind the tape at both OPEN and CLOSE times and lose the positioning.

7.6.4.14 VERIFY KSDS

DOS/VS example (input):

```
// JOB VERIFY KSDS
// DLBL KSDS,'CUSTOMER.K',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
        VERIFY   FILE (KSDS)
 /&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0202 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//KSDS     DD DSN=CUSTOMER.K,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
         VERIFY   FILE (KSDS)
/*
```

DOS/VS example (output):

```
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Notes:

- The condition code '0' shows that the VERIFY function worked properly.

  If the data set was in need of VERIFYing, messages IDC3300I and IDC3351I will be issued.

- The FILE parameter is required to identify the DD statement describing the data set to be verified.

7.7 ESDS (ENTRY SEQUENCED DATA SET)

7.7.1 DEFINE ESDS

DOS/VS example (input):

```
// JOB DEFINE ESDS IN VSAM SPACE
// DLBL ESDS,'CUSTOMER.E',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
        DEF CLUSTER                    -
                (                      -
                NAME (CUSTOMER.E)      -
                FILE (ESDS)            -
                VOL (WTVSAM)           -
                CYL (5 2)              -
                NONINDEXED             -
                )                      -
            DATA (                     -
                NAME (CUSTOMER.E.D)    -
                RECORDSIZE (200 200)   -
                CISZ (4096)            -
                )                      -
            CAT (PRIMER.UCAT1/UCATMRPW)
/&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0212 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//ESDS  DD VOL=SER=WTVSAM,DISP=OLD,UNIT=3330
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
         DEF CLUSTER                   -
                 (                     -
                 NAME (CUSTOMER.E)     -
                 FILE (ESDS)           -
                 VOL (WTVSAM)          -
                 CYL (5 2)             -
                 NONINDEXED            -
                 )                     -
             DATA (                    -
                 NAME (CUSTOMER.E.D)   -
                 RECORDSIZE (200 200)  -
                 CISZ (4096)           -
                 )                     -
                 CAT (PRIMER.UCAT1/UCATUPDT)
/*
```

DOS/VS example (output):

```
IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Notes:

- NONINDEXED is the parameter to specify a DEFINE ESDS.
- See also section 7.6.1 on page 106.

DOS/VS notes:

- IJSYSUC is stored in the STDLABEL area.
- The VSAM data set name in the DLBL statement is optional for DEFINE.

7.7.2 REPRO (LOAD AN ESDS)

DOS/VS example (input):

```
// JOB LOAD ESDS FROM DISK-SAM
// DLBL SQDSK,'LARGE.ESDS.DATA'
// EXTENT SYS000,,,,6802,95
// ASSGN SYS000,DISK,VOL=DOS30Z,SHR
// DLBL ESDS,'CUSTOMER.E',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
        REPRO                          -
              INFILE (SQDSK            -
                 ENV (RECFM (FB)       -
                 BLKSZ (1600)          -
                 RECSZ (200)           -
                 PDEV (3330))          -
                 )                     -
              OUTFILE (ESDS)
/&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIMO222 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//SQDSK    DD VOL=SER=WTVS1R,UNIT=3330,
//         DSN=LOADESDS,DISP=(OLD,KEEP),
//         DCB=(RECFM=FB,LRECL=200,BLKSIZE=1600)
//ESDS     DD DSN=CUSTOMER.E,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
         REPRO                         -
               INFILE (SQDSK)          -
               OUTFILE (ESDS)
/*
```

DOS/VS example (output):

```
IDC0005I NUMBER OF RECORDS PROCESSED WAS 3000
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
IDC0005I NUMBER OF RECORDS PROCESSED WAS 3000
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

DOS/VS notes:

- Access Method Services needs complete specifications of any nonVSAM data set.

7.7.2.1 DEFINE AIX, DEFINE PATH

DOS/VS example (input):

```
// JOB DEF AIX 'SALESMAN' + PATH 'SALES.CUST'
// DLBL ESDS,'CUSTOMER.E',,VSAM
// EXTENT SYS010,WTVSAM
// DLBL AIXE1,'SALESMAN.E',,VSAM
// EXTENT SYS010,WTVSAM
// DLBL PATHE1,'SALES.CUST.E',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
        DEF AIX   (                      -
                  NAME (SALESMAN.E)      -
                  FILE (AIXE1)           -
                  RELATE (CUSTOMER.E)    -
                  KEYS (12 40)           -
                  VOL (WTVSAM)           -
                  CYL (1 1)              -
                  IMBD                   -
                  NONUNIQUEKEY           -
                  UPGRADE                -
                  )                      -
              DATA (                     -
                  NAME (SALESMAN.E.D)    -
                  CISZ (4096)            -
                  FSPC (20 10)           -
                  )                      -
              INDEX (                    -
                  NAME (SALESMAN.E.I)    -
                  )                      -
                  CAT (PRIMER.UCAT1/UCATMRPW)
        DEF PATH  (                      -
                  NAME (SALES.CUST.E)    -
                  FILE (PATHE1)          -
                  PATHENTRY (SALESMAN.E) -
                  UPDATE                 -
                  )                      -
                  CAT (PRIMER.UCAT1/UCATMRPW)
/&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIMO232 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//AIXE1    DD VOL=SER=WTVSAM,DISP=OLD,UNIT=3330
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
         DEF AIX   (                     -
                   NAME (SALESMAN.E)     -
                   FILE (AIXE1)          -
                   RELATE (CUSTOMER.E)   -
                   KEYS (12 40)          -
                   VOL (WTVSAM)          -
                   CYL (1 1)             -
                   IMBD                  -
                   NONUNIQUEKEY          -
                   UPGRADE               -
                   )                     -
               DATA (                    -
                   NAME (SALESMAN.E.D)   -
                   CISZ (4096)           -
                   FSPC (20 10)          -
                   )                     -
               INDEX (                   -
                   NAME (SALESMAN.E.I)   -
                   )                     -
                   CATALOG (PRIMER.UCAT1/UCATMRPW)
         DEF PATH  (                     -
                   NAME (SALES.CUST.E)   -
                   FILE (AIXE1)          -
                   UPDATE                -
                   PATHENTRY (SALESMAN.E) -
                   )                     -
                   CATALOG (PRIMER.UCAT1/UCATMRPW)
/*
```

Output listing is shown on next page

DOS/VS example (output):

```
*****  THIS IS THE OUTPUT OF DEFINE AIX  *****

IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0509I INDEX ALLOC.STAT. FOR VOLUME WTVSAM IS 0
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0


*****  THIS IS THE OUTPUT OF DEFINE PATH  *****

IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
*****  THIS IS THE OUTPUT OF DEFINE AIX  *****

IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0509I INDEX ALLOC.STAT. FOR VOLUME WTVSAM IS 0
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0


*****  THIS IS THE OUTPUT OF DEFINE PATH  *****

IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

LISTCAT output of this object is shown later in section 7.9.2

Note: see notes in section 7.6.4.2 on page 111.

7.7.2.2 BLDINDEX AIX ON ESDS

DOS/VS example (input):

```
// JOB BUILD AIX SALESMAN-CUSTOMER.E ON ESDS
// DLBL ESDS,'CUSTOMER.E',,VSAM
// EXTENT SYS010,WTVSAM
// DLBL AIXE1,'SALESMAN.E',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
      BLDINDEX  INFILE (ESDS)      -
                OUTFILE (AIXE1)
 /&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0242 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//ESDS     DD DSN=CUSTOMER.E,DISP=OLD
//AIXE1    DD DSN=SALESMAN.E,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
       BLDINDEX  INFILE (ESDS)          -
                 OUTFILE (AIXE1)
/*
```

DOS/VS example (output):

```
IDC0652I SALESMAN.E  BUILT
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
IDC0652I SALESMAN.E  BUILT
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Note: see notes in section 7.6.4.3 on page 113.

7.7.2.3 PRINT ESDS

DOS/VS example (input):

```
// JOB PRINT ESDS (PARTIAL)
// DLBL ESDS,'CUSTOMER.E',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
         PRINT  INFILE (ESDS)        -
                CHAR                 -
                COUNT (5)
 /&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0572 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//ESDS     DD DSN=CUSTOMER.E,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
         PRINT  INFILE (ESDS)          -
                CHAR                   -
                COUNT (5)
/*
```

DOS/VS example (output):

```
LISTING OF DS -CUSTOMER.E
RBA OF RECORD - 0
 6000                    MIAMI          FLEMING
RBA OF RECORD - 200
   20                    NEW YORK       ROBERTS
RBA OF RECORD - 400
16000                    MIAMI          FLEMING
RBA OF RECORD - 600
   50                    RENO           SMITH
RBA OF RECORD - 800
   60                    HOUSTON        JACKSON
IDC0005I NUMBER OF RECORDS PROCESSED WAS 5
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
LISTING OF DS -CUSTOMER.E
RBA OF RECORD - 0
 6000                    MIAMI          FLEMING
RBA OF RECORD - 200
   20                    NEW YORK       ROBERTS
RBA OF RECORD - 400
16000                    MIAMI          FLEMING
RBA OF RECORD - 600
   50                    RENO           SMITH
RBA OF RECORD - 800
   60                    HOUSTON        JACKSON
IDC0005I NUMBER OF RECORDS PROCESSED WAS 5
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Note:

- Since data loaded into the ESDS base cluster are not the same used in examples of section 7.6 (on page 106) for the KSDS base cluster, the listings will differ as well.

7.7.2.4 PRINT AIX

DOS/VS example (input):

```
// JOB PRINT AIXE1 (PARTIAL)
// DLBL AIXE1,'SALESMAN.E',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
         PRINT  INFILE (AIXE1)       -
                DUMP                 -
                COUNT (3)
 /&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0252 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//AIXE1    DD DSN=SALESMAN.E,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
       PRINT INFILE (ESDS)              -
             DUMP                       -
             COUNT (3)
/*
```

Output of Access Method Services (DOS/VS and OS/VS are similar):

```
LISTING OF DS -SALESMAN.E
KEY OF RECORD - C1E2C8404040404040404040
0000  00040048 0CC1E2C8 40404040 40404040 40000004 B000008E D8000090 C8000092 *.....ASH         ...//
0020  580000AC 800000AD 4800011B B800011D 4800011E D8000122 58000123 20000123 *.....................//
0040  E800013A 2800013A F000013B B80001AB B80001AD 480001AE D80001B2 580001B3 *Y.......0.............//
0060  200001B3 E80001CA 280001CA F00001CB B8000242 58000243 20000243 E800025A *....Y.......0.......//
0080  2800025A F000025B B80002D2 580002D3 200002D3 E80002EA 280002EA F00002EB *....0..$...K...L...L//
00A0  B8000317 D0000318 98000319 60000362 58000363 20000363 E80003A7 D00003A8 *.............-......//
00C0  980003A9 60000437 D0000438 98000439 600004C7 D00004C8 980004C9 60000557 *....-...........-..G//
00E0  D0000558 98000559 600005B7 D00005B8 980005B9 60000647 D0000648 98000649 *........-...........//
0100  600006D7 D00006D8 980006D9 60000767 D0000768 98000769 600007F7 D00007F8 *-..P...Q...R-........//
0120  980007F9 6000095B B800095D 4800095E D8                                  *...9-..$...)...;Q
KEY OF RECORD - C1E2D7C9D5C1D3D340404040
0000  00040039 0CC1E2D7 C9D5C1D3 D3404040 40000075 78000077 D000007A 28000080 *.....ASPINALL  ...../// 
0020  C8000082 58000083 E8000100 00000103 20000106 4000010C 8000010D 4800010E *H.......Y...........//
0040  10000190 00000193 20000196 4000019C 8000019D 4800019E 10000222 58000224 *.............  .......//
0060  B0000227 080002D9 600002DA 280002DA F000036E 1000036E D8000370 000003F9 *.......R-........0..>//
0080  600003FA 280003FA F000048E 1000048E D8000490 0000051E 1000051E D8000520 *-.......0.......Q...//
00A0  000006BA F00006BC 800006BE 1000074A F000074C 8000074E 100007DD 480007DE *....0...........0..<//
00C0  100007DE D800086D 4800086E 1000086E D80008BC 800008BE 100008C0 000008FD *....Q.._...>...>Q...//
00E0  480008FE 100008FE D800094A 2800094B B800094D 48                          *........Q..........(//
KEY OF RECORD - C2C1E7E3C5D9404040404040
0000  00040039 0CC2C1E7 E3C5D940 40404040 40000076 40000078 9800007A F0000100 *.....BAXTER  ... ...//
0020  C8000103 E8000107 08000190 C8000193 E8000197 080001FA 280001FB B80001FD *H...Y.......H...Y...//
0040  48000287 D0000289 6000028A F000031A F000031D 48000320 0000035A F000035B *........-...0...0...//
0060  B800035C 800003AA F00003AC 800003AE 100003E8 980003E9 600003EA 2800043A *...*....0.........Y//
0080  F000043C 8000043E 10000478 98000479 6000047A 28000506 40000507 08000507 *0.............-..:// 
00A0  D0000598 98000599 6000059A 2800061E D8000620 00000620 C80006AC 800006AD *........-.......Q...//
00C0  480006AE 1000073C 8000073D 4800073E 100007CC 800007CD 480007CE 1000085E *.....................//
00E0  D8000860 00000860 C8000945 78000946 40000947 08                          *Q..-...-H....... ...//
IDC0005I NUMBER OF RECORDS PROCESSED WAS 3
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Note: The output lines were truncated on the right side due to limited page size.

7.7.2.5 PRINT PATH

DOS/VS example (input):

```
// JOB PRINT PATHE1 (PARTIAL)
// DLBL PATHE1,'SALES.CUST.E',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
         PRINT  INFILE (PATHE1)      -
                CHAR                 -
                COUNT (5)
 /&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0262 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//PATHE1   DD DSN=SALES.CUST.E,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
          PRINT  INFILE (PATHE1)         -
                 CHAR                    -
                 COUNT (5)
/*
```

DOS/VS example (output):

```
LISTING OF DS -SALES.CUST.E
KEY OF RECORD - ASH
   70                    WICHITA        ASH
KEY OF RECORD - ASH
 4050                    NEW ORLEANS    ASH
KEY OF RECORD - ASH
14050                    NEW ORLEANS    ASH
KEY OF RECORD - ASH
24050                    NEW ORLEANS    ASH
KEY OF RECORD - ASH
10070                    WICHITA        ASH
IDC0005I NUMBER OF RECORDS PROCESSED WAS 5
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
LISTING OF DS -SALES.CUST.E
KEY OF RECORD - ASH
   70                    WICHITA        ASH
KEY OF RECORD - ASH
 4050                    NEW ORLEANS    ASH
KEY OF RECORD - ASH
14050                    NEW ORLEANS    ASH
KEY OF RECORD - ASH
24050                    NEW ORLEANS    ASH
KEY OF RECORD - ASH
10070                    WICHITA        ASH
IDC0005I NUMBER OF RECORDS PROCESSED WAS 5
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

7.7.2.6 DEFINE AIX, DEFINE PATH

DOS/VS example (input):

```
// JOB DEF AIX 'CITY' + PATH 'CITY-CUST'
// DLBL ESDS,'CUSTOMER.E',,VSAM
// EXTENT SYS010,WTVSAM
// DLBL AIXE2,'CITY.E',,VSAM
// EXTENT SYS010,WTVSAM
// DLBL PATHE2,'CITY.CUST.E',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
        DEF AIX   (                      -
                  NAME (CITY.E)          -
                  FILE (AIXE2)           -
                  RELATE (CUSTOMER.E)    -
                  KEYS (15 25)           -
                  VOL (WTVSAM)           -
                  CYL (1 1)              -
                  IMBD                   -
                  NONUNIQUEKEY           -
                  UPGRADE                -
                  )                      -
           DATA   (                      -
                  NAME (CITY.E.D)        -
                  CISZ (4096)            -
                  FSPC (20 10)           -
                  )                      -
           INDEX  (                      -
                  NAME (CITY.E.I)        -
                  )                      -
                  CAT (PRIMER.UCAT1/UCATMRPW)
        DEF PATH  (                      -
                  NAME (CITY.CUST.E)     -
                  FILE (PATHE2)          -
                  PATHENTRY (CITY.E)     -
                  UPDATE                 -
                  )                      -
                  CAT (PRIMER.UCAT1/UCATMRPW)
/&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0272 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//AIXE2    DD VOL=SER=WTVSAM,DISP=OLD,UNIT=3330
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
         DEF AIX   (                     -
                   NAME (CITY.E)         -
                   FILE (AIXE2)          -
                   RELATE (CUSTOMER.E)   -
                   KEYS (15 25)          -
                   VOL (WTVSAM)          -
                   CYL (1 1)             -
                   IMBD                  -
                   NONUNIQUEKEY          -
                   UPGRADE               -
                   )                     -
              DATA (                     -
                   NAME (CITY.E.D)       -
                   CISZ (4096)           -
                   FSPC (20 10)          -
                   )                     -
              INDEX (                    -
                   NAME (CITY.E.I)       -
                   )                     -
                   CAT (PRIMER.UCAT1/UCATMRPW)
         DEF PATH  (                     -
                   NAME (CITY.CUST.E)    -
                   FILE (AIXE2)          -
                   UPDATE                -
                   PATHENTRY (CITY.E)    -
                   )                     -
                   CAT (PRIMER.UCAT1/UCATMRPW)
/*
```

Output listing is shown on next page

DOS/VS example (output):

```
*****  THIS IS THE OUTPUT OF DEFINE AIX  *****

IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0509I INDEX ALLOC.STAT. FOR VOLUME WTVSAM IS 0
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0


*****  THIS IS THE OUTPUT OF DEFINE PATH  *****

IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
*****  THIS IS THE OUTPUT OF DEFINE AIX  *****

IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0509I INDEX ALLOC.STAT. FOR VOLUME WTVSAM IS 0
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0


*****  THIS IS THE OUTPUT OF DEFINE PATH  *****

IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

LISTCAT output of this object is shown later in section 7.9.2

Note: see notes in section 7.6.4.2 on page 111.

7.7.2.7 BLDINDEX AIX ON ESDS

DOS/VS example (input):

```
// JOB BUILD AIX CITY-CUSTOMER.E ON ESDS
// DLBL ESDS,'CUSTOMER.E',,VSAM
// EXTENT SYS010,WTVSAM
// DLBL AIXE2,'CITY.E',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
     BLDINDEX  INFILE (ESDS)          -
               OUTFILE (AIXE2)
 /&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIMO282 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//ESDS     DD DSN=CUSTOMER.E,DISP=OLD
//AIXE2    DD DSN=CITY.E,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
         BLDINDEX  INFILE (ESDS)
                   OUTFILE (AIXE2)
/*
```

DOS/VS example (output):

```
IDC0652I CITY.E  BUILT
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
IDC0652I CITY.E  BUILT
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Note: see notes in section 7.6.4.3 on page 113.

7.7.2.8 PRINT AIX

DOS/VS example (input):

```
// JOB PRINT AIXE2 (PARTIAL)
// DLBL AIXE2,'CITY.E',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
          PRINT  INFILE (AIXE2)       -
                 DUMP                 -
                 COUNT (3)
 /&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0292 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//AIXE2    DD DSN=CITY.E,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
         PRINT INFILE (AIXE2)          -
               DUMP                    -
               COUNT (3)
/*
```

Output of Access Method Services (DOS/VS and OS/VS are similar):

```
LISTING OF DS -CITY.E
KEY OF RECORD - C1D3C2C1D5E84040404040404040
0000  0004002D 0FC1D3C2 C1D5E840 40404040 40404040 000020C8 00002190 00002258 *.....ALBANY   ...H..//
0020  00012258 00012320 000123E8 0001B258 0001B320 0001B3E8 00024258 00024320 *...........Y........//
0040  000243E8 00027258 00027320 000273E8 0002D258 0002D320 0002D3E8 00030258 *...Y...........Y..K.//
0060  00030320 000303E8 00036258 00036320 000363E8 00039258 00039320 000393E8 *.......Y...........Y//
0080  000424B0 00042578 00042640 0004B4B0 0004B578 0004B640 000737D0 00073898 *.......... ........//
00A0  00073960 0007C7D0 0007C898 0007C960 00085A28 00085AF0 00085BB8 0008EA28 *...-..G...H...I-....//
00C0  0008EAF0 0008EBB8                                                       *...0....
KEY OF RECORD - C1E3D3C1D5E3C14040404040404040
0000  0004001B 0FC1E3D3 C1D5E3C1 40404040 40404040 00004000 000040C8 00004190 *.....ATLANTA    .. .//
0020  0004E4B0 0004E578 0004E640 00057258 00057320 000573E8 000604B0 00060578 *..U...V...W ........//
0040  00060640 00069708 000697D0 00069898 000724B0 00072578 00072640 0007E708 *... ................//
0060  0007E7D0 0007E898 00087960 00087A28 00087AF0 00090960 00090A28 00090AF0 *..X...Y....-..:...:0//
KEY OF RECORD - C1E4E2E3C9D5404040404040404040
0000  00040057 0FC1E4E2 E3C9D540 40404040 40404040 00005A28 00005C80 00005ED8 *.....AUSTIN   ......//
0020  0000A3E8 0000A4B0 0000A578 000130C8 00013190 00013258 0001C320 0001C3E8 *...Y...........H....//
0040  0001C4B0 000250C8 00025190 00025258 0002B0C8 0002B190 0002B258 00034320 *..D...&H...........H//
0060  000343E8 000344B0 0003D578 0003D640 0003D708 00040320 000403E8 000404B0 *...Y......N...O ..P.//
0080  00046320 000463E8 000464B0 00049320 000493E8 000494B0 0004F0C8 0004F190 *.......Y...........Y//
00A0  0004F258 000523E8 00052578 00052708 0005AED8 0005B0C8 0005B258 0005DBB8 *..2....Y...........Q//
00C0  0005DC80 0005DD48 00063ED8 000640C8 00064258 00065898 00065960 00065A28 *...........Q.. H....//
00E0  00066BB8 00066C80 00066D48 0006EAF0 0006EBB8 0006EC80 0006FBB8 0006FC80 *..,...%..._....0....//
0100  0006FD48 00077640 00077708 000777D0 00077898 00077960 00077A28 00078C80 *....... ............//
0120  00078E10 00079000 00080640 00080708 000807D0 00080898 00080960 00080A28 *.......... ........//
0140  00081C80 00081E10 00082000 000893E8 000894B0 00089578 00089640 00089708 *...............Y....//
0160  000897D0 00092320 00092578 000927D0                                     *................
IDC0005I NUMBER OF RECORDS PROCESSED WAS 3
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Note: The output lines were truncated on the right side due to limited page size.

7.7.2.9 PRINT PATH

DOS/VS example (input):

```
// JOB PRINT PATHE2 (PARTIAL)
// DLBL PATHE2,'CITY.CUST.E',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
         PRINT  INFILE (PATHE2)      -
                CHAR                 -
                COUNT (5)
 /&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0302 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//PATHE2   DD DSN=CITY.CUST.E,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
        PRINT     INFILE (PATHE2)       -
                  CHAR                  -
                  COUNT (5)
/*
```

DOS/VS example (output):

```
LISTING OF DS -CITY.CUST.E
KEY OF RECORD - ALBANY
 9010                    ALBANY         REYNOLDS
KEY OF RECORD - ALBANY
19010                    ALBANY         REYNOLDS
KEY OF RECORD - ALBANY
29010                    ALBANY         REYNOLDS
KEY OF RECORD - ALBANY
 1120                    ALBANY         ASH
KEY OF RECORD - ALBANY
11120                    ALBANY         ASH
IDC0005I NUMBER OF RECORDS PROCESSED WAS 5
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
LISTING OF DS -CITY.CUST.E
KEY OF RECORD - ALBANY
 9010                    ALBANY         REYNOLDS
KEY OF RECORD - ALBANY
19010                    ALBANY         REYNOLDS
KEY OF RECORD - ALBANY
29010                    ALBANY         REYNOLDS
KEY OF RECORD - ALBANY
 1120                    ALBANY         ASH
KEY OF RECORD - ALBANY
11120                    ALBANY         ASH
IDC0005I NUMBER OF RECORDS PROCESSED WAS 5
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

7.8 RRDS (RELATIVE RECORD DATA SET)

7.8.1 DEFINE RRDS

DOS/VS example (input):

```
// JOB DEFINE RRDS IN VSAM SPACE
// DLBL RRDS,'CUSTOMER.R',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
        DEF CLUSTER                     -
                (                       -
                NAME (CUSTOMER.R)       -
                FILE (RRDS)             -
                VOL (WTVSAM)            -
                CYL (5 2)               -
                NUMBERED                -
                )                       -
           DATA (                       -
                NAME (CUSTOMER.R.D)     -
                RECORDSIZE (80 80)      -
                CISZ (4096)             -
                )                       -
           CAT (PRIMER.UCAT1/UCATMRPW)
/&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0312 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//RRDS  DD VOL=SER=WTVSAM,DISP=OLD,UNIT=3330
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
         DEF CLUSTER                    -
                (                       -
                NAME (CUSTOMER.R)       -
                FILE (RRDS)             -
                VOL (WTVSAM)            -
                CYL (5 2)               -
                NUMBERED                -
                )                       -
             DATA (                     -
                NAME (CUSTOMER.R.D)     -
                RECORDSIZE (80 80)      -
                CISZ (4096)             -
                )                       -
                CAT (PRIMER.UCAT1/UCATUPDT)
/*
```

DOS/VS example (output):

```
IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Notes:

- NUMBERED is the parameter to specify to define an RRDS.
- See also notes in section 7.6.1 on page 106.

7.8.2 REPRO (LOAD AN RRDS)

DOS/VS example (input):

```
// JOB LOAD RRDS FROM DISK-SAM
// DLBL SQDSK,'RRDS.DATA'
// EXTENT SYS000,,,,4693,19
// ASSGN SYS000,DISK,VOL=DOS30Z,SHR
// DLBL RRDS,'CUSTOMER.R',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
        REPRO                              -
              INFILE (SQDSK                -
                 ENV (RECFM (FB)           -
                 BLKSZ (1600)              -
                 RECSZ (80)                -
                 PDEV (3330))              -
                 )                         -
              OUTFILE (RRDS)
/&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIMO322 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//SQDSK    DD VOL=SER=WTVS1R,UNIT=3330,
//         DSN=LOADRRDS,DISP=(OLD,KEEP),
//         DCB=(RECFM=FB,LRECL=80,BLKSIZE=1600)
//RRDS     DD DSN=CUSTOMER.R,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
         REPRO                             -
               INFILE (SQDSK)              -
               OUTFILE (RRDS)
/*
```

DOS/VS example (output):

```
IDC0005I NUMBER OF RECORDS PROCESSED WAS 206
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
IDC0005I NUMBER OF RECORDS PROCESSED WAS 206
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Note:

- REPLACE is only used for RRDS to RRDS copy.

DOS/VS note:

- Access Method Services needs complete specifications of any nonVSAM data set.

7.9 LISTCAT

7.9.1 LISTCAT NAME

DOS/VS example (input):

```
// JOB LIST UCAT NAME
// DLBL IJSYSUC,'PRIMER.UCAT1',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
      LISTCAT  NAME                        -
               CAT (PRIMER.UCAT1/UCATMRPW)
 /&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0332 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
      LISTCAT  NAME                        -
               CAT (PRIMER.UCAT1/UCATMRPW)
/*
```

DOS/VS example (output):

```
        LISTING FROM CATALOG -- PRIMER.UCAT1
AIX ----------- CITY.E
   DATA ------- CITY.E.D
   INDEX ------ CITY.E.I
   PATH ------- CITY.CUST.E
AIX ----------- CITY.K
   DATA ------- CITY.K.D
   INDEX ------ CITY.K.I
   PATH ------- CITY.CUST.K
CLUSTER ------- CUSTOMER.E
   DATA ------- CUSTOMER.E.D
CLUSTER ------- CUSTOMER.K
   DATA ------- CUSTOMER.K.D
   INDEX ------ CUSTOMER.K.I
CLUSTER ------- CUSTOMER.R
   DATA ------- CUSTOMER.R.D
CLUSTER ------- PRIMER.UCAT1
   DATA ------- VSAM.CATALOG.BASE.DATA.RECORD
   INDEX ------ VSAM.CATALOG.BASE.INDEX.RECORD
AIX ----------- SALESMAN.E
   DATA ------- SALESMAN.E.D
   INDEX ------ SALESMAN.E.I
   PATH ------- SALES.CUST.E
AIX ----------- SALESMAN.K
   DATA ------- SALESMAN.K.D
   INDEX ------ SALESMAN.K.I
   PATH ------- SALES.CUST.K
VOLUME -------- WTVSAM
```

OS/VS example (output):

```
        LISTING FROM CATALOG -- PRIMER.UCAT1
AIX ----------- CITY.E
   DATA ------- CITY.E.D
   INDEX ------ CITY.E.I
   PATH ------- CITY.CUST.E
AIX ----------- CITY.K
   DATA ------- CITY.K.D
   INDEX ------ CITY.K.I
   PATH ------- CITY.CUST.K
CLUSTER ------- CUSTOMER.E
   DATA ------- CUSTOMER.E.D
CLUSTER ------- CUSTOMER.K
   DATA ------- CUSTOMER.K.D
   INDEX ------ CUSTOMER.K.I
CLUSTER ------- CUSTOMER.R
   DATA ------- CUSTOMER.R.D
CLUSTER ------- PRIMER.UCAT1
   DATA ------- VSAM.CATALOG.BASE.DATA.RECORD
   INDEX ------ VSAM.CATALOG.BASE.INDEX.RECORD
AIX ----------- SALESMAN.E
   DATA ------- SALESMAN.E.D
   INDEX ------ SALESMAN.E.I
   PATH ------- SALES.CUST.E
AIX ----------- SALESMAN.K
   DATA ------- SALESMAN.K.D
   INDEX ------ SALESMAN.K.I
   PATH ------- SALES.CUST.K
VOLUME -------- WTVSAM
```

Output listing is continued on next page

```
   THE NUMBER OF ENTRIES PROCESSED WAS:
            AIX ------------------4
            CLUSTER --------------4
            DATA -----------------8
            INDEX ----------------6
            NONVSAM --------------0
            PATH -----------------4
            SPACE ----------------1
            USERCATALOG ----------0
            TOTAL ----------------27
 THE NUMBER OF PROTECTED ENTRIES SUPPRESSED WAS 0
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

```
   THE NUMBER OF ENTRIES PROCESSED WAS:
            AIX ------------------4
            ALIAS ----------------0
            CLUSTER --------------4
            DATA -----------------8
            GDG ------------------0
            INDEX ----------------6
            NONVSAM --------------0
            PATH -----------------4
            SPACE ----------------1
            USERCATALOG ----------0
            TOTAL ----------------27
 THE NUMBER OF PROTECTED ENTRIES SUPPRESSED WAS 0
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Note:

- No catalog password is required, since it is a read-only operation and the various passwords are not to be displayed. NAME indicates that only names of objects and its type will be listed.

7.9.2 LISTCAT ALL AND EXPLANATION OF THE FIELDS

This output has been produced after having defined the USER CATALOG, the SPACE, loaded the KSDS and built one alternate index (see examples page 100 through 115). The values are established by VSAM by its own logic and by default.

DOS/VS example (input):

```
// JOB LIST UCAT ALL
// DLBL IJSYSUC,'PRIMER.UCAT1',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
      LISTCAT ALL                          -
              CAT (PRIMER.UCAT1/UCATMRPW)
 /&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIMO112 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
          LISTCAT ALL                      -
                  CAT (PRIMER.UCAT1/UCATMRPW)
/*
```

Output of Access Method Services (DOS/VS and OS/VS are similar):

```
                    LISTING FROM CATALOG -- PRIMER.UCAT1
CLUSTER ------- CUSTOMER.K
  HISTORY
    OWNER-IDENT-------(NULL) CREATION----------77.355 RCVY-VOL----------WTVSAM RCVY-CI--------X'00000E'
    RELEASE----------------2 EXPIRATION--------00.000 RCVY-DEVT----X'30502009'
  PROTECTION-PSWD-----(NULL) RACF----------------(NO)
  ASSOCIATIONS
    DATA-----CUSTOMER.K.D
    INDEX----CUSTOMER.K.I
    AIX------SALESMAN.K

 DATA ------- CUSTOMER.K.D
  HISTORY
    OWNER-IDENT-------(NULL) CREATION----------77.355 RCVY-VOL----------WTVSAM RCVY-CI--------X'00000D'
    RELEASE----------------2 EXPIRATION--------00.000 RCVY-DEVT----X'30502009'
  PROTECTION-PSWD-----(NULL) RACF----------------(NO)
  ASSOCIATIONS
    CLUSTER--CUSTOMER.K
  ATTRIBUTES
    KEYLEN-----------------5 AVGLRECL-------------200 BUFSPACE------------4096 CISIZE--------------1024
    RKP--------------------0 MAXLRECL-------------200 EXCPEXIT----------(NULL) CI/CA----------------198
    SHROPTNS(1,3)   RECOVERY SUBALLOC          NOERASE INDEXED       NOWRITECHK IMBED          NOREPLICAT
    UNORDERED        NOREUSE NONSPANNED
  STATISTICS
    REC-TOTAL-----------3000 SPLITS-CI--------------0 EXCPS---------------1601
    REC-DELETED------------0 SPLITS-CA--------------0 EXTENTS---------------1
    REC-INSERTED-----------0 FREESPACE-%CI---------20 SYSTEM-TIMESTAMP:
    REC-UPDATED------------0 FREESPACE-%CA---------10      X'8BDE7F92DF1F7800'
    REC-RETRIEVED-------3010 FREESPC-BYTES-----245760
  ALLOCATION
    SPACE-TYPE------CYLINDER HI-ALLOC-RBA-----1013760
    SPACE-PRI--------------5 HI-USED-RBA------1013760
    SPACE-SEC--------------2
  VOLUME
```

```
    VOLSER------------WTVSAM PHYREC-SIZE---------1024 HI-ALLOC-RBA-----1013760 EXTENT-NUMBER----------1
    DEVTYPE------X'30502009' PHYRECS/TRK-----------11 HI-USED-RBA------1013760 EXTENT-TYPE--------X'00'
    VOLFLAG------------PRIME TRACKS/CA-------------19
    EXTENTS:
    LOW-CCHH-----X'00040000' LOW-RBA----------------0 TRACKS----------------95
    HIGH-CCHH----X'00080012' HIGH-RBA---------1013759

 INDEX ------ CUSTOMER.K.I
  HISTORY
    OWNER-IDENT-------(NULL) CREATION----------77.355 RCVY-VOL----------WTVSAM RCVY-CI--------X'00000F'
    RELEASE----------------2 EXPIRATION--------00.000 RCVY-DEVT----X'30502009'
  PROTECTION-PSWD-----(NULL) RACF----------------(NO)
  ASSOCIATIONS
    CLUSTER--CUSTOMER.K
  ATTRIBUTES

    KEYLEN-----------------5 AVGLRECL---------------0 BUFSPACE---------------0 CISIZE--------------2048
    RKP--------------------0 MAXLRECL------------2041 EXCPEXIT----------(NULL) CI/CA------------------6
    SHROPTNS(1,3)   RECOVERY SUBALLOC          NOERASE NOWRITECHK          IMBED NOREPLICAT     UNORDERED
    NOREUSE
  STATISTICS
    REC-TOTAL--------------6 SPLITS-CI--------------0 EXCPS-----------------48 INDEX:
    REC-DELETED------------0 SPLITS-CA--------------0 EXTENTS----------------2 LEVELS-----------------2
    REC-INSERTED-----------0 FREESPACE-%CI----------0 SYSTEM-TIMESTAMP:        ENTRIES/SECT----------14
    REC-UPDATED------------0 FREESPACE-%CA----------0      X'8BDE7F92DF1F7800' SEQ-SET-RBA--------12288
    REC-RETRIEVED----------0 FREESPC-BYTES------10240                          HI-LEVEL-RBA-----------0
  ALLOCATION
    SPACE-TYPE---------TRACK HI-ALLOC-RBA-------22528
    SPACE-PRI--------------1 HI-USED-RBA--------22528
    SPACE-SEC--------------1
  VOLUME
    VOLSER------------WTVSAM PHYREC-SIZE---------2048 HI-ALLOC-RBA-------12288 EXTENT-NUMBER----------1
    DEVTYPE------X'30502009' PHYRECS/TRK------------6 HI-USED-RBA---------2048 EXTENT-TYPE--------X'00'
    VOLFLAG------------PRIME TRACKS/CA--------------1
    EXTENTS:
    LOW-CCHH-----X'00030011' LOW-RBA----------------0 TRACKS-----------------1
    HIGH-CCHH----X'00030011' HIGH-RBA-----------12287
  VOLUME
    VOLSER------------WTVSAM PHYREC-SIZE---------2048 HI-ALLOC-RBA-------22528 EXTENT-NUMBER----------1
    DEVTYPE------X'30502009' PHYRECS/TRK------------6 HI-USED-RBA--------22528 EXTENT-TYPE--------X'80'
    VOLFLAG------------PRIME TRACKS/CA-------------19
    EXTENTS:
    LOW-CCHH-----X'00040000' LOW-RBA------------12288 TRACKS----------------95
    HIGH-CCHH----X'00080012' HIGH-RBA-----------22527

CLUSTER ------- PRIMER.UCAT1
  HISTORY
    OWNER-IDENT-------(NULL) CREATION----------77.355
    RELEASE----------------2 EXPIRATION--------00.000
  PROTECTION
    MASTERPW--------UCATMRPW UPDATEPW--------UCATUPDT CODE--------------(NULL) RACF----------------(NO)
    CONTROLPW---------(NULL) READPW------------(NULL) ATTEMPTS--------------2 USVR--------------(NULL)
    USAR--------------(NONE)
  ASSOCIATIONS
    DATA-----VSAM.CATALOG.BASE.DATA.RECORD
    INDEX----VSAM.CATALOG.BASE.INDEX.RECORD
```

```
DATA ------- VSAM.CATALOG.BASE.DATA.RECORD
 HISTORY
   OWNER-IDENT-------(NULL) CREATION----------00.000
   RELEASE----------------2 EXPIRATION--------00.000
 PROTECTION-PSWD-----(NULL) RACF----------------(NO)
 ASSOCIATIONS
   CLUSTER--PRIMER.UCAT1

 ATTRIBUTES
   KEYLEN-----------------44 AVGLRECL-------------505 BUFSPACE------------3072 CISIZE---------------512
   RKP--------------------0 MAXLRECL-------------505 EXCPEXIT----------(NULL) CI/CA-----------------40
   SHROPTNS(3,3)   RECOVERY SUBALLOC          NOERASE INDEXED        NOWRITECHK IMBED          NOREPLICAT
   UNORDERED        NOREUSE NONSPANNED           BIND RECVABLE
 STATISTICS
   REC-TOTAL-------------15 SPLITS-CI--------------0 EXCPS-----------------22
   REC-DELETED------------0 SPLITS-CA--------------0 EXTENTS----------------2
   REC-INSERTED-----------0 FREESPACE-%CI----------0 SYSTEM-TIMESTAMP:
   REC-UPDATED------------0 FREESPACE-%CA----------0      X'8BDE7F4D55E43800'
   REC-RETRIEVED----------0 FREESPC-BYTES-----218112
 ALLOCATION
   SPACE-TYPE---------TRACK HI-ALLOC-RBA------225280
   SPACE-PRI-------------33 HI-USED-RBA-------225280
   SPACE-SEC-------------18
 VOLUME
   VOLSER------------WTVSAM PHYREC-SIZE----------512 HI-ALLOC-RBA------204800 EXTENT-NUMBER----------1
   DEVTYPE------X'30502009' PHYRECS/TRK-----------20 HI-USED-RBA--------20480 EXTENT-TYPE--------X'00'
   VOLFLAG------------PRIME TRACKS/CA--------------3
   LOW-KEY---------------00
   HIGH-KEY--------------3F
   HI-KEY-RBA----------6144
   EXTENTS:
   LOW-CCHH-----X'00010000' LOW-RBA----------------0 TRACKS----------------30
   HIGH-CCHH----X'0002000A' HIGH-RBA----------204799
 VOLUME
   VOLSER------------WTVSAM PHYREC-SIZE----------512 HI-ALLOC-RBA------225280 EXTENT-NUMBER----------1
   DEVTYPE------X'30502009' PHYRECS/TRK-----------20 HI-USED-RBA-------225280 EXTENT-TYPE--------X'00'
   VOLFLAG------------PRIME TRACKS/CA--------------3
   LOW-KEY---------------40
   HIGH-KEY--------------FF
   HI-KEY-RBA--------204800
   EXTENTS:
   LOW-CCHH-----X'0002000E' LOW-RBA-----------204800 TRACKS-----------------3
   HIGH-CCHH----X'00020010' HIGH-RBA----------225279

INDEX ------ VSAM.CATALOG.BASE.INDEX.RECORD
 HISTORY
   OWNER-IDENT-------(NULL) CREATION----------00.000
   RELEASE----------------2 EXPIRATION--------00.000
 PROTECTION-PSWD-----(NULL) RACF----------------(NO)
 ASSOCIATIONS
   CLUSTER--PRIMER.UCAT1
 ATTRIBUTES
   KEYLEN-----------------44 AVGLRECL---------------0 BUFSPACE---------------0 CISIZE---------------512
   RKP--------------------0 MAXLRECL-------------505 EXCPEXIT----------(NULL) CI/CA-----------------20
   SHROPTNS(3,3)   RECOVERY SUBALLOC          NOERASE NOWRITECHK          IMBED NOREPLICAT     UNORDERED
```

```
    NOREUSE              BIND

  STATISTICS
    REC-TOTAL-------------3 SPLITS-CI-------------0 EXCPS----------------13 INDEX:
    REC-DELETED-----------0 SPLITS-CA-------------0 EXTENTS---------------3 LEVELS----------------2
    REC-INSERTED----------0 FREESPACE-%CI----------0 SYSTEM-TIMESTAMP:        ENTRIES/SECT----------7
    REC-UPDATED-----------0 FREESPACE-%CA----------0      X'8BDE7F4D55E43800' SEQ-SET-RBA--------30720
    REC-RETRIEVED---------0 FREESPC-BYTES------34816                          HI-LEVEL-RBA----------0
  ALLOCATION
    SPACE-TYPE---------TRACK HI-ALLOC-RBA-------36352
    SPACE-PRI----------------3 HI-USED-RBA--------36352
    SPACE-SEC----------------3
  VOLUME
    VOLSER-----------WTVSAM PHYREC-SIZE----------512 HI-ALLOC-RBA-------30720 EXTENT-NUMBER----------1
    DEVTYPE------X'30502009' PHYRECS/TRK----------20 HI-USED-RBA----------512 EXTENT-TYPE--------X'00'
    VOLFLAG------------PRIME TRACKS/CA--------------1
    EXTENTS:
    LOW-CCHH-----X'0002000B' LOW-RBA---------------0 TRACKS----------------3
    HIGH-CCHH----X'0002000D' HIGH-RBA----------30719
  VOLUME
    VOLSER-----------WTVSAM PHYREC-SIZE----------512 HI-ALLOC-RBA-------35840 EXTENT-NUMBER----------1
    DEVTYPE------X'30502009' PHYRECS/TRK----------20 HI-USED-RBA--------31232 EXTENT-TYPE--------X'80'
    VOLFLAG------------PRIME TRACKS/CA--------------3
    LOW-KEY---------------00
    HIGH-KEY--------------3F
    EXTENTS:
    LOW-CCHH-----X'00010000' LOW-RBA-----------30720 TRACKS----------------30
    HIGH-CCHH----X'0002000A' HIGH-RBA----------35839
  VOLUME
    VOLSER-----------WTVSAM PHYREC-SIZE----------512 HI-ALLOC-RBA-------36352 EXTENT-NUMBER----------2
    DEVTYPE------X'30502009' PHYRECS/TRK----------20 HI-USED-RBA--------36352 EXTENT-TYPE--------X'80'
    VOLFLAG------------PRIME TRACKS/CA--------------3
    LOW-KEY---------------40
    HIGH-KEY--------------FF
    EXTENTS:
    LOW-CCHH-----X'0002000E' LOW-RBA-----------35840 TRACKS----------------3
    HIGH-CCHH----X'00020010' HIGH-RBA----------36351

AIX ----------- SALESMAN.K
  HISTORY
    OWNER-IDENT-------(NULL) CREATION----------77.355 RCVY-VOL----------WTVSAM RCVY-CI--------X'000010'
    RELEASE----------------2 EXPIRATION--------00.000 RCVY-DEVT----X'30502009'
  PROTECTION-PSWD-----(NULL) RACF----------------(NO)
  ASSOCIATIONS
    DATA-----SALESMAN.K.D
    INDEX----SALESMAN.K.I
    CLUSTER--CUSTOMER.K
    PATH-----SALES.CUST.K
  ATTRIBUTES
    UPGRADE

 DATA ------- SALESMAN.K.D
  HISTORY
    OWNER-IDENT-------(NULL) CREATION----------77.355 RCVY-VOL----------WTVSAM RCVY-CI--------X'000011'
    RELEASE----------------2 EXPIRATION--------00.000 RCVY-DEVT----X'30502009'
  PROTECTION-PSWD-----(NULL) RACF----------------(NO)
```

```
ASSOCIATIONS
  AIX------SALESMAN.K
ATTRIBUTES
  KEYLEN----------------12 AVGLRECL-----------4086 BUFSPACE-----------8704 CISIZE-------------4096
  RKP-------------------5 MAXLRECL----------32600 EXCPEXIT----------(NULL) CI/CA-----------------54
  AXRKP-----------------40
  SHROPTNS(1,3)   RECOVERY SUBALLOC         NOERASE INDEXED       NOWRITECHK IMBED         NOREPLICAT
  UNORDERED        NOREUSE SPANNED       NONUNIQKEY
STATISTICS
  REC-TOTAL-------------50 SPLITS-CI-------------0 EXCPS----------------26
  REC-DELETED-----------0 SPLITS-CA-------------0 EXTENTS---------------1
  REC-INSERTED----------0 FREESPACE-%CI---------20 SYSTEM-TIMESTAMP:
  REC-UPDATED-----------0 FREESPACE-%CA---------10      X'8BDE7FB0B33BB800'
  REC-RETRIEVED----------6 FREESPC-BYTES-----196608
ALLOCATION
  SPACE-TYPE------CYLINDER HI-ALLOC-RBA------221184
  SPACE-PRI--------------1 HI-USED-RBA-------221184
  SPACE-SEC--------------1
VOLUME
  VOLSER------------WTVSAM PHYREC-SIZE---------4096 HI-ALLOC-RBA------221184 EXTENT-NUMBER----------1
  DEVTYPE------X'30502009' PHYRECS/TRK------------3 HI-USED-RBA-------221184 EXTENT-TYPE--------X'00'
  VOLFLAG------------PRIME TRACKS/CA-------------19
  EXTENTS:
  LOW-CCHH-----X'00090000' LOW-RBA----------------0 TRACKS----------------19
  HIGH-CCHH----X'00090012' HIGH-RBA----------221183

INDEX ------ SALESMAN.K.I
HISTORY
  OWNER-IDENT-------(NULL) CREATION----------77.355 RCVY-VOL----------WTVSAM RCVY-CI--------X'000012'
  RELEASE----------------2 EXPIRATION--------00.000 RCVY-DEVT----X'30502009'
PROTECTION-PSWD-----(NULL) RACF----------------(NO)
ASSOCIATIONS
  AIX------SALESMAN.K
ATTRIBUTES
  KEYLEN----------------12 AVGLRECL---------------0 BUFSPACE---------------0 CISIZE---------------512
  RKP-------------------5 MAXLRECL-------------505 EXCPEXIT----------(NULL) CI/CA-----------------20
  SHROPTNS(1,3)   RECOVERY SUBALLOC         NOERASE NOWRITECHK          IMBED NOREPLICAT     UNORDERED
  NOREUSE
STATISTICS
  REC-TOTAL--------------1 SPLITS-CI-------------0 EXCPS-----------------6 INDEX:
  REC-DELETED------------0 SPLITS-CA-------------0 EXTENTS---------------2 LEVELS-----------------1
  REC-INSERTED-----------0 FREESPACE-%CI----------0 SYSTEM-TIMESTAMP:        ENTRIES/SECT-----------7
  REC-UPDATED------------0 FREESPACE-%CA----------0      X'8BDE7FB0B33BB800' SEQ-SET-RBA--------10240

  REC-RETRIEVED----------0 FREESPC-BYTES------10240                          HI-LEVEL-RBA-------10240
ALLOCATION
  SPACE-TYPE---------TRACK HI-ALLOC-RBA-------10752
  SPACE-PRI--------------1 HI-USED-RBA--------10752
  SPACE-SEC--------------1
VOLUME
  VOLSER------------WTVSAM PHYREC-SIZE----------512 HI-ALLOC-RBA-------10240 EXTENT-NUMBER----------1
  DEVTYPE------X'30502009' PHYRECS/TRK-----------20 HI-USED-RBA------------0 EXTENT-TYPE--------X'00'
  VOLFLAG------------PRIME TRACKS/CA-------------1
  EXTENTS:
  LOW-CCHH-----X'00030012' LOW-RBA----------------0 TRACKS----------------1
  HIGH-CCHH----X'00030012' HIGH-RBA-----------10239
```

```
 VOLUME
   VOLSER------------WTVSAM PHYREC-SIZE----------512 HI-ALLOC-RBA-------10752 EXTENT-NUMBER----------1
   DEVTYPE------X'30502009' PHYRECS/TRK-----------20 HI-USED-RBA--------10752 EXTENT-TYPE--------X'80'
   VOLFLAG------------PRIME TRACKS/CA-------------19
   EXTENTS:
   LOW-CCHH-----X'00090000' LOW-RBA------------10240 TRACKS----------------19
   HIGH-CCHH----X'00090012' HIGH-RBA-----------10751

PATH ------- SALES.CUST.K
 HISTORY
   OWNER-IDENT-------(NULL) CREATION----------77.355 RCVY-VOL----------WTVSAM RCVY-CI--------X'000014'
   RELEASE----------------2 EXPIRATION--------00.000 RCVY-DEVT----X'30502009'
 PROTECTION-PSWD-----(NULL) RACF----------------(NO)
 ASSOCIATIONS
   AIX------SALESMAN.K
   DATA-----SALESMAN.K.D
   INDEX----SALESMAN.K.I
   DATA-----CUSTOMER.K.D
   INDEX----CUSTOMER.K.I
 ATTRIBUTES
   UPDATE

VOLUME -------- WTVSAM
 HISTORY
   RELEASE----------------2 RCVY-VOL----------WTVSAM RCVY-DEVT----X'30502009' RCVY-CI--------X'000009'
 CHARACTERISTICS
   BYTES/TRK----------13165 DEVTYPE------X'30502009' MAX-PHYREC-SZ------13030 DATASETS-ON-VOL--------5
   TRKS/CYL--------------19 VOLUME-TIMESTAMP:        MAX-EXT/ALLOC----------5 DATASPCS-ON-VOL--------2
   CYLS/VOL-------------411      X'8BDE7F9BA51F9000'
 DATASPACE
   DATASETS---------------3 FORMAT-1-DSCB:           ATTRIBUTES:
   EXTENTS----------------1 CCHHR------X'0000000303' SUBALLOC
   SEC-ALLOC--------------1 TIMESTAMP                EXPLICIT
   TYPE------------CYLINDER      X'8BDE7F4AFA2AB800' USERCAT
   EXTENT-DESCRIPTOR:
   TRACKS-TOTAL----------57 BEG-CCHH-----X'00010000' SPACE-MAP-----------39

   TRACKS-USED-----------57
   DATASET-DIRECTORY:
     DSN----PRIMER.UCAT1                             ATTRIBUTES--------(NULL) EXTENTS----------------3
     DSN----CUSTOMER.K.I                             ATTRIBUTES--------(NULL) EXTENTS----------------1
     DSN----SALESMAN.K.I                             ATTRIBUTES--------(NULL) EXTENTS----------------1
 DATASPACE
   DATASETS---------------2 FORMAT-1-DSCB:           ATTRIBUTES:
   EXTENTS----------------1 CCHHR------X'0000000304' SUBALLOC
   SEC-ALLOC--------------2 TIMESTAMP                EXPLICIT
   TYPE------------CYLINDER      X'8BDE7F50C9DD7800'
   EXTENT-DESCRIPTOR:
   TRACKS-TOTAL--------1900 BEG-CCHH-----X'00040000' SPACE-MAP-----------72FD06FA
   TRACKS-USED----------114
   DATASET-DIRECTORY:
     DSN----CUSTOMER.K.D                             ATTRIBUTES--------(NULL) EXTENTS----------------1
     DSN----SALESMAN.K.D                             ATTRIBUTES--------(NULL) EXTENTS----------------1

     THE NUMBER OF ENTRIES PROCESSED WAS:
              AIX ------------------1
```

```
              ALIAS -----------------0  (*)
              CLUSTER ---------------2
              DATA ------------------3
              GDG -------------------0  (*)
              INDEX -----------------3
              NONVSAM ---------------0
              PAGESPACE -------------0  (*)
              PATH ------------------1
              SPACE -----------------1
              USERCATALOG -----------0
              TOTAL ----------------11
  THE NUMBER OF PROTECTED ENTRIES SUPPRESSED WAS 0
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

(*) = these lines do not apply to DOS/VS.

7.9.2.1 EXTRACTING ALL ALLOCATIONS FROM LISTCAT ALL

The following information containing the allocation values of all components on the volume WTVSAM has been extracted from the previous list. The entries have been sorted to show the sequence of allocating space (see figure 41 in next section on page 151).

1. User catalog data component:

The first extent describes the Low-Keyrange and the second the High-Keyrange (the index allocation is between).

```
   CLUSTER ------- PRIMER.UCAT1
      DATA ------- VSAM.CATALOG.BASE.DATA.RECORD
        VOLUME
          EXTENTS:
          LOW-CCHH-----X'00010000' LOW-RBA----------------0 TRACKS----------------30
          HIGH-CCHH----X'0002000A' HIGH-RBA----------204799

        VOLUME
          EXTENTS:
          LOW-CCHH-----X'0002000E' LOW-RBA-----------204800 TRACKS-----------------3
          HIGH-CCHH----X'00020010' HIGH-RBA----------225279
```

2. User catalog Index component:

```
CLUSTER ------- PRIMER.UCAT1
   INDEX ------ VSAM.CATALOG.BASE.INDEX.RECORD
     VOLUME
       EXTENTS:
       LOW-CCHH-----X'0002000B' LOW-RBA----------------0 TRACKS-----------------3
       HIGH-CCHH----X'0002000D' HIGH-RBA-----------30719
```

Note: The second and third volume entry of the index component repeat the data component entries, since the Index Sequence Set records are stored with the data (IMBED).

3. Cluster CUSTOMER.K Index component

The cluster was defined with IMBED. This is the Index Set. The Index Sequence Set is stored with the data component.

```
CLUSTER ------- CUSTOMER.K
   INDEX ------ CUSTOMER.K.I
     VOLUME
       EXTENTS:
       LOW-CCHH-----X'00030011' LOW-RBA----------------0 TRACKS-----------------1
       HIGH-CCHH----X'00030011' HIGH-RBA-----------12287
```

Note: The second and third volume entry of the index component repeat the data component entries, since the Index Sequence Set records are stored with the data (IMBED).

4. Alternate index SALESMAN.K Index component:

The alternate index was defined with IMBED (see description above).

```
AIX ------------ SALESMAN.K
   INDEX ------ SALESMAN.K.I
     VOLUME
       EXTENTS:
       LOW-CCHH-----X'00030012' LOW-RBA----------------0 TRACKS-----------------1
       HIGH-CCHH----X'00030012' HIGH-RBA-----------10239
```

Note: The second and third volume entry of the index component repeat the data component entries, since the Index Sequence Set records are stored with the data (IMBED).

5. Cluster CUSTOMER.K Data component

```
CLUSTER ------- CUSTOMER.K
   DATA ------- CUSTOMER.K.D
     VOLUME
       EXTENTS:
       LOW-CCHH-----X'00040000' LOW-RBA----------------0 TRACKS----------------95
       HIGH-CCHH----X'00080012' HIGH-RBA---------1013759
```

6. Alternate index SALESMAN.K Data component:

```
AIX ----------- SALESMAN.K
   DATA ------- SALESMAN.K.D
     VOLUME
       EXTENTS:
       LOW-CCHH-----X'00090000' LOW-RBA----------------0 TRACKS----------------19
       HIGH-CCHH----X'00090012' HIGH-RBA----------221183
```

7.9.2.2 ALLOCATION LAYOUT ON THE VSAM 3330 VOLUME

See description in section 7.9.2.1 on page 148

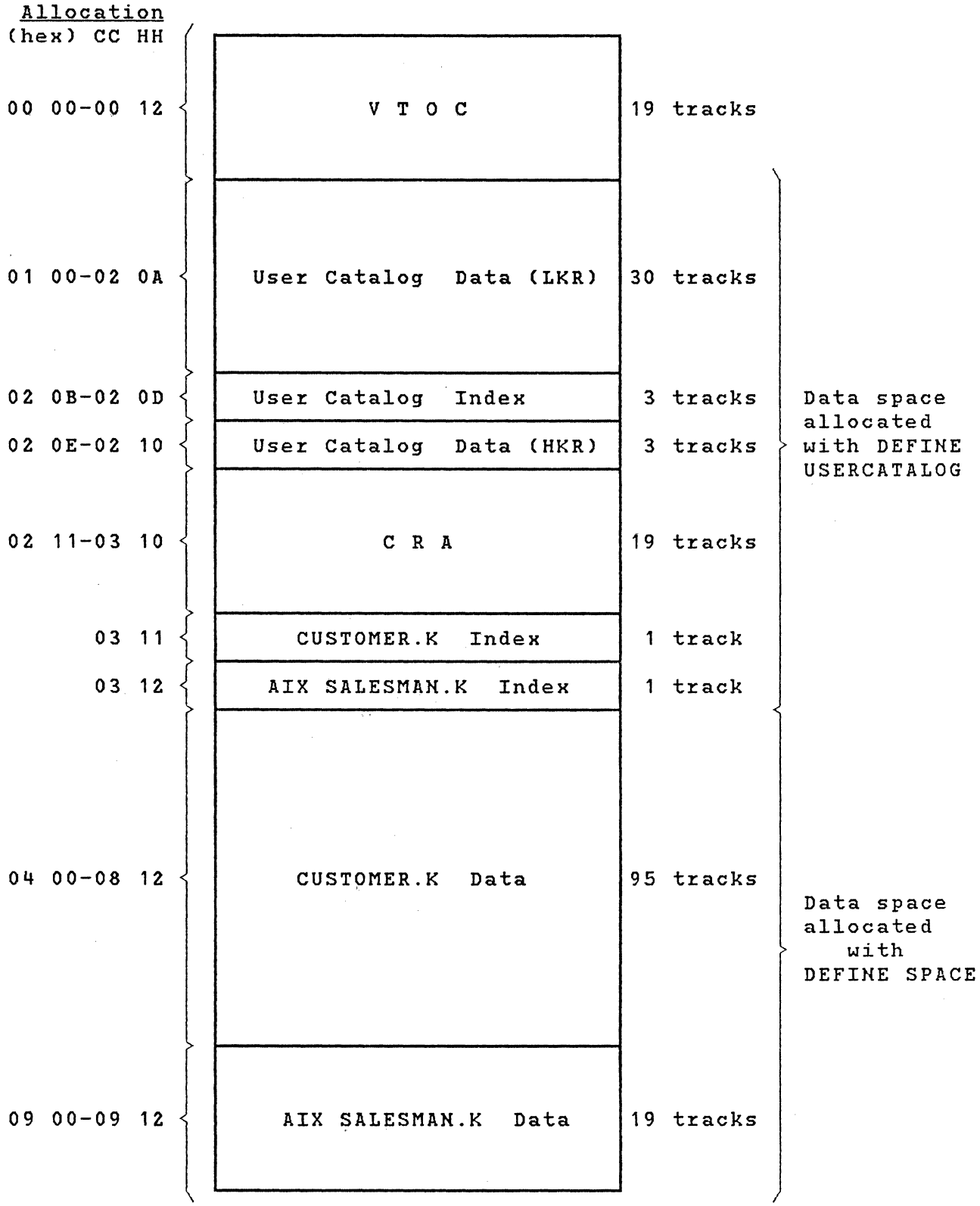


Figure 41. Volume allocation layout

7.9.2.3 EXPLANATION OF THE LISTCAT OUTPUT FIELDS:

Cluster entry

```
CLUSTER ------- CUSTOMER.K
  HISTORY
    OWNER-IDENT-------(NULL) CREATION----------77.355 RCVY-VOL----------WTVSAM RCVY-CI--------X'00000E'
    RELEASE----------------2 EXPIRATION--------00.000 RCVY-DEVT----X'30502009'
  PROTECTION-PSWD-----(NULL) RACF----------------(NO)
  ASSOCIATIONS
    DATA-----CUSTOMER.K.D
    INDEX----CUSTOMER.K.I
    AIX------SALESMAN.K
```

Explanation:

OWNER-IDENT: 1 to 8 characters that can be specified for DEFINE CLUSTER. Used to identify the OWNER (owner-id) of the object. '(NULL)' means no owner-id was specified.

CREATION: the julian date (YY.DDD) the entry was created.

EXPIRATION: the cluster can be deleted without specifying the PURGE parameter (00.000).

RCVY-VOL: the volume serial where the CRA containing duplicated catalog records for this object is located

RCVY-CI: the CI number in the CRA which contains the duplicated catalog entry for this cluster. X'0E' means CI number 14.

RELEASE: Release number of VSAM under which the entry was created (the format of catalog records may change between releases, however, externals for the user are maintained).

```
DOS/VS: 1 = DOS/VS Rel. 28, 29, and 30
        2 = DOS/VS Rel. 31 and up
OS/MVS: 1 = OS/VS2 Rel. 2 and 3
        2 = OS/VS2 Rel. 3.6 and up
OS/SVS: 1 = SVS without VSAM ICR
        2 = SVS with VSAM ICR
OS/VS1: 1 = OS/VS1 incl. Rel. 3.0
        2 = OS/VS1 Rel 4.0 and up
```

RECY-DEVT: device type of the volume containing the CRA for this object. A translate table is included in the Access Method Services manual and in section C.1 on page 279. X'30502009' means 3330.

The explanations are continued on the next page.

PROTECTION-PSWD: indicates the MASTERPW, CONTROLPW, UPDATEPW and/or READPW for this object. '(NULL)' means no password has been specified.

RACF: (this heading appears in MVS only) '(NO)' means this entry is not RACF protected.

ASSOCIATIONS: this group lists the types and entry names of each entry associated with the present entry. This object (cluster) is associated with its data and index objects and also with an alternate index (AIX) cluster.

Cluster data component entry

```
DATA ------- CUSTOMER.K.D
 HISTORY
   OWNER-IDENT-------(NULL) CREATION----------77.355 RCVY-VOL----------WTVSAM RCVY-CI--------X'00000D'
   RELEASE----------------2 EXPIRATION--------00.000 RCVY-DEVT----X'30502009'
 PROTECTION-PSWD-----(NULL) RACF----------------(NO)
 ASSOCIATIONS
   CLUSTER--CUSTOMER.K
 ATTRIBUTES
   KEYLEN-----------------5 AVGLRECL-------------200 BUFSPACE------------4096 CISIZE--------------1024
   RKP--------------------0 MAXLRECL-------------200 EXCPEXIT----------(NULL) CI/CA----------------198
   SHROPTNS(1,3)   RECOVERY SUBALLOC          NOERASE INDEXED       NOWRITECHK IMBED          NOREPLICAT
   UNORDERED        NOREUSE NONSPANNED
```

Explanation:

For explanation of HISTORY, PROTECTION and ASSOCIATIONS see previous section.

Note that 3850 (MSS) default parameters NODESTAGEWAIT and STAGE are not listed with LISTCAT.

KEYLEN: length of primary key (5 bytes).

RKP: relative key position in the logical record (key starts with Byte 0).

AVGLRECL: logical record length average is 200 bytes.

MAXLRECL: maximum record length is 200 bytes.

The explanations are continued on the next page

BUFSPACE: minimum space in bytes that will be used by VSAM for data and index buffers for this KSDS cluster. This value was defaulted by VSAM as no parameter was specified. It can be specified with the DEFINE command or changed with the ALTER command. It can also be temporarily overridden by a user program at OPEN time either by specifying a greater value in the ACB (see chapter 8.11 starting on page 217) or in the JCL (see appropriate section in chapter 6.0).

CISIZE: control interval size for this data component in bytes.

CI/CA: number of control intervals per control area.

EXCPEXIT: '(NULL)' means no exception exit routine entry was specified in the DEFINE command.

SHROPTNS: the SHAREOPTIONS were defaulted to 1/3 (see section 7.2.7 on page 95).

RECOVERY: is a loading option (see description in section 7.2.8 on page 97.
The opposite attribute is SPEED which gives better performance when loading the data set, but does not provide the RECOVERY facilities (also see section 7.2.8 on page 97).

SUBALLOC: the space for this cluster is suballocated by VSAM from a suballocatable data space.

NOERASE: the data are not overwritten with X'00' when the entry is deleted.

INDEXED: INDEXED identifies this cluster as KSDS.

NOWRITECHECK: write operations are not checked for correctness. Specification of WRITECHECK results in time consuming operations not needed for DASDs like 3330, 3340, or 3350.

IMBED: the sequence-set index record is stored along with its data control area and is replicated on its track.

NOREPLICAT: the index records on the higher index level are <u>not</u> replicated (NOREPLICAT) (also see section 7.2.3 on page 89).

UNORDERED: this is a meaningless default parameter as only on volume is used. If more volumes are used, it specifies the order of using the volumes for allocation.

NOREUSE: this data set cannot be used as temporary work data set. This parameter <u>cannot</u> be changed with ALTER.

NONSPANNED: the records cannot span control intervals.

Cluster data component entry (statistics)

```
STATISTICS
  REC-TOTAL-----------3000 SPLITS-CI--------------0 EXCPS---------------1601
  REC-DELETED---------------0 SPLITS-CA--------------0 EXTENTS----------------1
  REC-INSERTED--------------0 FREESPACE-%CI---------20 SYSTEM-TIMESTAMP:
  REC-UPDATED---------------0 FREESPACE-%CA---------10      X'8BDE7F92DF1F7800'
  REC-RETRIEVED-------3010 FREESPC-BYTES-----245760
```

Explanation:

REC-TOTAL: this data component contains 3000 records.

REC-DELETED: no records have been deleted since the data set was loaded.

REC-INSERTED: no records have been inserted after loading the data set (records added at the end are not counted here, therefore this field will always be '0' for an ESDS).

REC-UPDATED: no records have been updated in the data set. This field is updated when a PUT UPDATE request is issued for a record.

REC-RETRIEVED: 3010 records have been read (3000 by BLDINDEX (page 113) when the alternate index was created and 10 by PRINT (page 109 and page 115)).

SPLITS CI/CA: number of CI and CA splits since the data set was loaded. The 'SPLIT CA' field should be examined frequently. If the value increases too much, the data set should be reorganized.

FREESPACE-%CI/CA: 20% free space in the CI (1024 * 20% = 204 bytes) and 10% of the number of CI per CA (198 * 10% = 19 CI) are left free when the data set is loaded (also see section 7.2.2 on page 88).

FREESPC-BYTES: this is the amount of free bytes in the data component, excluding the free space in the partially filled CI's. This value can be calculated as follows:

1. records per CI (minus free space) = 4
2. CI required (3000 rcd / 4) = 750
3. tot. alloc. CIs(198 (CI/CA) * 5 (CA=cyl)) = 990
4. free CIs (990 - 750 = 240) * 1024 bytes = 245760

EXCPS: 1601 EXCP (execute channel program - SVC 0) macro instructions have been issued against this data component (about 50% when loading the data set, the rest by BLDINDEX and PRINT).

The explanations are continued on the next page

EXTENTS: the number of extents of this data component.

SYSTEM-TIMESTAMP: the time (System/370 time-of-day clock value) this data component was last closed (no important timestamp).

Cluster data component entry (allocation)

```
ALLOCATION
  SPACE-TYPE------CYLINDER HI-ALLOC-RBA-----1013760
  SPACE-PRI-------------5 HI-USED-RBA------1013760
  SPACE-SEC-------------2
```

Explanation:

SPACE-TYPE: indicates whether the suballocation of this data component is in terms of CYLINDERs or TRACKs.

SPACE-PRI: contains the number of units indicated by SPACE-TYPE of space allocated to this data component when it was defined (the same amount would be allocated on a new CANDIDATE volume if the data component space must be extended.

SPACE-SEC: contains the number of units indicated by SPACE-TYPE of space to be allocated whenever the data component is to be extended on the same volume.

HI-ALLOC-RBA: the highest RBA (plus 1) allocated by VSAM for this data component to store data. The calculation is as follows:

990 (CIs) * 1024 (bytes/CI) = 1 013 760 bytes

HI-USED-RBA: the highest RBA (plus 1) within allocated space that actually contains data. Since the data set was loaded with the RECOVERY option, the last CA is always preformatted. Actually it does not contain records at the end but some FREESPC-BYTES for extending/adding more data records.

Cluster data component entry (volume information)

```
VOLUME
  VOLSER------------WTVSAM PHYREC-SIZE---------1024 HI-ALLOC-RBA-----1013760 EXTENT-NUMBER----------1
  DEVTYPE------X'30502009' PHYRECS/TRK-----------11 HI-USED-RBA------1013760 EXTENT-TYPE--------X'00'
  VOLFLAG------------PRIME TRACKS/CA-------------19
  EXTENTS:
  LOW-CCHH-----X'00040000' LOW-RBA----------------0 TRACKS----------------95
  HIGH-CCHH----X'00080012' HIGH-RBA---------1013759
```

Explanation:

VOLSER: the volume serial number WTVSAM.

DEVTYPE: the device type is 3330 (the Access Method Services manual contains a 'Device Type Translate Table' to translate this code, also see section C.1 on page 279).

VOLFLAG: PRIME indicates that this is the first volume on which data records for this data component are stored. Other values may indicate OVERFLOW when the component contains a Keyrange, or CANDIDATE where the component can be extended.

PHYREC-SIZE: physical record size (sometimes called block size) that VSAM uses to write CIs for this data component on this device type (also see section 8.2, page 196).

PHYSRECS/TRK: 11 physical records with 1024 bytes (no key) are written per track on a 3330.

TRACKS/CA: one CA is 19 tracks. Since the allocation was made in terms of CYLINDERs, VSAM set the CA size equal to one cylinder. One cylinder on a 3330 contains 19 tracks.

HI-ALLOC-RBA: see previous page (allocation).
HI-USED-RBA : see previous page (allocation).

EXTENT-NUMBER: number of extents allocated for this data component on volume WTVSAM.

EXTENT-TYPE: X'00' means the extents are contiguous.
X'40' means the extents are not preformatted.
X'80' is only used in an index component. As the volume group for the data component is repeated in an index component when IMBED is specified, X'80' indicates that the sequence set occupies the first track of the data CA.

LOW/HIGH-RBA: the RBA indicating the begin/end of this extent.

TRACKS: 95 tracks (5 cylinders) are allocated for this extent.

Cluster index component entry

```
INDEX ------ CUSTOMER.K.I
 HISTORY
   OWNER-IDENT-------(NULL) CREATION----------77.355 RCVY-VOL----------WTVSAM RCVY-CI--------X'00000F'
   RELEASE----------------2 EXPIRATION--------00.000 RCVY-DEVT----X'30502009'
 PROTECTION-PSWD-----(NULL) RACF----------------(NO)
 ASSOCIATIONS
   CLUSTER--CUSTOMER.K
 ATTRIBUTES

   KEYLEN-----------------5 AVGLRECL---------------0 BUFSPACE---------------0 CISIZE--------------2048
   RKP--------------------0 MAXLRECL------------2041 EXCPEXIT----------(NULL) CI/CA------------------6
   SHROPTNS(1,3)   RECOVERY SUBALLOC          NOERASE NOWRITECHK          IMBED NOREPLICAT     UNORDERED
   NOREUSE
 STATISTICS
   REC-TOTAL--------------6 SPLITS-CI--------------0 EXCPS-----------------48 INDEX:
   REC-DELETED------------0 SPLITS-CA--------------0 EXTENTS----------------2 LEVELS-----------------2
   REC-INSERTED-----------0 FREESPACE-%CI----------0 SYSTEM-TIMESTAMP:        ENTRIES/SECT----------14
   REC-UPDATED------------0 FREESPACE-%CA----------0      X'8BDE7F92DF1F7800' SEQ-SET-RBA--------12288
   REC-RETRIEVED----------0 FREESPC-BYTES------10240                          HI-LEVEL-RBA-----------0
```

Explanation:

Almost every parameter was explained in the previous sections.

The only new and/or important parameters here are:

INDEX LEVELS: '2' specifies that 2 index levels are built. The first level is the 'Sequence set level' which is stored with the data component (IMBED). The second level is the 'Index set level'.

REC-TOTAL: '6' means a total of 6 index records (CIs) exist. The sequence set consists of 5 index records (one per CA) and the index set consists of one index record (with the five entries pointing to the sequence set records).

Catalog volume record (with volume timestamp)

```
VOLUME -------- WTVSAM
  HISTORY
    RELEASE----------------2 RCVY-VOL----------WTVSAM RCVY-DEVT----X'30502009' RCVY-CI--------X'000009'
  CHARACTERISTICS
    BYTES/TRK----------13165 DEVTYPE------X'30502009' MAX-PHYREC-SZ------13030 DATASETS-ON-VOL--------5
    TRKS/CYL--------------19 VOLUME-TIMESTAMP:        MAX-EXT/ALLOC----------5 DATASPCS-ON-VOL--------2
    CYLS/VOL-------------411      X'8BDE7F9BA51F9000'
```

Explanation:

BYTES/TRK: specifies the maximum bytes per track (not all usable by VSAM).

TRKS/CYL: this device type (3330) has 19 tracks per cylinder.

CYL/VOL: this device type (3330) has 411 cylinder per volume.

DEVTYPE: this is a 3330 (see explanation in previous section).

MAX-PHYREC-SZ: the theoretical maximum physical record size VSAM can write on this volume (see physical record sizes in section 8.2 on page 196).

MAX-EXT/ALLOC: the maximum number of extents per allocation request. This means if a primary or secondary allocation request for a suballocated data set (in OS/VS also for an UNIQUE data set) is issued, for example, for 10 cylinders, VSAM (or in OS/VS for an UNIQUE data set OS/DADSM) tries to allocate these 10 cylinders in a maximum of 5 pieces.

DATASETS-ON-VOL: there are 5 logical data sets stored on the volume:

1. Catalog - CLUSTER
2. KSDS - DATA
3. KSDS - INDEX
4. AIX - DATA
5. AIX - INDEX

DATASPCS-ON-VOL: there are 2 data spaces on the volume (one created with DEFINE USERCATALOG, and one created with DEFINE SPACE).

VOLUME-TIMESTAMP: this is the most and only important timestamp in VSAM. This timestamp is checked by VSAM against the timestamp in the VTOC F-4 DSCB timestamp.

The timestamp is described in detail in section A.2 on page 244.

7.10 LISTCRA

7.10.1 LISTCRA NOCOMPARE

DOS/VS example (input):

```
// JOB LIST CRA NO COMPARE
// DLBL WTVSAM,,,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
       LISTCRA  INFILE (WTVSAM)          -
                NOCOMPARE NAME           -
                MASTERPW (MCATMRPW)      -
       CAT (PRIMER.UCAT1/UCATMPRW WTVSAM)
 /&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIMO342 EXEC PGM=IDCAMS
//WTVSAM   DD VOL=SER=WTVSAM,DISP=OLD,UNIT=3330
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
   LISTCRA  INFILE (WTVSAM)              -
            NOCOMPARE NAME               -
            MASTERPW (MCATMRPW)          -
            CAT (PRIMER.UCAT1/UCATMPRW WTVSAM)
/*
```

DOS/VS example (output):

```
LISTING OF CRA FOR VOLUME --WTVSAM-- VSAM ENTRIES
VOL  - WTVSAM
    CRAVOLRCD  -  10/20/77   15:50:33
    F4DSCBVSAM -  10/20/77   15:50:33
    F4DSCBDUMP -  10/20/77   15:50:33
AIX  - CITY.E
  DATA - CITY.E.D
    DATA VOL -
      WTVSAM
  INDX - CITY.E.I
    INDX VOL -
      WTVSAM
      WTVSAM
  CLUS - CUSTOMER.E
  PATH - CITY.CUST.E
AIX  - CITY.K
  DATA - CITY.K.D
    DATA VOL -
      WTVSAM
  INDX - CITY.K.I
    INDX VOL -
      WTVSAM
      WTVSAM
  CLUS - CUSTOMER.K
  PATH - CITY.CUST.K
```

OS/VS example (output):

```
LISTING OF CRA FOR VOLUME --WTVSAM-- VSAM ENTRIES
VOL  - WTVSAM
    CRAVOLRCD  -  12/22/77   05:35:30
    F4DSCBVSAM -  12/22/77   05:35:30
    F4DSCBDUMP -  12/22/77   05:35:30
AIX  - CITY.E
  DATA - CITY.E.D
    DATA VOL -
      WTVSAM
  INDX - CITY.E.I
    INDX VOL -
      WTVSAM
      WTVSAM
  CLUS - CUSTOMER.E
  PATH - CITY.CUST.E
AIX  - CITY.K
  DATA - CITY.K.D
    DATA VOL -
      WTVSAM
  INDX - CITY.K.I
    INDX VOL -
      WTVSAM
      WTVSAM
  CLUS - CUSTOMER.K
  PATH - CITY.CUST.K
```

Output listing is continued on next page

```
CLUS - CUSTOMER.E
  DATA - CUSTOMER.E.D
    DATA VOL -
      WTVSAM
  AIX  - SALESMAN.E
    DATA VOL -
      WTVSAM
    INDX VOL -
      WTVSAM
      WTVSAM
  AIX  - CITY.E
    DATA VOL -
      WTVSAM
    INDX VOL -
      WTVSAM
      WTVSAM
  UPGD -
CLUS - CUSTOMER.K
  DATA - CUSTOMER.K.D
    DATA VOL -
      WTVSAM
  INDX - CUSTOMER.K.I
    INDX VOL -
      WTVSAM
      WTVSAM
  AIX  - SALESMAN.K
    DATA VOL -
      WTVSAM
    INDX VOL -
      WTVSAM
      WTVSAM
  AIX  - CITY.K
    DATA VOL -
      WTVSAM
    INDX VOL -
      WTVSAM
      WTVSAM
  UPGD -
CLUS - CUSTOMER.R
  DATA - CUSTOMER.R.D
    DATA VOL -
      WTVSAM
AIX  - SALESMAN.E
  DATA - SALESMAN.E.D
    DATA VOL -
      WTVSAM
  INDX - SALESMAN.E.I
    INDX VOL -
      WTVSAM
      WTVSAM
  CLUS - CUSTOMER.E
  PATH - SALES.CUST.E
```

```
CLUS - CUSTOMER.E
  DATA - CUSTOMER.E.D
    DATA VOL -
      WTVSAM
  AIX  - SALESMAN.E
    DATA VOL -
      WTVSAM
    INDX VOL -
      WTVSAM
      WTVSAM
  AIX  - CITY.E
    DATA VOL -
      WTVSAM
    INDX VOL -
      WTVSAM
      WTVSAM
  UPGD -
CLUS - CUSTOMER.K
  DATA - CUSTOMER.K.D
    DATA VOL -
      WTVSAM
  INDX - CUSTOMER.K.I
    INDX VOL -
      WTVSAM
      WTVSAM
  AIX  - SALESMAN.K
    DATA VOL -
      WTVSAM
    INDX VOL -
      WTVSAM
      WTVSAM
  AIX  - CITY.K
    DATA VOL -
      WTVSAM
    INDX VOL -
      WTVSAM
      WTVSAM
  UPGD -
CLUS - CUSTOMER.R
  DATA - CUSTOMER.R.D
    DATA VOL -
      WTVSAM
AIX  - SALESMAN.E
  DATA - SALESMAN.E.D
    DATA VOL -
      WTVSAM
  INDX - SALESMAN.E.I
    INDX VOL -
      WTVSAM
      WTVSAM
  CLUS - CUSTOMER.E
  PATH - SALES.CUST.E
```

Output listing is continued on next page

```
AIX  - SALESMAN.K
  DATA - SALESMAN.K.D
    DATA VOL -
      WTVSAM
  INDX - SALESMAN.K.I
    INDX VOL -
      WTVSAM
      WTVSAM
  CLUS - CUSTOMER.K
  PATH - SALES.CUST.K
NUMBER OF ENTRIES PROCESSED
 CLUS -    3
 DATA -    7
 AIX  -    4
 INDX -    5
 PATH -    4
 VOL  -    1
 UPGD -    2
 SUM  -   26
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

```
AIX  - SALESMAN.K
  DATA - SALESMAN.K.D
    DATA VOL -
      WTVSAM
  INDX - SALESMAN.K.I
    INDX VOL -
      WTVSAM
      WTVSAM
  CLUS - CUSTOMER.K
  PATH - SALES.CUST.K
NUMBER OF ENTRIES PROCESSED
 CLUS -    3
 DATA -    7
 AIX  -    4
 INDX -    5
 PATH -    4
 VOL  -    1
 UPGD -    2
 SUM  -   26
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Note:

- The master password of the master catalog (MASTERPW) must be specified if the master catalog is password protected.

7.10.2 LISTCRA COMPARE

DOS/VS example (input):

```
// JOB LIST CRA COMPARE
// DLBL WTVSAM,,,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
   LISTCRA INFILE (WTVSAM)          -
           COMPARE NAME             -
           CAT (PRIMER.UCAT1/UCATMRPW IJSYSUC)-
           MASTERPW (MCATMRPW)
 /&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIMO352 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=OLD
//WTVSAM   DD VOL=SER=WTVSAM,DISP=OLD,UNIT=3330
//CATVOL   DD DSN=PRIMER.UCAT1,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
    LISTCRA  INFILE (WTVSAM)          -
             COMPARE NAME             -
             MASTERPW (MCATMRPW)      -
             CAT (PRIMER.UCAT1/UCATMRPW CATVOL)
/*
```

DOS/VS example (output):

```
LISTING OF CRA FOR VOLUME --WTVSAM-- VSAM ENTRIES
    CATVOLRCD  -  10/20/77   15:50:33
    CRAVOLRCD  -  10/20/77   15:50:33
    F4DSCBVSAM -  10/20/77   15:50:33
    F4DSCBDUMP -  10/20/77   15:50:33
NUMBER OF ENTRIES PROCESSED
 CLUS -    3
 DATA -    7
 AIX  -    4
 INDX -    5
 PATH -    4
 VOL  -    1
 UPGD -    2
 SUM  -   26
IDC0665I NUMBER ENTRIES MISCOMP. IN THIS CRA - 0
IDC0877I NUMBER RECORDS MISCOMP. IN THIS CRA - 0
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
LISTING CRA FOR VOLUME --WTVSAM-- VSAM ENTRIES
    CATVOLRCD  -  12/22/77   05:35:30
    CRAVOLRCD  -  12/22/77   05:35:30
    F4DSCBVSAM -  12/22/77   05:35:30
    F4DSCBDUMP -  12/22/77   05:35:30
NUMBER ENTRIES PROCESSED
 CLUS -    3
 DATA -    7
 AIX  -    4
 INDX -    5
 PATH -    4
 VOL  -    1
 UPGD -    2
 SUM  -   26
IDC0665I NUMBER ENTRIES MISCOMP. IN THIS CRA - 0
IDC0877I NUMBER RECORDS MISCOMP. IN THIS CRA - 0
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Notes:

- LISTCRA with the COMPARE option can be used to detect the following conditions for a recoverable catalog:

  - data set not properly closed
  - inaccessible data set
  - inaccessible volume

  The mismatches detected between the catalog and the CRA are listed. Then, depending on the degree of seriousness, different procedures to recover are to be taken as suggested in the appropriate Access Method Services manual.

7.11 RESETCAT

DOS/VS example (input):

```
// JOB RESET USER CAT
// DLBL WTVSAM,,,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// DLBL IJSYSWC,'PRIMER.UCAT2',99/365,VSAM
// EXTENT SYS020,WTVS1R
// DLBL IDCUT1,,,VSAM,CAT=IJSYSWC
// EXTENT SYS020,WTVS1R
// ASSGN SYS020,DISK,VOL=WTVS1R,SHR
// EXEC IDCAMS,SIZE=AUTO
   RESETCAT  CAT (PRIMER.UCAT1/UCATMRPW IJSYSUC)-
             CRAFILES (WTVSAM ALL)          -
             WORKCAT (PRIMER.UCAT2 IJSYSWC)-
             MASTERPW (MCATMRPW)            -
             NOIGNORE
 /&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIMO362 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//         DD DSN=PRIMER.UCAT2,DISP=OLD
//UCATDD   DD DSN=PRIMER.UCAT1,DISP=SHR
//WTVSAM   DD VOL=SER=WTVSAM,UNIT=3330,DISP=OLD
//WORKF    DD DSN=WORKFILE,DISP=OLD,UNIT=3330,
//         VOL=SER=WTVS1R,AMP='AMORG'
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
 RESETCAT CAT(PRIMER.UCAT1/UCATMRPW UCATDD)-
     CRAFILES ((WTVSAM ALL))              -
     WORKCAT (PRIMER.UCAT2)               -
     MASTERPW (MCATMRPW)                  -
     WORKFILE (WORKF)
/*
```

DOS/VS example (output):

```
IDC01002I RESETCAT CATALOG PRIMER.UCAT1
          VOL WTVSAM LEVEL 10/20/77 16:24:15
IDC01011I CRA FOR RST-VOL WTVSAM
                    LEVEL 10/20/77 16:24:15
IDC01037I PRIMER.UCAT1 HAS BEEN RESET
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
IDC01002I RESETCAT CATALOG PRIMER.UCAT1
        VOL WTVSAM LEVEL 12/22/77 05:35:30
IDC01011I CRA FOR RST-VOL WTVSAM
                  LEVEL 12/22/77 05:35:30
IDC01037I PRIMER.UCAT1 HAS BEEN RESET
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Notes:

- RESETCAT must be used with caution to recover accessibility of a volume that contains a portion of a multivolume file. Prior to issuing RESETCAT, compatible levels of volumes containing multivolume files should be restored. Refer to the Access Method Services manual for more information.

- RESETCAT function requires a WORKFILE for use as temporary storage while processing the command. The space required is suballocated from VSAM space(s) on the volume specified in its EXTENT statement. Be sure the available suballocatable space on the WORKFILE volume is at least as large as the resultant catalog.

- The master password of the master catalog (MASTERPW) must be specified if the master catalog is password protected.

- The WORKCAT catalog will be used to define this WORKFILE (and to delete it at the end) since the catalog being reset can not be used during such operation. Any other catalog can be used. It must be connected to the master catalog.

The notes are continued on the next page.

DOS/VS notes:

- The WORKFILE's DLBL/EXTENT must specify the 'dname' (IDCUT1 is the default name in case the parameter WORKFILE (dname) is not specified), the 'volid' where the WORKFILE will be allocated (WTVS1R) and the CAT parameter indicating the catalog this volume belongs to (IJSYSWC). The DLBL/EXTENT information for the WORKCAT catalog must be provided.

7.12 EXPORT DISCONNECT USER CATALOG FROM MASTER CATALOG

DOS/VS example (input):

```
// JOB EXPORT DISCONNECT UCAT
// DLBL IJSYSUC,'PRIMER.UCAT1',,VSAM
// EXTENT SYS010,WTVSAM,1,0,19,57
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
      EXPORT PRIMER.UCAT1/UCATMRPW      -
            DISCONNECT
/&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0372 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
      EXPORT PRIMER.UCAT1/UCATMRPW      -
            DISCONNECT
/*
```

DOS/VS example (output):

```
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Notes:

- EXPORT DISCONNECT is required to signal the action to disconnect the user catalog PRIMER.UCAT1 (whose UPDATEPW or MASTERPW is required) from the master catalog.
- The user catalog volume need not to be mounted.

DOS/VS note:

- No DLBL/EXTENT statements are required.

MVS notes:

- The EXPORT DISCONNECT command removes the ALIASes for the user catalog from the master catalog (the IMPORT CONNECT command does not recatalog the ALIASes).

  Before issuing an EXPORT DISCONNECT command issue a LISTCAT VOLUMES command to list all ALIASes. After the IMPORT CONNECT command, each ALIAS must be entered into the master catalog with a separate DEFINE ALIAS command (see example on page 183).

7.13 IMPORT CONNECT A USER CATALOG TO THE MASTER CATALOG

DOS/VS example (input):

```
// JOB IMPORT CONNECT UCAT
// DLBL IJSYSUC,'PRIMER.UCAT1',,VSAM
// EXTENT SYS010,WTVSAM,1,0,19,57
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
         IMPORT   CONNECT                -
                  OBJECT (               -
                    PRIMER.UCAT1         -
                    DEVT (3330)          -
                    VOL (WTVSAM) )       -
                  CAT (PRIMER.MCAT/MCATMRPW)
/&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0382 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
         IMPORT CONNECT                  -
                OBJECTS (                -
                   PRIMER.UCAT1          -
                   DEVT (3330)           -
                   VOLUME (WTVSAM) )     -
                CAT (PRIMER.MCAT/MCATMRPW)
/*
```

DOS/VS example (output):

```
IDC0603I CONNECT UCAT PRIMER.UCAT1 SUCCESSFUL
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
IDC0603I CONNECT UCAT PRIMER.UCAT1 SUCCESSFUL
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Notes:

- IMPORT CONNECT must be specified to connect (or reconnect an already disconnected) user catalog to the master catalog.

  A user catalog may be connected to more than one master catalog but should only be used from one at a time.

  A user catalog cannot be connected to another user catalog.

- OBJECT names the user catalog PRIMER.UCAT1 indicating the device type and volume serial number of the volume containing the user catalog.

  With this information, the volume containing the user catalog need not be mounted.

- CAT names the master catalog PRIMER.MCAT and indicates its UPDATEPW or MASTERPW needed for the CONNECT operation.

DOS/VS note:

- No DLBL/EXTENT statements are required.

7.14 EXPORTRA

DOS/VS example (input):

```
// JOB EXPORT CRA
// DLBL CRAVOL1,,,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// TLBL SAVECRA,'SAVE.CRA.WTVSAM'
// ASSGN SYS005,TAPE,VOL=057454
// MTC REW,SYS005
// EXEC IDCAMS,SIZE=AUTO
    EXPORTRA CRA ((CRAVOL1 ALL))          -
             OUTFILE (SAVECRA             -
               ENV (BLKSZ (4096)          -
                    PDEV (2400))          -
                   )                      -
             MASTERPW (MCATMRPW)
/&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0392 EXEC PGM=IDCAMS
//EXPCRA   DD VOL=(,RETAIN,,,SER=057454),
//         UNIT=3400-6,LABEL=(5,SL),
//         DISP=(NEW,PASS),DSN=EXPCRA
//WTVSAM   DD UNIT=3330,VOL=SER=WTVSAM,
//         DISP=OLD,AMP='AMORG'
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
    EXPORTRA CRA ((WTVSAM ALL))          -
             OUTFILE (EXPCRA)
/*
```

DOS/VS example (output):

```
IDC0669I EXPORTING FROM CRA ON VOLUME WTVSAM
IDC0670I DS  EXP.
IDC0674I ** NAME IS CUSTOMER.R
IDC0670I DS  EXP.
IDC0674I ** NAME IS CUSTOMER.E
IDC0670I DS  EXP.
IDC0674I ** NAME IS CUSTOMER.K
IDC0670I DS  EXP.
IDC0674I ** NAME IS CITY.E
IDC0670I DS  EXP.
IDC0674I ** NAME IS SALESMAN.E
IDC0670I DS  EXP.
IDC0674I ** NAME IS CITY.K
IDC0670I DS  EXP.
IDC0674I ** NAME IS SALESMAN.K
IDC0676I PORT. DS CREATED  ON 10/20/77 AT 16:50:1
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
IDC0669I EXPORTING FROM CRA ON VOLUME WTVSAM
IDC0670I DS  EXP.
IDC0674I ** NAME IS CUSTOMER.R
IDC0670I DS  EXP.
IDC0674I ** NAME IS CUSTOMER.E
IDC0670I DS  EXP.
IDC0674I ** NAME IS CUSTOMER.K
IDC0670I DS  EXP.
IDC0674I ** NAME IS CITY.E
IDC0670I DS  EXP.
IDC0674I ** NAME IS SALESMAN.E
IDC0670I DS  EXP.
IDC0674I ** NAME IS SALESMAN.K
IDC0670I DS  EXP.
IDC0674I ** NAME IS CITY.K
IDC0676I PORT. DS CREATED  ON 12/21/77 AT 21:38:05
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

See notes on next page.

Notes:

- A description of EXPORTRA/IMPORTRA and their functions is included in section A.6.3 on page 259.
- The master password of the master catalog (MASTERPW) must be specified if the master catalog is password protected.
- There is no option in EXPORTRA to delete the exported objects from their catalog.
- As with RESETCAT, special attention should be paid when working with multivolume files. Read carefully the explanation of these subparameters in the Access Method Services manual:

  ALL | INFILE | NONE , FORCE | NOFORCE

  ALL means the data sets are exported using the CRA information and not the catalog information.
- Note that the user catalog itself is not exported.

DOS/VS notes:

- The DLBL/EXTENT for the CRA is used to point to the volume where the primary CRA resides. CRAVOL1 was used as a filename for the DLBL statement and is so coded in the EXPORTRA command.
- The tape (SAM) device used for unloading is defined in the ENVironment parameter.

7.15 IMPORTRA

DOS/VS example (input):

```
// JOB IMPORT CRA
// DLBL CRA,'WTVSAM',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// TLBL SAVECRA,'SAVE.CRA.WTVSAM'
// ASSGN SYS004,TAPE,VOL=057454
// MTC REW,SYS004
// EXEC IDCAMS,SIZE=AUTO
    IMPORTRA INFILE (SAVECRA                -
               ENV (BLKSZ (4096)            -
                    PDEV (2400))            -
                    )                       -
             OUTFILE (CRA)                  -
               CAT (PRIMER.UCAT1/UCATMRPW)
 /&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0402 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=OLD
//IMPCRA   DD VOL=(,RETAIN,,,SER=057454),
//         UNIT=3400-6,LABEL=(5,SL),
//         DISP=(OLD,PASS),DSN=EXPCRA
//WTDUMMY  DD UNIT=3330,VOL=SER=WTVSAM,DISP=OLD,
//         DSN=DUMMY.NAME,AMP='AMORG'
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
    IMPORTRA INFILE (IMPCRA)              -
             OUTFILE (WTDUMMY)            -
               CAT (PRIMER.UCAT1/UCATMRPW)
/*
```

DOS/VS example (output):

```
IDC0604I DS BEING IMP. WAS EXP. 10/20/77 AT 16:49
IDC0550I ENTRY (C) CUSTOMER.R DELETED
IDC0550I ENTRY (D) CUSTOMER.R.D DELETED
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0626I IMPORTRA SUCCEEDED FOR  CUSTOMER.R
IDC0604I DS BEING IMP. WAS EXP. 10/20/77 AT 16:49
IDC0550I ENTRY (R) SALES.CUST.E DELETED
IDC0550I ENTRY (G) SALESMAN.E DELETED
IDC0550I ENTRY (D) SALESMAN.E.D DELETED
IDC0550I ENTRY (I) SALESMAN.E.I DELETED
IDC0550I ENTRY (R) CITY.CUST.E DELETED
IDC0550I ENTRY (G) CITY.E DELETED
IDC0550I ENTRY (D) CITY.E.D DELETED
IDC0550I ENTRY (I) CITY.E.I DELETED
IDC0550I ENTRY (C) CUSTOMER.E DELETED
IDC0550I ENTRY (D) CUSTOMER.E.D DELETED
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0626I IMPORTRA SUCCEEDED FOR  CUSTOMER.E
```

OS/VS example (output):

```
IDC0604I DS BEING IMP. WAS EXP. 12/21/77 AT 21:37
IDC0550I ENTRY (D) CUSTOMER.R.D DELETED
IDC0550I ENTRY (C) CUSTOMER.R DELETED
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0626I IMPORTRA SUCCEEDED FOR  CUSTOMER.R
IDC0604I DS BEING IMP. WAS EXP. 12/21/77 AT 21:37
IDC0550I ENTRY (R) SALES.CUST.E DELETED
IDC0550I ENTRY (D) SALESMAN.E.D DELETED
IDC0550I ENTRY (I) SALESMAN.E.I DELETED
IDC0550I ENTRY (G) SALESMAN.E DELETED
IDC0550I ENTRY (R) CITY.CUST.E DELETED
IDC0550I ENTRY (D) CITY.E.D DELETED
IDC0550I ENTRY (I) CITY.E.I DELETED
IDC0550I ENTRY (G) CITY.E DELETED
IDC0550I ENTRY (D) CUSTOMER.E.D DELETED
IDC0550I ENTRY (C) CUSTOMER.E DELETED
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0626I IMPORTRA SUCCEEDED FOR  CUSTOMER.E
```

Output listing is continued on next page

```
IDC0604I DS BEING IMP. WAS EXP. 10/20/77 AT 16:49
IDC0550I ENTRY (R) SALES.CUST.K DELETED
IDC0550I ENTRY (G) SALESMAN.K DELETED
IDC0550I ENTRY (D) SALESMAN.K.D DELETED
IDC0550I ENTRY (I) SALESMAN.K.I DELETED
IDC0550I ENTRY (R) CITY.CUST.K DELETED
IDC0550I ENTRY (G) CITY.K DELETED
IDC0550I ENTRY (D) CITY.K.D DELETED
IDC0550I ENTRY (I) CITY.K.I DELETED
IDC0550I ENTRY (C) CUSTOMER.K DELETED
IDC0550I ENTRY (D) CUSTOMER.K.D DELETED
IDC0550I ENTRY (I) CUSTOMER.K.I DELETED
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0509I INDEX ALLOC.STAT. FOR VOLUME WTVSAM IS 0
IDC0626I IMPORTRA SUCCEEDED FOR  CUSTOMER.K
IDC0604I DS BEING IMP. WAS EXP. 10/20/77 AT 16:49
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0509I INDEX ALLOC.STAT. FOR VOLUME WTVSAM IS 0
IDC0626I IMPORTRA SUCCEEDED FOR  CITY.E
IDC0604I DS BEING IMP. WAS EXP. 10/20/77 AT 16:49
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0509I INDEX ALLOC.STAT. FOR VOLUME WTVSAM IS 0
IDC0626I IMPORTRA SUCCEEDED FOR  SALESMAN.E
IDC0604I DS BEING IMP. WAS EXP. 10/20/77 AT 16:49
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0509I INDEX ALLOC.STAT. FOR VOLUME WTVSAM IS 0
IDC0626I IMPORTRA SUCCEEDED FOR  CITY.K
IDC0604I DS BEING IMP. WAS EXP. 10/20/77 AT 16:50
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0509I INDEX ALLOC.STAT. FOR VOLUME WTVSAM IS 0
IDC0626I IMPORTRA SUCCEEDED FOR  SALESMAN.K
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

```
IDC0604I DS BEING IMP. WAS EXP. 12/21/77 AT 21:37
IDC0550I ENTRY (R) SALES.CUST.K DELETED
IDC0550I ENTRY (D) SALESMAN.K.D DELETED
IDC0550I ENTRY (I) SALESMAN.K.I DELETED
IDC0550I ENTRY (G) SALESMAN.K DELETED
IDC0550I ENTRY (R) CITY.CUST.K DELETED
IDC0550I ENTRY (D) CITY.K.D DELETED
IDC0550I ENTRY (I) CITY.K.I DELETED
IDC0550I ENTRY (G) CITY.K DELETED
IDC0550I ENTRY (D) CUSTOMER.K.D DELETED
IDC0550I ENTRY (I) CUSTOMER.K.I DELETED
IDC0550I ENTRY (C) CUSTOMER.K DELETED
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0509I INDEX ALLOC.STAT. FOR VOLUME WTVSAM IS 0
IDC0626I IMPORTRA SUCCEEDED FOR  CUSTOMER.K
IDC0604I DS BEING IMP. WAS EXP. 12/21/77 AT 21:37
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0509I INDEX ALLOC.STAT. FOR VOLUME WTVSAM IS 0
IDC0626I IMPORTRA SUCCEEDED FOR  CITY.E
IDC0604I DS BEING IMP. WAS EXP. 12/21/77 AT 21:38
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0509I INDEX ALLOC.STAT. FOR VOLUME WTVSAM IS 0
IDC0626I IMPORTRA SUCCEEDED FOR  SALESMAN.E
IDC0604I DS BEING IMP. WAS EXP. 12/21/77 AT 21:38
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0509I INDEX ALLOC.STAT. FOR VOLUME WTVSAM IS 0
IDC0626I IMPORTRA SUCCEEDED FOR  SALESMAN.K
IDC0604I DS BEING IMP. WAS EXP. 12/21/77 AT 21:38
IDC0520I CATALOG RECOVERY VOLUME IS WTVSAM
IDC0508I DATA ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0509I INDEX ALLOC.STAT. FOR VOLUME WTVSAM IS 0
IDC0626I IMPORTRA SUCCEEDED FOR  CITY.K
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Notes:

- A description of EXPORTRA/IMPORTRA and their functions is included in section A.6.3 on page 259.
- It is not possible to selectively restore VSAM objects by specifying them in the OBJECTS parameter.

DOS/VS notes:

- EXTENT specifies the 'volid' where the IMPORTRA will import the VSAM objects. This is indicated in the OUTFILE parameter, where CRA was chosen for the DLBL/EXTENT filename.

7.16 REPRO (UNLOAD - BACKUP A USER CATALOG

DOS/VS example (input):

```
// JOB REPRO UCAT TO TAPE
// DLBL IJSYSUC,'PRIMER.UCAT1',,VSAM
// EXTENT SYS010,WTVSAM,1,0,19,57
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// TLBL SAVUCAT,'SAVE.UCAT'
// ASSGN SYS005,TAPE,VOL=057454
// MTC REW,SYS005
// EXEC IDCAMS,SIZE=AUTO
     REPRO  INFILE (IJSYSUC/UCATUPDT)   -
                OUTFILE (SAVUCAT        -
                  ENV (                 -
                    BLKSZ (4096)        -
                    RECFM (VB)          -
                    PDEV (2400) )       -
                      )
/&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0412 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//CATIN    DD DSN=PRIMER.UCAT1,DISP=SHR
//SAVEUCAT DD VOL=(,RETAIN,,,SER=057454),
//         UNIT=3400-6,LABEL=(6,SL),
//         DISP=(NEW,PASS),DSN=SAVEUC,
//         DCB=(LRECL=516,RECFM=VB,BLKSIZE=5164)
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
           REPRO     INFILE (CATIN/UCATMRPW)-
                     OUTFILE (SAVEUCAT)
/*
```

DOS/VS example (output):

```
IDC0005I NUMBER OF RECORDS PROCESSED WAS 413
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
IDC0005I NUMBER OF RECORDS PROCESSED WAS 422
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Notes:

- The READPW, UPDATEPW or MASTERPW of the REPROduced object must be specified. In this case it is the UPDATEPW.
- Read also section A.5.1 on page 254.

DOS/VS notes:

- Access Method Services needs complete specifications of any nonVSAM data set.
- IJSYSUC was the name chosen for the DLBL/EXTENT filename to address the object to be copied as PRIMER.UCAT1 (file-ID) on volume WTVSAM at SYS010.

7.17 REPRO (RELOAD A USER CATALOG FROM BACKUP TAPE)

DOS/VS example (input):

```
// JOB REPRO UCAT FROM TAPE
// DLBL IJSYSUC,'PRIMER.UCAT1',,VSAM
// EXTENT SYS010,WTVSAM,1,0,19,57
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// TLBL SAVUCAT,'SAVE.UCAT'
// ASSGN SYS004,TAPE,VOL=057454
// MTC REW,SYS004
// EXEC IDCAMS,SIZE=AUTO
         REPRO     INFILE (SAVUCAT       -
                     ENV (               -
                       BLKSZ (4096)      -
                       PDEV (2400)       -
                       RECFM (VB) )      -
                         )               -
                   OUTFILE (IJSYSUC/UCATUPDT)
/&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0422 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//SAVEUCAT DD VOL=(,RETAIN,,,SER=057454),
//         UNIT=3400-6,LABEL=(6,SL),
//         DISP=(OLD,PASS),DSN=SAVEUC,
//         DCB=(LRECL=516,RECFM=VB,BLKSIZE=5164)
//CATOUT   DD DSN=PRIMER.UCAT1,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
         REPRO     INFILE (SAVEUCAT)      -
                   OUTFILE (CATOUT/UCATMRPW)
/*
```

DOS/VS example (output):

```
IDC0571I CATALOG RELOAD HAS BEEN INVOKED
IDC0005I NUMBER OF RECORDS PROCESSED WAS 413
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
IDC0571I CATALOG RELOAD HAS BEEN INVOKED
IDC0005I NUMBER OF RECORDS PROCESSED WAS 422
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

DOS/VS notes:

- Access Method Services needs complete specifications of any nonVSAM data set.
- The same consideration as in REPRO UNLOAD on page 171 applies.

7.18 DELETE

7.18.1 DELETE AIX

DELETE just one AIX and its related path(s).

DOS/VS example (input):

```
// JOB DEL AIX SALESMAN-CUSTOMER.K ON KSDS
// DLBL AIXK1,'SALESMAN.K',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
        DELETE    SALESMAN.K              -
                  ALTERNATEINDEX          -
                  FILE (AIXK1)            -
                  CAT (PRIMER.UCAT1/UCATMRPW)
 /&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0432 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//AIXK1    DD VOL=SER=WTVSAM,DISP=OLD,UNIT=3330
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
         DELETE    SALESMAN.K              -
                   ALTERNATEINDEX          -
                   FILE (AIXK1)            -
                   CAT (PRIMER.UCAT1/UCATMRPW)
/*
```

DOS/VS example (output):

```
IDC0550I ENTRY (R) SALES.CUST.K DELETED
IDC0550I ENTRY (G) SALESMAN.K DELETED
IDC0550I ENTRY (D) SALESMAN.K.D DELETED
IDC0550I ENTRY (I) SALESMAN.K.I DELETED
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
IDC0550I ENTRY (R) SALES.CUST.K DELETED
IDC0550I ENTRY (D) SALESMAN.K.D DELETED
IDC0550I ENTRY (I) SALESMAN.K.I DELETED
IDC0550I ENTRY (G) SALESMAN.K DELETED
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Notes:

- The FILE parameter is required (not for MVS, also see section 7.2.1 on page 88) to address the CRA (AIXK1). If the catalog were not recoverable , there would be no need for this parameter nor for the CRA volume to be mounted.
- The entry types (D, G, I, and R) are explained on page 51.

7.18.2 DELETE A KSDS CLUSTER (AND ITS AIX'S)

When a cluster is deleted, all its related objects (i.e., AIXs, PATHs) will automatically be deleted at the same time.

DOS/VS example (input):

```
// JOB DELETE KSDS
// DLBL KSDS,'CUSTOMER.K',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
        DELETE    CUSTOMER.K            -
                  FILE (KSDS)           -
                  CLUSTER               -
                  CAT (PRIMER.UCAT1/UCATMRPW)
/&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0442 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//KSDS     DD VOL=SER=WTVSAM,DISP=OLD,UNIT=3330
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
         DELETE    CUSTOMER.K            -
                   FILE (KSDS)           -
                   CLUSTER               -
                   CAT (PRIMER.UCAT1/UCATMRPW)
/*
```

DOS/VS example (output):

```
IDC0550I ENTRY (R) CITY.CUST.K DELETED
IDC0550I ENTRY (G) CITY.K DELETED
IDC0550I ENTRY (D) CITY.K.D DELETED
IDC0550I ENTRY (I) CITY.K.I DELETED
IDC0550I ENTRY (C) CUSTOMER.K DELETED
IDC0550I ENTRY (D) CUSTOMER.K.D DELETED
IDC0550I ENTRY (I) CUSTOMER.K.I DELETED
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
IDC0550I ENTRY (R) CITY.CUST.K DELETED
IDC0550I ENTRY (D) CITY.K.D DELETED
IDC0550I ENTRY (I) CITY.K.I DELETED
IDC0550I ENTRY (G) CITY.K DELETED
IDC0550I ENTRY (D) CUSTOMER.K.D DELETED
IDC0550I ENTRY (I) CUSTOMER.K.I DELETED
IDC0550I ENTRY (C) CUSTOMER.K DELETED
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Notes:

- The PURGE parameter need not to be specified, since no expiration date was specified at DEFINE time.
- ERASE can be specified to override the data component of an AIX or cluster to be deleted with binary zeros.
- The FILE parameter is required (not for MVS, also see section 7.2.1 on page 88) to address the CRA volume.
- The entry types (C, D, G, I, and R) are explained on page 51.

7.18.3 DELETE AN ESDS CLUSTER (AND ITS AIX'S)

DOS/VS example (input):

```
// JOB DELETE ESDS
// DLBL ESDS,'CUSTOMER.E',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
        DELETE    CUSTOMER.E              -
                  FILE (ESDS)             -
                  CLUSTER                 -
                  CAT (PRIMER.UCAT1/UCATMRPW)
/&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0452 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//ESDS     DD VOL=SER=WTVSAM,DISP=OLD,UNIT=3330
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
         DELETE    CUSTOMER.E              -
                   FILE (ESDS)             -
                   CLUSTER                 -
                   CAT (PRIMER.UCAT1/UCATMRPW)
/*
```

DOS/VS example (output):

```
IDC0550I ENTRY (R) CITY.CUST.E DELETED
IDC0550I ENTRY (G) CITY.E DELETED
IDC0550I ENTRY (D) CITY.E.D DELETED
IDC0550I ENTRY (I) CITY.E.I DELETED
IDC0550I ENTRY (R) SALES.CUST.E DELETED
IDC0550I ENTRY (G) SALESMAN.E DELETED
IDC0550I ENTRY (D) SALESMAN.E.D DELETED
IDC0550I ENTRY (I) SALESMAN.E.I DELETED
IDC0550I ENTRY (C) CUSTOMER.E DELETED
IDC0550I ENTRY (D) CUSTOMER.E.D DELETED
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
IDC0550I ENTRY (R) CITY.CUST.E DELETED
IDC0550I ENTRY (D) CITY.E.D DELETED
IDC0550I ENTRY (I) CITY.E.I DELETED
IDC0550I ENTRY (G) CITY.E DELETED
IDC0550I ENTRY (R) SALES.CUST.E DELETED
IDC0550I ENTRY (D) SALESMAN.E.D DELETED
IDC0550I ENTRY (I) SALESMAN.E.I DELETED
IDC0550I ENTRY (G) SALESMAN.E DELETED
IDC0550I ENTRY (D) CUSTOMER.E.D DELETED
IDC0550I ENTRY (C) CUSTOMER.E DELETED
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Notes:

- The FILE parameter is required (not for MVS, also see section 7.2.1 on page 88) to address the CRA volume.
- The entry types (C, D, G, I, and R) are explained on page 51.

7.18.4 DELETE AN RRDS CLUSTER

DOS/VS example (input):

```
// JOB DELETE RRDS
// DLBL RRDS,'CUSTOMER.R',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
        DELETE    CUSTOMER.R            -
                  FILE (RRDS)           -
                  CLUSTER               -
                  CAT (PRIMER.UCAT1/UCATMRPW)
 /&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0462 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//RRDS     DD VOL=SER=WTVSAM,DISP=OLD,UNIT=3330
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
         DELETE    CUSTOMER.R            -
                   FILE (RRDS)           -
                   CLUSTER               -
                   CAT (PRIMER.UCAT1/UCATMRPW)
/*
```

DOS/VS example (output):

```
IDC0550I ENTRY (C) CUSTOMER.R DELETED
IDC0550I ENTRY (D) CUSTOMER.R.D DELETED
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
IDC0550I ENTRY (D) CUSTOMER.R.D DELETED
IDC0550I ENTRY (C) CUSTOMER.R DELETED
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Notes:

- The FILE parameter is required (not for MVS, also see section 7.2.1 on page 88) to address the CRA volume.
- The entry types (C and D) are explained on page 51.

7.18.5 DELETE SPACE

DOS/VS example (input):

```
// JOB DELETE SPACE
// DLBL SPACE,'VSAM.SPACE',,VSAM
// EXTENT SYS010,WTVSAM,,,114,1900
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
        DELETE    (WTVSAM) SPACE          -
                  FILE (SPACE)            -
                  CAT (PRIMER.UCAT1/UCATMRPW)
 /&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIMO472 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//SPACE    DD VOL=SER=WTVSAM,DISP=OLD,UNIT=3330
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
         DELETE    (WTVSAM) SPACE          -
                   FILE (SPACE)            -
                   CAT (PRIMER.UCAT1/UCATMRPW)
/*
```

DOS/VS example (output):

```
IDC0555I DELETE OF SPACE DID NOT CAUSE WTVSAM
         TO BE DELETED
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
IDC0555I DELETE OF SPACE DID NOT CAUSE WTVSAM
         TO BE DELETED
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Notes:

- '(WTVSAM) SPACE' indicates that all the VSAM spaces on volume WTVSAM will be deleted provided they are empty. If nonempty VSAM data spaces should be scratched from the VTOC and VSAM volume ownership should be given up, the FORCE parameter may be specified. FORCE cannot be specified, if the volume contains a VSAM catalog.

- The FILE parameter is required (not for MVS, also see section 7.2.1 on page 88) to address the CRA volume and to mount the volume (also for nonrecoverable catalogs) to change the VTOC (delete F1-DSCBs describing the VSAM data spaces).

- The message IDC0555I means, that some VSAM space is still allocated on the volume WTVSAM, i.e., the catalog itself.

7.18.6 DELETE USER CATALOG

DOS/VS example (input):

```
// JOB DELETE UCAT
// DLBL IJSYSUC,'PRIMER.UCAT1',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
      DELETE    (PRIMER.UCAT1/UCATMRPW) -
                UCAT
/&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIMO482 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
         DELETE    (PRIMER.UCAT1/UCATMRPW)  -
                   UCAT                     -
                   FORCE
                                        /*
```

DOS/VS example (output):

```
IDC0550I ENTRY (U) PRIMER.UCAT1 DELETED
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

OS/VS example (output):

```
IDC0550I ENTRY (U) PRIMER.UCAT1 DELETED
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Notes:

- A user catalog can only be deleted if it is empty and is not in use by any other partition/region (in OS/VS the FORCE parameter can be specified to delete a non-empty catalog provided it is not opened by any other partition/region).
- The master catalog connector entry pointing to that user catalog is also deleted.
- The entry type 'U' specifies a user catalog entry. All catalog entry types are explained on page 51.

OS/VS note:

- The FORCE parameter is not needed since the catalog is empty.

DOS/VS notes:

- The user catalog to be deleted must be the current 'job catalog'. This is indicated by the DLBL/EXTENT/ASSGN statements with the filename=IJSYSUC and file-ID=PRIMER.UCAT1.
- The FORCE parameter is not allowed for catalogs and is not needed since the catalog is empty.

7.19 VOLUME CLEANUP

7.19.1 VOLUME CLEANUP DOS/VS (UTILITY IKQVDU)

A VSAM service aid has been provided to assist the system programmer in maintaining the VTOC and VOL1 labels on DASD devices.

This aid is described together with other VSAM aids in the DOS/VS HANDBOOK part 2 (SY33-8572) chapter 3 section SERVICE AIDS or in the DOS/VS VSAM LOGIC (SY33-8562).

It should be used with great caution since it is possible to modify and even scratch all or part of the DASD VTOC.

All VSAM aids can be installed by executing the following job :

```
// JOB CATAL IKQVEDA IKQVDU $$BCVS04
// OPTION CATAL
  INCLUDE IKQCLNLK
// EXEC LNKEDT
/&
```

To execute IKQVDU, the following job should be executed:

```
// JOB IKQVDU
// ASSGN SYS000,DISK,VOL=WTVSAM
// EXEC IKQVDU,SIZE=AUTO
```

The next page contains an example of executing IKQVDU.

The following is an example of executing IKQVDU using the system typewriter:

```
 SPECIFY FUNCTION OR
 REPLY  '?'  FOR OPTIONS
 READY
?
 TO SET THE VOLUME OWNERSHIP FLAG REPLY 'SET OWNERSHIP'
 TO SET THE CRA POINTER REPLY 'SET OWNERSHIP'
 TO RESET THE VOLUME OWNERSHIP FLAG AND CRA POINTER REPLY 'RESET OWNERSHIP' OR 'RESET CRA'
 TO SET THE SECURITY FLAG IN A F1 DSCB REPLY 'SET SECURITY'
 TO RESET THE SECURITY FLAG IN A F1 DSCB REPLY 'RESET SECURITY'
 TO REMOVE A DSCB FROM THE VTOC REPLY 'SCRATCH'
 TO RENAME A DSCB REPLY 'RENAME'
 TO ALLOCATE A DSCB REPLY 'ALLOCATE'
 TO REINITIATE PROCESSING REPLY 'RESTART'
 TO ALTER OR DISPLAY A DASD VOL1 LABEL
 REPLY 'CLIP LABEL=SER=N..N' OR 'CLIP LABEL=DISPLAY'
 TO TERMINATE PROCESSING REPLY 'END'
 READY
SCRATCH DSN
 ENTER DSN
Z9999994.VSAMDSPC.T8B8486E.T3293420
 SCRATCH SECURITY PROTECTED CATALOG?    Z9999994.VSAMDSPC.T8B8486E.T3293420
 REPLY YES/NO
YES
 READY
SCRATCH DSN
 ENTER DSN
Z9999992.VSAMDSPC.T8B84870.T54A8380
 SCRATCH SECURITY PROTECTED DATA SPACE? Z9999992.VSAMDSPC.T8B84870.T54A8380
 REPLY YES/NO
YES
 READY
RESET OWNERSHIP
 VOLUME OWNERSHIP FLAG AND CRA POINTER RESET
 READY
END
```

Explanation of entered commands (start in column 1):

?	= all functions of IKQVDU are to be listed
SCRATCH DSN	= a data set (in this case a VSAM catalog and a VSAM space) is to be scratched.
Z9999994. ...	= the (VTOC) data set name of the VSAM catalog
Z9999992. ...	= the (VTOC) data set name of the VSAM space
RESET OWNERSHIP	= remove VSAM ownership after scratching all VSAM data sets
END	= terminate utility

7.19.2 VOLUME CLEANUP OS/VS (1/2) (ALTER REMOVEVOLUMES)

This step is only necessary, if the volume to contain the user catalog or any VSAM object (as first VSAM object on that volume) is owned by a unknown or inaccessible VSAM catalog. If there are problems in defining a VSAM user catalog, or a VSAM data space on a volume without a catalog, this step can be executed to delete all VSAM information from a disk, without accessing any VSAM catalog.

Attention:

This function destroys VSAM catalogs/data sets on that volume.

The VSAM master catalog name and its master password (if any) must be supplied as data set name. If the master catalog is not known, a LISTCAT command without any parameter should be issued. If, however the VSAM master catalog is password protected, this password will not be printed on the LISTCAT output.

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0492 EXEC PGM=IDCAMS
//WTVSAM   DD VOL=SER=WTVSAM,DISP=OLD,UNIT=3330
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
              REMOVEVOLUMES(WTVSAM)       -
              MASTERPW(MCATMRPW)          -
              FILE(WTVSAM)
/*
```

Output of Access Method Services:

```
IDC0526I ALTERED ALLOC. STAT. FOR VOLUME WTVSAM IS 0
IDC0531I ENTRY PRIMER.MCAT ALTERED
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

7.19.3 VOLUME CLEANUP OS/VS (2/2) (EXPORT DISCONNECT)

If a user catalog, still cataloged in the VSAM master catalog, was on the old volume, this user catalog entry must be removed from the master catalog with the following job.

If the user catalog is still not cataloged in the master catalog, the user catalog can now be defined (see previous section 7.4 on page 102).

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0374 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
       EXPORT PRIMER.UCAT1/UCATMRPW      -
              DISCONNECT
/*
```

- Output of Access Method Services:

```
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

7.20 DEFINE NONVSAM (OS/VS)

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0582 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT2,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
         DEFINE    NONVSAM                -
                   (NAME(LOADKSDS)        -
                   DEVT(3330)             -
                   VOL(WTVS1R) )          -
                   CAT (PRIMER.UCAT2/UCATMRPW)
/*
```

- Output of Access Method Services:

```
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Note: This object was defined in VSAM catalog PRIMER.UCAT2. The definition of this catalog is not included in this manual.

7.21 DEFINE ALIAS (MVS)

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIM0592 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT2,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
         DEFINE    ALIAS                  -
                   (NAME(LOAD)            -
                   RELATE (LOADKSDS) )    -
                   CAT (PRIMER.UCAT2/UCATMRPW)
/*
```

- Output of Access Method Services:

```
IDC0001I FUNCTION COMPL., HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC. COMPL. MAX. COND.CODE WAS 0
```

Note: This object was defined in VSAM catalog PRIMER.UCAT2. The definition of this catalog is not included in this manual.

8.0 MISCELLANEOUS

8.1 PERFORMANCE

8.1.1 CHOOSING THE DATA CI-SIZE

- Space utilization

  A given logical record size may fit some CI sizes better than others. Generally, large CI sizes provide better fits. Also some CI sizes fit a track of a given device better than others. For example, on a IBM 3340 track a 512 byte CI yields a potential 6144 bytes (12 CIs), which is 67% of usable space per track (see section 8.2) whereas a 4096 byte CI yields 8192 bytes (2 CIs) of data on a IBM 3340 track, which is 94% of usable space. Assuming a 80 byte record; in one case there would be 72 records per track and in the other case there would be 102 records per track.

- Direct processing

  If direct processing is the predominant manner of accessing the data, then a choice of a small data CI is preferable. In general, select the smallest data CI that yields a reasonable space utilization. Normally 1024 or 2048 byte CIs are good, but on a device with large cylinder and track capacities like the IBM 3350 device, the index CI-size will be 4 K when using 1 K data CIs (see page 198).

- Sequential processing

  If the processing is predominantly sequential, even larger data CIs may be a good choice.

- Data CI size selection:

  Figure 7 represents a summary of generally applicable choices. There are conditions where the next smaller or next larger CI than is recommended is actually a better choice based upon space utilization, index CI size considerations or CPU storage capabilities.

Accessing pattern	Condition or device	Recommended data CI size
Direct	3340	1024
	2314	1024 or 2048
	3330, 3350	4096
Sequential	2314, 3330	4096 or 6144
	3340	4096 or 8192
	3350	4096, 6144 or 8192
Skip sequential	<2 records referenced per track	Choose CI size as for Direct access
	otherwise	4096

Figure 42. VSAM data CI size recommendations

8.1.2 CHOOSING THE INDEX CI SIZE (KSDS DATA SETS)

If no index CI value is defined for a KSDS data set (which is suggested) VSAM chooses the minimum index CI based on the calculation as explained in section 8.3. The VSAM chosen index CI-size may be checked using the LISTCAT command (the fields are explained on page 158).

This size can be increased by defining an index CI up to 4096. This definition may help to have fewer index levels, as the highest level index CI can hold a maximum of about 502 entries (CI size = 4096).

Using the index formula described in section 8.7 on page 206) to calculate the CI size for 502 entries would result in 4092 bytes. A data set, which is smaller than 502 cylinders (assumed 1 CA = 1 cyl), has then only two index levels (sequence set + 1 index-set CI). The disadvantage however, is the huge amount of buffer space needed to hold the index CIs (for buffers see section 8.1.5 on page 190). Page 195 contains a table with the average number of entries for the 4 index CI sizes.

The size of an index CI also affects the replication factor (when REPLICATION or IMBED is used) and thus affects performance.

Notes:

If you specify an index CI size which is too small to hold all necessary entries for all CIs of one CA, the CA's are never filled to their maximum capacity and no message is issued by VSAM to indicate this condition. Then it is better to let VSAM choose the index CI size. After DEFINE, a LISTCAT command can be issued and the entries be checked to redefine the cluster with a different data CA size if the index record is too big.

If an index CI size is specified in a DEFINE command this size is checked based on the specified/default data CA size, and may be overridden without notice.

8.1.3 DATA SET SPACE ALLOCATION

8.1.3.1 SMALL DATA SETS

For data sets less than one cylinder in size it is more advantageous for space utilisation considerations (but not for performance) to specify the maximum number of tracks required in the primary allocation of the data component, one track for the non-imbedded sequence set index, and no secondary definition for either data or index. The allocations for this data set should be set so that only 1 index record is allocated. This is possible if only one CA is allocated for the data component (see also next section).

8.1.3.2 MULTIPLE CYLINDER DATA SETS (NOT MULTIVOLUME)

It is usually best to calculate the number of cylinders needed for data in a newly created data set and specify this amount in cylinders for the primary allocation of the data component (a calculation example is included in section 8.3 on page 197).

Make the secondary allocation equal or greater than one cylinder but less than the primary allocation.

VSAM calculates the CA size based on primary and secondary allocations.

- If primary or secondary allocation is smaller than one cylinder, the smaller value (rounded up to full tracks in case of RECORDS) is used as CA size.
- If primary and secondary allocation is larger than one cylinder, the CA size is one cylinder (maximum CA size).

8.1.3.3 MULTIVOLUME DATA SETS

When defining multivolume data sets, the following rules must be considered:

1. For suballocatable multivolume data sets a VSAM data space must exist on all volumes specified in the VOLUMES parameter.

2. For unique multivolume data sets all volumes which are specified in the VOLUMES parameter (except the first volume in the list), must already be owned by the catalog, in which the multivolume data set is to be cataloged.

3. The primary or secondary allocation value must not exceed the maximum capacity of one volume.

Explanation of the rules:

Rule 1:

- Suballocated data sets

  Before defining the multivolume data set, a DEFINE SPACE command must have been issued for all volumes defined in the VOLUMES parameter of the multivolume data set. When there is not enough space in a VSAM data space for a secondary allocation, the data space may be extended (not in DOS/VS) (see section 8.1.3.4).

- UNIQUE data sets

  Check if all volumes of the VOLUMES parameter (except the first) belong to the catalog, in which the data set is to be cataloged (for DOS/VS see description and notes in section 2.9 on page 40). If not, issue a DEFINE SPACE CANDIDATE command for all volumes not belonging to that catalog (in DOS/VS DEFINE SPACE CANDIDATE is not allowed when using a recoverable catalog). After correct operation, define the multivolume data set.

  Note: All volumes specified in the DEFINE SPACE CANDIDATE command must be mounted, since the VSAM ownership bit in the F4-DSCB is set ON.

  In OS/VS systems when using recoverable catalogs also a F1-DSCB describing the one cylinder CRA is written into the VTOC of each volume when the command is executed.

Rule 2:

VSAM allocates the primary space on the first volume at DEFINE-time for non-Keyrange data sets. When loading the data set and the primary space is filled, the secondary amount of space is allocated on the first volume as long as there is enough space available. If there is no more space for secondary allocations on the first volume, VSAM allocates primary space on the second volume. Then secondary space is allocated on volume 2, etc.

Generally, VSAM always allocates primary space on a new volume first and continues with secondary allocations. This is important for the last volume. When, for example the space allocation was CYL (200,50), then 200 cylinders are allocated, even when only 20 cylinders are needed. In this case, the specification CYL (50,200) would fit better.

8.1.3.4 DATA SPACE ALLOCATION AND EXTENSION

- DOS/VS

  In DOS/VS the location and size of a VSAM data space or of a UNIQUE data set must be specified in an EXTENT statement.

  DOS/VS does not allow secondary space allocations for VSAM data spaces or UNIQUE data sets. Therefore the use of UNIQUE data set should be limited to specific applications.

- VS1, SVS and MVS

  The location of a VSAM data space or of a UNIQUE data set cannot be specified with OS/VS JCL. The size is specified in the DEFINE SPACE or DEFINE CLUSTER ... UNIQUE command.

  When VSAM recognizes that a data space or a UNIQUE data set must be allocated or extended (secondary allocation), OS/DADSM is automatically called to allocate the requested space. A data space can only be extended on one volume. When a multivolume UNIQUE data set has to be extended to the next volume (when there is not enough space on the first one) OS/DADSM will perform the allocation.

  When VSAM tries to allocate secondary space for a suballocated data set and there is not enough room in the VSAM data space, VSAM , and in sequence OS/DADSM allocates an amount of space either in data space secondary allocation size, or, if this is not enough, in the requested data set secondary allocation size. A data space, however can only allocate secondary extents if a secondary value was specified in the DEFINE SPACE command.

8.1.4 INDEX PERFORMANCE CONSIDERATIONS

To improve performance during processing, each sequence set index record may be stored with the CA it points to (by using the definition parameter IMBED, see section 7.2.3 on page 89). This generally eliminates disk-arm movement because it is not necessary to do separate seeks to locate both the sequence index record and the data record (if the arm has not been moved by other activities between index and data reference).

When the imbedded sequence set record is then duplicated on the first track of its CA as many times as it will fit to reduce the rotational delay. A data CI can never reside on the same physical track as the index CI. Most of the index (index sequence set records and all higher level index records) can reside in virtual storage if enough buffer storage is specified by the user, but VSAM does not preload index buffer(s). Depending on the type of access index buffers are assigned and used differently (for further information see section 8.1.5 starting on page 190).

8.1.5 VSAM BUFFERS (NON-SHARED RESOURCES) AND STRINGS

Non-shared resources is the standard usage of VSAM buffers for one data set only.

VSAM buffers are used by VSAM to read/write CIs from/to DASD.

To increase performance, there are parameters (three for KSDS and two for ESDS and RRDS), to override the VSAM default values:

- BUFNI - number of index buffers (default 1) (KSDS only)
- BUFND - number of data buffers (default 2 (DOS/VS read-only = 1))
- BUFSP - amount of virtual storage to be reserved for VSAM buffers, when opening the data set (default = 2 data buffer + 1 index buffer)

BUFND and BUFNI are ACB parameters (see figures 47, 48 on page 212, 215) and in OS/VS systems also JCL parameters (see description starting on page 213 on how to specify these parameter in the JCL statement). BUFSP is a DEFINE, ACB and JCL parameter (the ACB macro instruction is described in section 8.11.1.1 on page 217). Any combination of these parameters can be used.

If a parameter is specified in JCL, this value overrides any previous specification of this parameter (ACB or DEFINE), provided it is a larger value (DOS/VS). In OS/VS the JCL overrides the ACB and catalog values even if it is smaller.

If one of the parameters is specified in the ACB macro instruction (in the user program) this value overrides the default values, or in case of BUFSP the DEFINE value, even if it is a smaller value.

- Data buffers (BUFND).

  To calculate the number of data buffers, the number of strings used must be determined. A string is a request to a VSAM data set requiring data set positioning. VSAM stores e.g. for a sequential access the information about which record has been accessed. If different concurrent accesses to the same data set are necessary multiple strings are used. For multiple string buffers see also page 194.

  It is possible to work with multiple strings (ACB parameter STRNO=x). Multiple strings allow, for example, access to the data set directly with one string and sequentially with the second string. For each string there must be at least one buffer.

  OS/VS:
  Usually the user defines the amount of strings he intends to use in his program (ACB STRNO=x) (see also section 8.11.4 on page 223 and figures 47, 48 on pages 212,215). But when all strings are active and there is another request waiting, a new string is built dynamically. Dynamic string addition does not apply to shared resources as used in CICS and IMS. The string is not erased, when it is not used any more, but it can be reused for the next request.

  The minimum number of data buffers is two (1 per string + 1 for splits). It is very important to know that the OS/VS user can use only one of these two buffers. One buffer is always reserved by VSAM for splits. If 4 buffers should be used, 5 must be specified.

  DOS/VS:
  If in DOS/VS BUFND=2 is specified, the user has access to 2 buffers (and not just to 1 as in OS/VS).

- Index buffers (BUFNI).

  The minimum number of index buffers is one. As described later, it may be advisable to have more index buffers for extensive index operations (direct). If multiple strings are used, the minimum needed is one index buffer per string.

- Buffer space (BUFSP).

  This value specifies the amount of virtual storage, which must be available when opening the data set. If more buffer space is defined than used for default buffers (2 (DOS/VS = 1 if read-only) data and 1 index buffer) or specified by BUFND plus BUFNI, VSAM allocates more buffers in this area as determined by its algorithms.

Note that the allocation of buffers by VSAM is based upon the access mode indicated in the ACB at "OPEN" time; this may result in inefficient buffer usage if the ACB is opened in one form (sequential or direct) but the majority of processing is in the other).

8.1.5.1 BUFFERS FOR DIRECT ACCESS

Generally, the more index buffers, the better performance that can be obtained. Data buffers are not important, because only one is used for each access.

As there is always a top down search through the index, at least two index buffers should be specified, to hold the highest index CI in the buffer.

If more than one index buffer is specified, the additional buffers can hold index set CIs but only one index buffer is used for the sequence set per string.

If too few index buffers are specified, the result may be poor performance, as extensive EXCPs may have to be issued. If there are three index levels and only one index buffer, three index EXCPs are necessary to find the pointer to the correct data CI.

Suggested number of buffers:

```
BUFNI: minimum = no. of index levels + 1 * STRNO (default = 1)
       maximum = (no. of records (CI's) in index set + 1) * STRNO
BUFND: default = 2 (STRNO value + 1 for splits)
       (DOS/VS read-only = 1)
```

The default BUFND is STRNO + 1, in turn STRNO's default is 1.

8.1.5.2 BUFFERS FOR SKIP SEQUENTIAL ACCESS

There is no good way to predict the number of buffers required, when using skip sequential access, as only 1 data buffer is used (as for direct processing) and only 1 index buffer is used for the sequence set level. The suggestion below is a starting suggestion and should be modified by experience.

There is no look ahead for index sequence set CIs, so there may be poor performance if the keys to access are not adjacent.

Suggested number of buffers:

```
BUFNI : default = 1
BUFND : default = 2 (DOS/VS read-only = 1)
```

If the keys to be processed are grouped, that is concentrated on one or more places in the data set, the approach for sequential access can be followed (see next section).

8.1.5.3 BUFFERS FOR SEQUENTIAL ACCESS

Generally, the more data buffers, the better performance that can be obtained.

Index Buffers DOS/VS:

It may be sometimes advantageous to have 2 index buffers, especially if IMBED is not used. DOS/VS VSAM will read ahead on the next sequence set if 2 buffers are available.

Index buffers OS/VS:

For keyed sequential access in a KSDS only one index buffer (for the sequence set record) is used.

Data buffers:

If multiple data buffers are specified. VSAM uses a so called read-ahead function. The next data control intervals are read in before they are needed.

The following figure demonstrates how VSAM decreases the number of EXCPs when reading 1000 CIs with a different number of data buffers (remember in OS/VS 1 data buffer is not used). BUFNI = 1 (default).

DOS/VS			OS/VS	
BUFND	number of EXCPs	I/O overlapped with processing	BUFND	number of EXCPs
1	1000	No	2	1000
2	500	No	3	500
3	334	No	4	334
4	500	Yes	5	334
5	334	Yes	6	225
6	334	Yes	7	225
7	250	Yes	8	200

The difference between the number of EXCPs (or I/O operations) for DOS/VS and OS/VS is because of different way of buffer usage.

Suggested number of buffers:

```
BUFNI = 1  (or STRNO (number of strings))
BUFND = >4 (for multiple strings see section 8.1.5.4)
```

8.1.5.4 BUFFERS FOR MULTIPLE STRINGS

- Index buffer

  Index buffers with index set records (CIs) are used for all strings. Index Sequence Set records are handled as follows:

  - records are read and used separately for each string, even when another string already read it into a buffer.

  - DOS/VS: Records are reread only if the needed CI is not already in a buffer (except with SHAREOPTION 4).

- Direct processing (data buffer)

  As described on page 191 there must be at least one buffer per string plus one buffer for splits. If STRNO=2 is specified, three buffers are allocated (VSAM default).

- Sequential processing (data buffer)

  If multiple strings are used for sequential processing, the following must be considered:

  DOS/VS:

  - There is no buffer reserved for splits

  - There is no buffer allocated to each string, but rather buffers are attached to the last string. In DOS/VS a string doing sequential processing will repeatedly try to get a surplus buffer from strings not actively doing sequential processing.

  OS/VS:

  - one buffer is used for VSAM splits

  - one buffer is automatically allocated per string

  - The first sequential request is allocated all remaining buffers. These buffers are kept for this string until an ENDREQ (end request) macro (see description in section 8.11 starting on page 217 is issued, even when they are not used. A second string in this case has no read-ahead capability as there is no extra free buffer available.

8.2 VSAM SIZES FOR DOS/VS AND OS/VS (CI-, CA-SIZES ETC.)

- Catalog

  - Minimum size on 2305, 3330, 3350 = 9 trks
  - Minimum size on 2314, 3340 = 15 trks
  - Maximum number of extents (data component) = 13

- Control area (CA)

  - Minimum size 1 track max. = 1 cyl
    (so the actual size is device dependent)

- Control interval (CI)

  - Data component (minimum 512 bytes) max. = 32768 bytes
    (in increments of 512 bytes up to 8 K,
    then in increments of 2 K up to maximum)
  - Index component (minimum 512 bytes) max. = 4096 bytes
    (only sizes 512, 1024, 2048, 4096)

- CRA (recoverable catalog only)

  - Minimum size = 1 cyl
  - Maximum size = 16 cyl

- Data set

  - Maximum size of a multivolume data set = 2^{32} bytes
    (2^{32} = 4 294 967 296 bytes)
  - Maximum number of VSAM extents = 123
  - Maximum number of AIX per base cluster = 253
  - Maximum no. extents/volume (UNIQUE) (OS/VS) = 16

- Data space

  - Maximum size = 1 vol
  - Maximum number of extents (OS/VS) = 16

- Index control interval number of entries:

  An index control interval may hold the following number of entries (8 bytes/entry based on normal key compression):

CI Size	Number of entries
512	58
1024	121
2048	248
4096	502

- Key length

  - maximum key length = 255 bytes
  - used key length Keyranges max. = 64 bytes (0-63)

- Logical record length:

  - Non-spanned (max. CI-size minus 7) max. = 32761 bytes
    (1 RDF + 1 CIDF = 7 bytes)

  - Spanned
    (max. 1 CA-size minus 10 * number CI/CA)
    (10 = 2 RDFs plus 1 CIDF) max. = < 1 cyl

- Physical record (block) sizes (no DEFINE-option)

  - 4 sizes: 512, 1024, 2048 and (3330,3340,3350) 4096 bytes

Each control interval is automatically divided by VSAM into an integral number of physical blocks (e.g. CI = 512 results in physical block = 512, CI = 1536 results in physical block = 512, so 3 blocks are needed to hold one CI; CI = 6144 results in physical block = 2048, so 3 blocks are needed (3330 and 3350 only, 3340 would be set to 1K), etc.). For further information see also page 9.

- Device characteristics

device characteristics			number of physical blocks/track			
device	trk/cyl	cyl/vol	512	1024	2048	4096
3330-1	19	404	20	11	6	3
3330-11	19	808	20	11	6	3
3340-35	12	348	12	7	*	2
3340-70	12	696	12	7	*	2
3350	30	555	27	15	8	4

* 2K is not used for 3340 (would result in only 75% track utilization)

Example: On a 3340, 2 physical blocks with a length of 4096 can be stored on one track, but only 7 physical blocks with each 1024 bytes.

8.3 HOW TO CALCULATE A VSAM KSDS DATA SET

This example shows how to calculate the following values for the KSDS CUSTOMER.K (see section 7.6.1 on page 106).

- Size of data component
- Size of index component
- Minimum size of index control interval

The following is assumed for the calculations:

-	Device type is	=	3330
-	Control area size (CA)	=	1 cyl
-	Data control interval size (CI)	=	1024 bytes
-	Physical blocksize (calculated by VSAM)	=	1024 bytes
-	Record size	=	200 bytes
-	Key length	=	5 bytes
-	Free space definition control interval	=	20 %
-	Free space definition control area	=	10 %
-	Number of records to be loaded	=	3000
-	Index options defined	=	IMBED/ REPLICATE

The calculations are shown on the next page.

Based on the previous specifications the following can be calculated (see also notes on the following page):

- Data component:

```
(1) number maximum records/CI ((1024-10)/200)    =     5
    min.free bytes (20% * CIsize)(see page 88) =   204
    number loaded records (1024-10-204) / 200    =     4
(2) number physical blocks/track                 =    11
(3) number max. CI/CA (11*18)                    =   198
     (1 cyl=19 trks - 1 track for index seq.set)
    number loaded CI/CA (198-10% freespace)      =   178
    number loaded records/cyl (4*178)            =   712
```

Total space for data component (3000/712) (rounded) = 5 cyl

- Index component

sequence set: (included in data component) = 0 trks

index set:

calculation for one sequence set index entry (1 entry per data CI):

```
(4)   key compressed                          = 3
(5)   control bytes                           = 2
(6)   pointer to a data CI in a CA            = 1

      total bytes per seq. set or index set entry =     6

(7) number index entr. per index CI (data CI/CA)=   198
    length index entr. per index CI (198*6)     =  1188
    control fields/index CI(header,RDF,CIDF)    =    31
    total (minimum) length/index CI bytes       =  1219
```

Minimum index CI-size (see CI sizes page 195) = 2048

```
(8) number of different index CI records in
    sequence set (included in data component)   =     5
    number of index CIs index set               =     1
    space for index set (1 CI/track (REPLICATE))=     1 trk
```

Total space for index component = 1 trk

Notes:

(1) The value (1024-10) is CI length minus 10 bytes for 2 RDFs and 1 CIDF for fixed length records. For variable length records 19 bytes (for 5 RDFs and 1 CIDF) are needed, but calculations are not affected.

(2) On IBM 3330, eleven physical blocks with 1024 bytes can be stored on one track (see physical block sizes on page 196).

(3) The value (11*18) is the number of physical blocks per track multiplied by the number of data tracks per cylinder (19 minus 1 for the sequence set control interval (IMBED)).

(4) For index calculations 3 bytes can be used as an average compressed key length. This average value is almost independent of the actual key length and is also valid for longer keys. Only in very unusual key structures, this value may be higher.

As a guidance if keys are compressed well, smaller keys (5-15 bytes) can compress to 3-5 bytes and longer. Keys (20-30 bytes) can compress to 6-7 bytes.

Index key compression is explained in section 8.6 on page 203. The formula for the index control interval size is explained in section 8.7 on page 206.

(5) The control fields have a fixed length of 2 bytes (used for the key compression).

(6) The length of the pointer (a control interval number) is usually 1 byte for the sequence set and 2 bytes for all higher index levels (the index set).

(7) For calculation of the minimum index CI size, the maximum number of entries per CA must be used.

(8) On IBM 3330 six physical blocks with the length of 2048 bytes can be stored on one track (see physical block sizes on page 196). See also index CI size formula explained on page 206.

One index CI of the sequence set addresses one CA. Since in our case 1 CA = 1 cylinder and the size of the KSDS is 5 cylinders, 5 index CI sequence set records are required. They are imbedded, that is replicated on the first track of the corresponding CA and do not occupy space in the index component itself. Then one index set CI would be enough to accommodate 5 entries. The minimum allocation of 1 track for an index CA (6 CI per CA) was chosen by VSAM.

8.4 LOADING A KSDS WITH DIFFERENT FREE SPACE

As discussed in section 7.2.2 on page 88, free space to be held free for later insertions can be defined in the DEFINE command. This value can be changed with the ALTER command. The following discussion shows an example using these two commands.

Assuming a large KSDS data set is to contain records with keys from 1 - 300000. It is expected to have no inserts in Keyrange 1 - 100000, some inserts in Keyrange 100001 - 200000 and heavy inserts in Keyrange 200001 - 300000.

An ideal data structure at loading time would be for example:

- Keyrange 1 - 100000 no free space
- Keyrange 100001 - 200000 5% CA free space
- Keyrange 200001 - 300000 5% CI and 20% CA free space

This structure can be built as follows:

1. DEFINE CLUSTER with FREESPACE (0 0), or without any FREESPACE parameter.

2. Load records 1 - 100000 with REPRO or any user program using sequential insertion technique.

3. CLOSE the data set.

4. Change the FREESPACE value of the cluster with the Access Method Services command 'ALTER clustername FREESPACE (0 5)'.

5. Load records 100001 - 200000 with REPRO or any user program using sequential insertion technique.

6. CLOSE the data set.

7. Change the FREESPACE value of the cluster with the Access Method Services command 'ALTER clustername FREESPACE (5 20)'.

8. Load records 200001 - 300000 with REPRO or any user program using sequential insertion technique.

As result, this procedure:

- prevents wasting space (as it would be, if e.g FREESPACE (0 10) would be defined for the whole data set)

- minimizes CI or CA splits (for the first inserts there would be a huge number of CI or CA splits, if e.g. FREESPACE (0 0) would be defined for the whole data set).

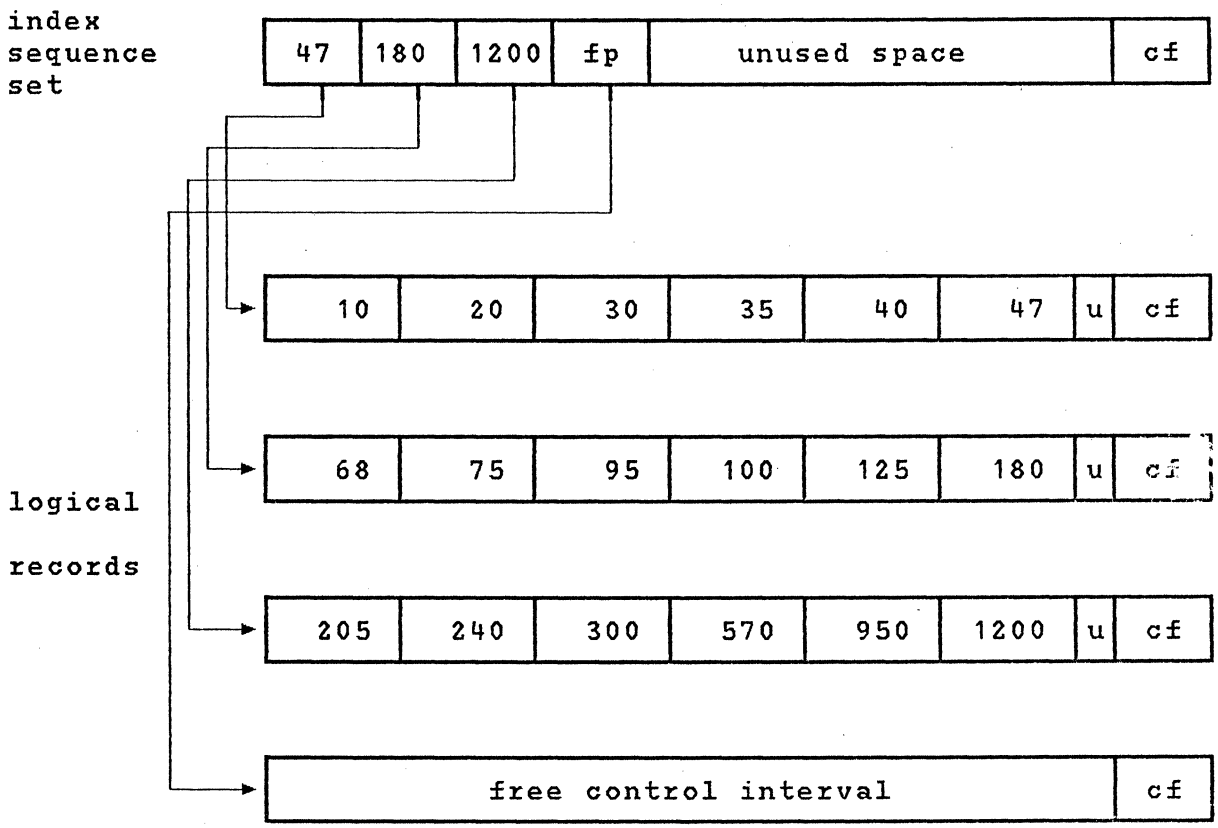
NOTE: A sequential insertion technique must be used to load the parts of the data set to reserve the amount of FREESPACE in the data set (see also section 8.8 on page 207, which describes this technique).

Subsequent sequential insertions to any point of the data set will preserve FREESPACE according to final value left in the catalog.

8.5 CONTROL INTERVAL SPLIT (DIRECT INSERT)

If the record to be inserted will not fit in the control interval, a control interval split takes place as illustrated in the following example.

- Assume the following control area contains 4 control intervals. A free space value has been specified to reserve 25% of the number of CIs to be used for later insertions. This results in one free control interval per CA (25% of 4 control intervals).



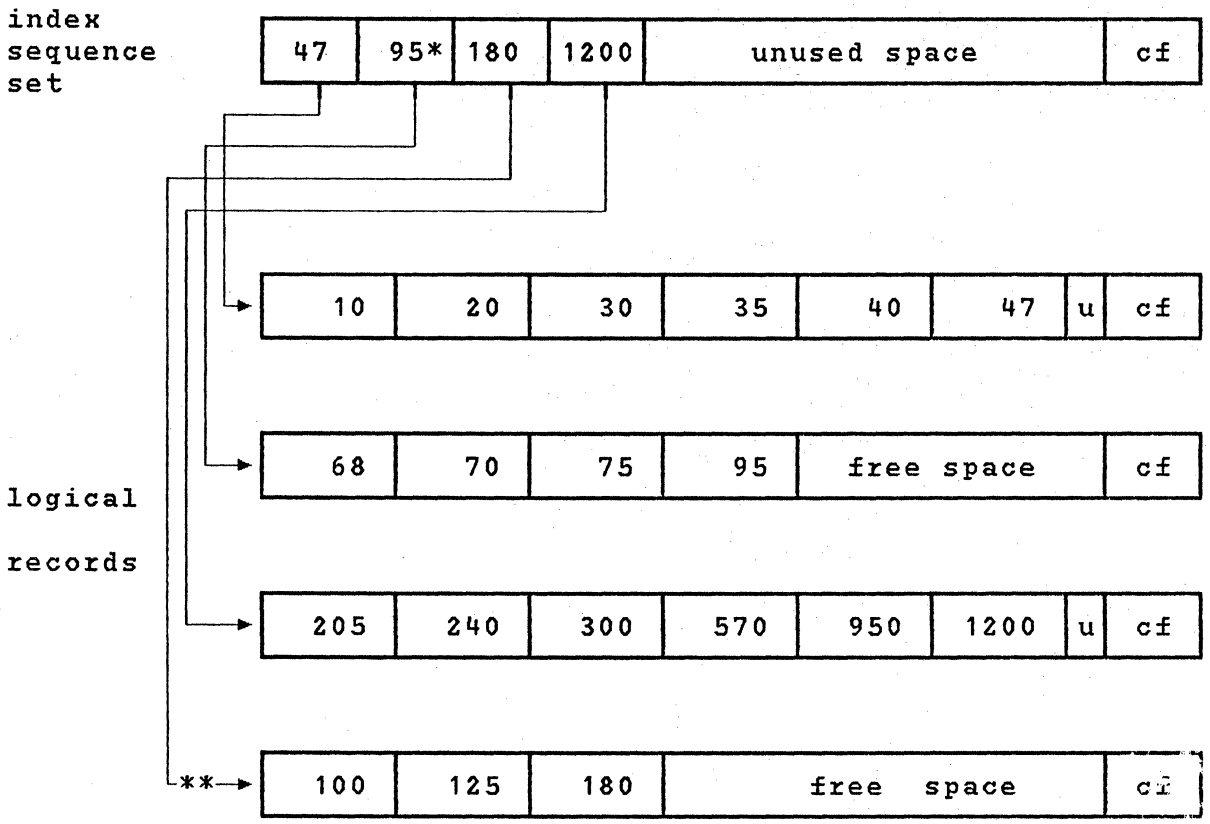
cf = control fields (RDFs, CIDF)
fp = pointer to free CI
u = unused space

Figure 43. Control interval split (1 of 2)

- Now record 70 is to be inserted. In this example, the second data control interval is read by VSAM into a VSAM buffer. By examining the control fields VSAM determines that there is not enough free space available for this record. About half of the control interval is moved from this buffer to a second empty buffer (in OS/VS systems it is already reserved for control interval split by VSAM; in DOS/VS systems it will be allocated when needed). Insert new record into appropriate buffer. If, as in this example, there is an empty control interval available (at the end of the control area), the second buffer is written into this empty control interval on auxiliary storage.

After this, the physical sequence of control intervals within the CA no longer represents the correct sequence of the logical records. Therefore, the primary index is updated to reflect this condition.

The first buffer is written back into the old control interval on auxiliary storage (the sequence of writing buffers back is different in DOS/VS systems).



cf = control fields (RDFs, CIDF)
u = unused space
* = entry added due to control interval split
** = pointer changed due to control interval split

Figure 43. Control interval split (2 of 2)

If there is no free control interval in the control area, the record is not inserted until a control area split takes place (in DOS/VS the CA is scanned for an 'erased' CI; if one found, it its used; if not, then the CA split proceeds) to provide free control intervals.

VSAM obtains a new empty control area from the end of the data set. Then the whole control area is split by reading about half of the control area into virtual storage and writing it back to auxiliary storage into the new control area.

The minimum amount of VSAM buffers used for the CA split is STRNO+1 (STRNO is the number of strings specified; see description on page 191).

Logically coincident with writing the 'new' CA to auxiliary storage, a sequence set record is constructed for it and written, the 'old' CA is rewritten, and the sequence set for the old CA is first adjusted and then rewritten.

After the control area split, the control interval split takes place and the record is inserted.

8.6 KEY COMPRESSION

As described in section 2.4 on page 13, 18 the index sequence set contains an entry for each data CI of a KSDS.

Each entry consists of the highest key in that CI and an RBA offset. To shorten the size of the index entry, the keys are compressed by VSAM routines. This is called key compression.

The key compression uses front key compression and rear key compression.

The following example shows how key compression works.

Assuming the following CIs exist in a cluster:

CI 1	10001	10002	10003	10009
CI 2	10052	10060	10070	10080
CI 3	10222	10250	10300	10333
CI 4	14021	14023	14024	14028

For front key compression the highest key of the first CI is compared and all identical values starting from left are compressed. Using the previous CIs the following compression would take place:

Full key	Front compressed key
10009 10080	10009
10080 10333	===80
10333 14028	==333
14028	=4028

For the rear key compression the highest key of the first CI is compared with the lowest key of the second CI, and so on. Value by value is compressed starting from the right as long as the second value is higher.

Full key	Rear compressed key
10009 10052	1000-
10080 10222	100--
10333 14021	10---
14028	14028

VSAM uses both types of compression. To indicate how many bytes are compressed and how large the compressed key is, two one-byte fields are used. The 'F' field contains the number of front key compressed bytes. The 'L' field contains the residual key length.

Combining front and rear key compression the compressed keys and their F+L fields are as follows ('=' is front and '-' is rear key compression):

	Full key	Full compressed key	F	L
CI 1	10009	1000-	0	4
CI 2	10080	====--	3	0
CI 3	10333	==----	2	0
CI 4	14028	14028	0	5

VSAM reconstructs a compressed key as follows:

- Front compressed values are taken from the previous uncompressed key.
- Rear compressed values are substituted by X'FF'

The previously compressed keys would then be reconstructed as follows ('f' represents the hex value X'FF'):

	Full key	Full compressed key	F	L	reconstructed key
CI 1	10009	1000-	0	4	1000f
CI 2	10080	====--	3	0	100ff
CI 3	10333	==----	2	0	10fff
CI 4	14028	14028	0	5	14028

Based on the key compression, VSAM determines, into which CI a new record has to be inserted. A record with the value 10088 would be inserted into CI 2 since the value is lower than 100ff (f=X'FF').A record with the value of 10100 would be inserted into CI 3.

Single field keys do compress well. Larger keys (20-30 bytes) compress out to 8 or 9 bytes including control information. Smaller keys (5-15 bytes) can compress to 3 to 5 bytes.

8.7 INDEX CI SIZE FORMULA

The formula used to determine the appropriate index CI size in DOS/VS and OS/VS systems is:

```
8*(CIs/CA)+2((CIs/CA)**0.5)+31 .
```

In OS/VS systems additionally another formula (for minimum calculation) is used:

```
2*(Keylength+2)+(3*(CI/CA))+31 .
```

Explanations:

Formula 1:

8*(CI/CA)	=	8 bytes per CI = (3 bytes compressed key + F + L + 3 bytes RBA displacement value; for compressed key, F and L see section 8.6 on page 203)
2*((CI/CA)**0.5)	=	calculation for the index segment entries (2 bytes * number of entries).
31	=	index control fields (24 bytes header + 3 bytes RDF + 4 bytes CIDF).

Formula 2 (OS/VS):

2*(Keylength+2)	=	2 CI/CA (minimum) * uncompressed key + L + F (for compressed key, F and L see section 8.6 on page 203)
(3*(CI/CA))	=	number of RBA pointer (displacement values) * 3 (maximum size is 3 bytes).
31	=	index control fields (24 bytes header + 3 bytes RDF + 4 bytes CIDF).

Note: The average number of entries per control interval is listed on page 195.

8.8 SINGLE/MASS-INSERTION

As described in section 2.6 starting on page 22 various techniques can be used to enter data into a KSDS:

- Keyed direct processing
- Keyed sequential processing
- Keyed skip-sequential processing

The following discussion tries to show the differences between direct processing and sequential processing when inserting records between existing records.

Mass insertion is a technique which is automatically used by VSAM when:

- The data set is opened for output (ACB MACRF=OUT)
- Sequential insertion technique is used (RPL OPTCD=SEQ)
- The records to be inserted are sorted in ascending sequence and
  - are to be loaded into an empty data set, or
  - fit between 2 existing records, or
  - are to be loaded at the end of a data set.

This technique is, for example, automatically used, when sorted records are loaded with REPRO into an empty data set.

Mass insertion reserves defined FREESPACE and does not perform CI- or CA splits (only 1 CI split is executed, if records are to be inserted between existing records). This improves loading time.

As there are no CI splits (except the first) this technique reduces DASD space usage dramatically.

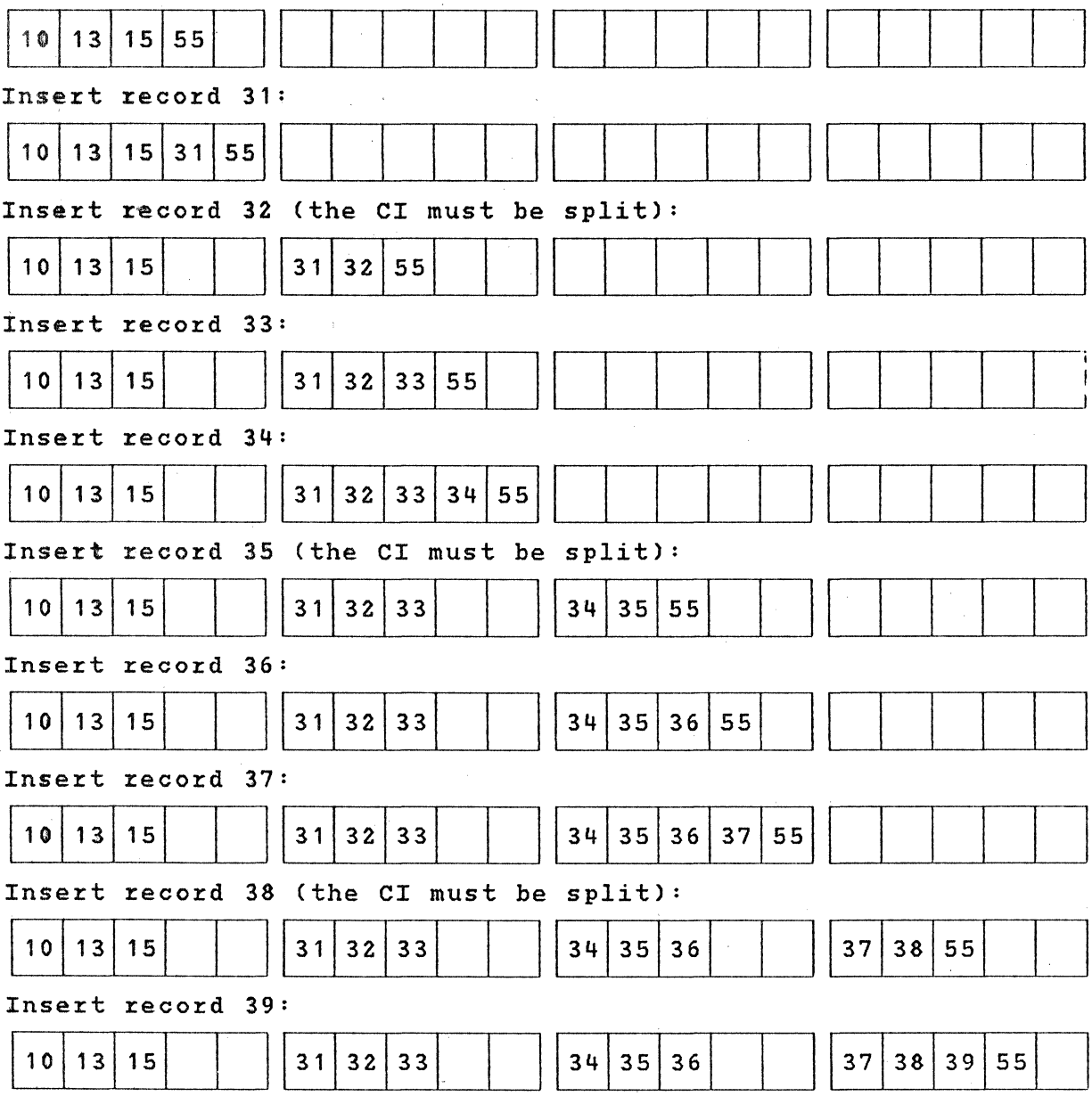
The following examples show the difference in space usage when inserting adjacent records with direct insertions or with sequential insertion (mass insert).

Note that buffers are written back to disk each time after insertion when using direct insertion and delayed (depending on the amount of available buffers) when using mass insertion.

Due to space limitations only the sequence of 9 records to be inserted can be shown. The insertion of only 9 records does not show a great enhancement (in disk space usage) of mass insertion (with FREESPACE) over direct insertion, but it is more effective, if more records are to be inserted.

In the following example records 31 to 39 are to be inserted with direct insertion technique (a search argument is supplied by the user for each record) into a VSAM data set already containing records 10, 13, 15, and 55 (first line).

The content of the 4 CIs is shown after each insert.

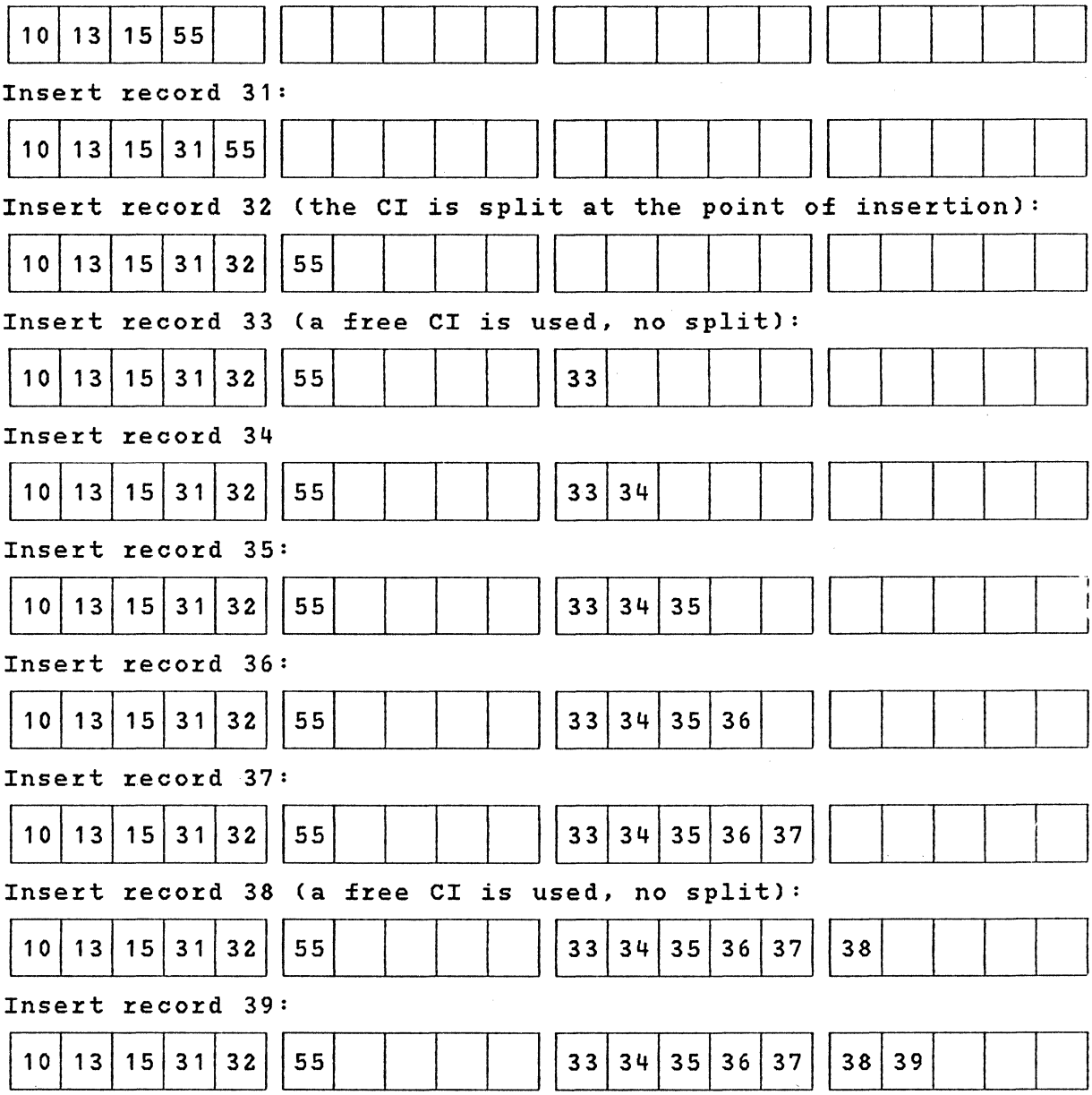


Result: 3 CIs were split and approximately 50% (in this example 40%) of their space wasted, because no records will be inserted in the free space, since adjacent records are already stored.

Figure 44. Insertions: Direct insertion

In the following example records 31 to 39 are to be inserted with mass sequential insertion technique into a data set already containing records 10, 13, 15, and 55 (first line). No FREESPACE value was defined for the data set.

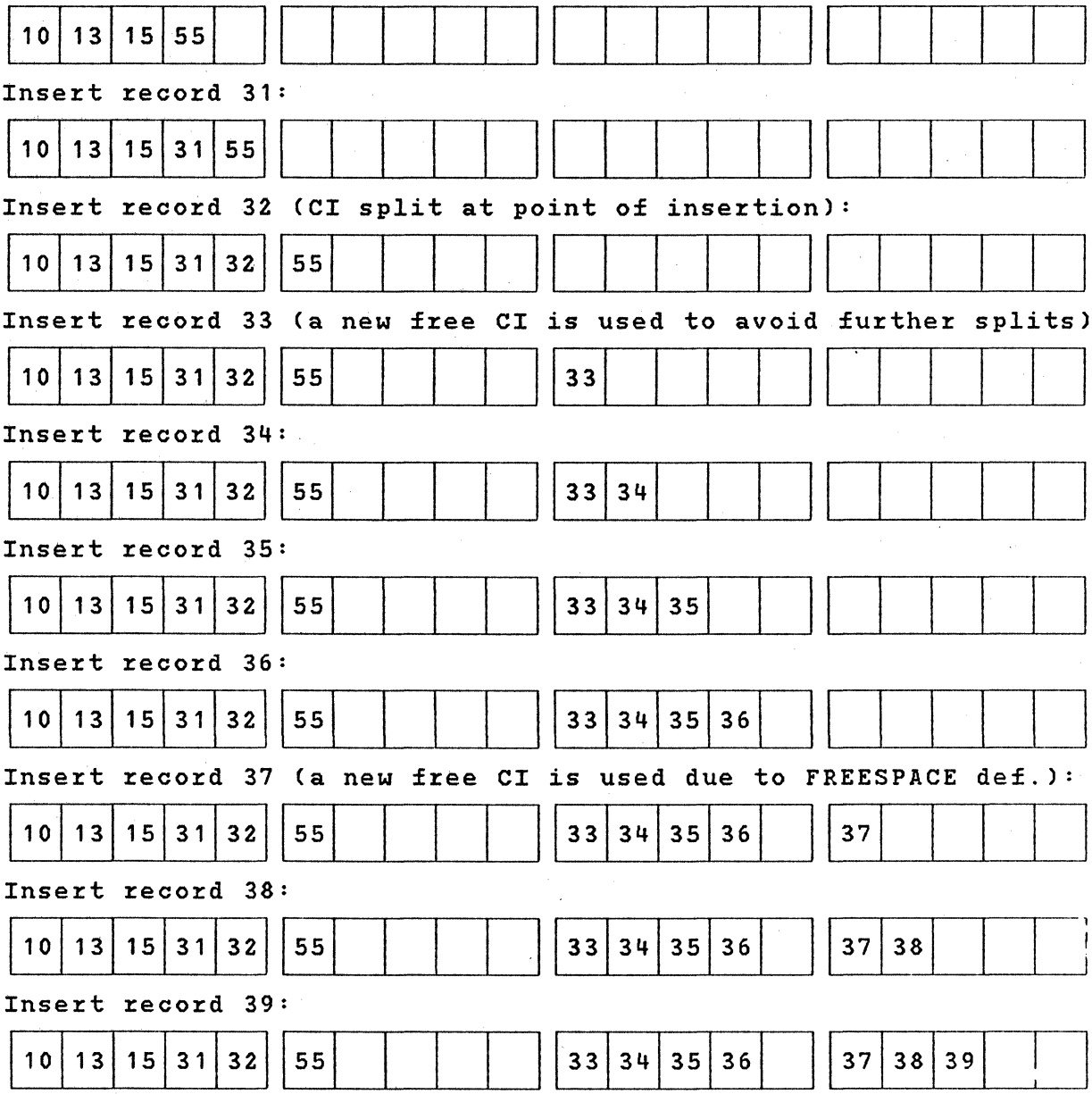
The content of the 4 CIs is shown after each insert.



Result: There is only 1 CI split at the point of insertions. Then the next free CI in the CA (or, if necessary the next free CI in the next free CA) is used for further insertions.

Figure 45. Insertions: Mass insertion (no FREESPACE)

In the following example records 31-39 are to be inserted with mass sequential insertion technique into a data set already containing records 10,13,15, and 55. A FREESPACE value of 20% free CI space was defined for the data set, which results in one free record per CI. The content of the 4 CIs is shown after each insert.



Result: The first control interval is filled independent of the FREESPACE definition, because the new records were inserted in the middle (and not at the end) of the control interval. Only for insertions at the end (which is the case starting with record 33) the amount of specified FREESPACE is held free. There is only 1 CI split at the point of insertions. Then the next free CI in the CA (or, if necessary the next free CI in the next free CA) is used for further insertions (1 record is held free per CI).

Figure 46. Insertions: Mass insertion (with FREESPACE)

8.9 VSAM RELATED PARAMETERS IN THE JCL

8.9.1 DOS/VS JCL PARAMETER

Since all VSAM data set related information is stored in a VSAM catalog, the JCL parameters for VSAM data sets are different.

```
// DLBL ...,VSAM[,BUFSP=n][,CAT=filename]

// EXTENT SYSnnn,volid[,,,start,# of tracks]
```

required for:	all VSAM files	SPACEs, UNIQUE data sets, and DEFINE MCAT/UCAT (Not needed for suballocated data sets).

```
// EXEC prog,SIZE=mK
```

NOTE: Parameter in brackets are optional.

Legend:

n = number of bytes for buffer space. This value, if higher, will replace the value indicated in the catalog or in the program.

filename = name of the catalog that owns the data set. It overrides defaults which are the master or the current user catalog (also called 'job catalog').

m = number of K bytes occupied by the program. The remaining GETVIS will be used as indicated in sections 9.2.1 and 6.1.1.

8.9.2 DOS/VS JCL TO VSAM MACRO RELATIONSHIP

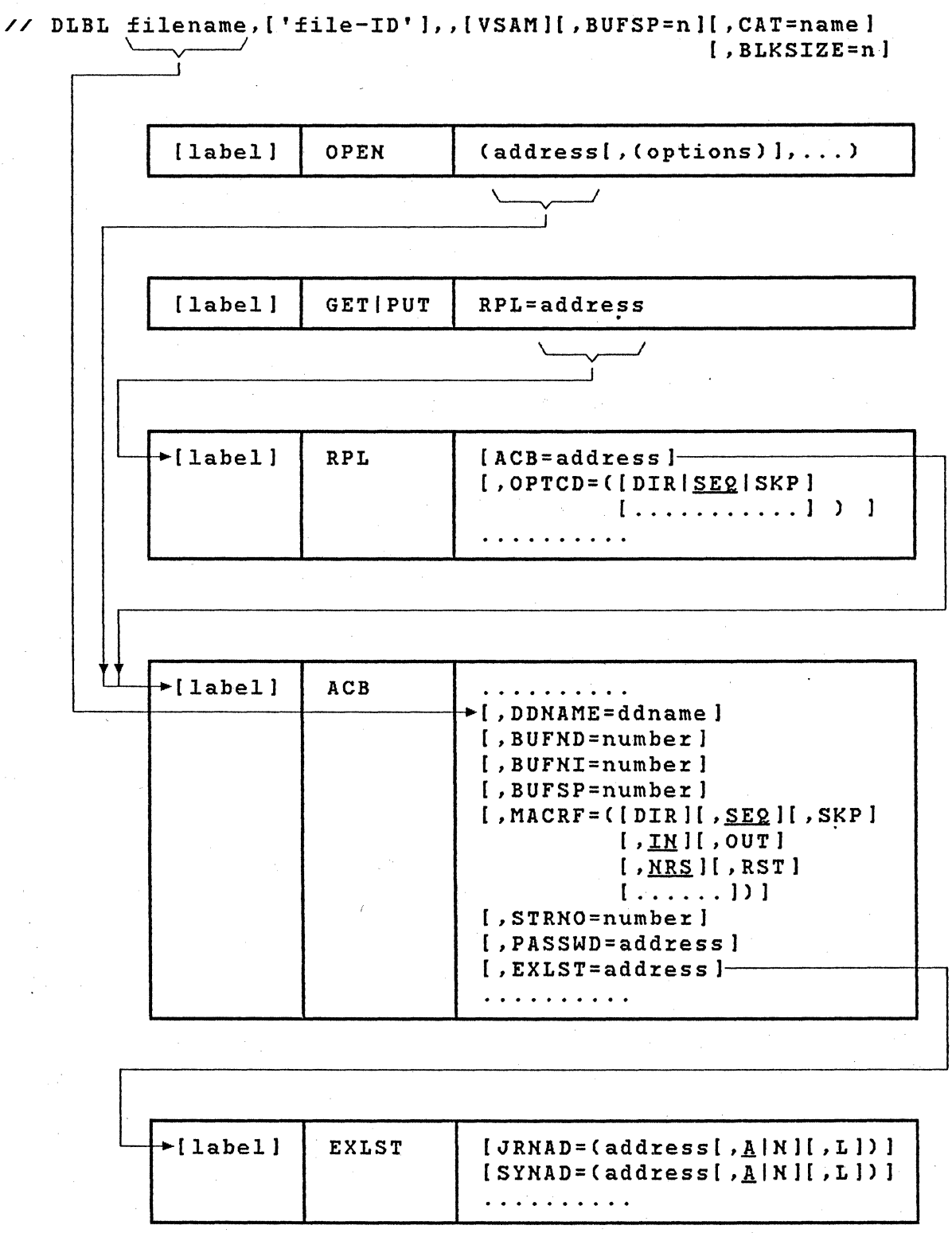


Figure 47. DOS/VS JCL to VSAM macro relationship

8.9.3 OS/VS JCL PARAMETER

Since all VSAM data set related information is stored in a VSAM catalog, the JCL parameters for VSAM data sets are different.

The DD statement has the following format:

```
//ddname DD DSN=dsname,DISP={OLD|SHR}[,UNIT=unit]
          [,VOL=SER=volser][,AMP=list]
```

DISP=OLD|PASS	usage like in normal OS/VS
DISP=SHR	for shared data sets (the amount of sharing is specified with the SHAREOPTION parameter at DEFINE time (or changed with ALTER)
UNIT and VOL	only necessary for Subset Mount (see page 216)

AMP has the following parameters:

[AMP=['AMORG']	only necessary if UNIT and VOL specified (see page 216)
[,'BUFND=number']	overrides ACB specification
[,'BUFNI=number']	overrides ACB specification
[,'BUFSP=number']	overrides ACB specification
[,'STRNO=number']	overrides ACB specification
[,'SYNAD=modulename']	overrides EXLST specification
[,'TRACE']	use with GTF for OPEN/CLOSE/EOV
[,'CROPS={RCK|NCK|NRE|NRC}']	Checkpt. Restart Opt. (GC26-3784)
[,'OPTCD={I|L|IL}']	ISAM options (for ISAM Interface)
[,'RECFM={F|FB|V|VB}']]	ISAM options (for ISAM Interface)

If data sets residing in a user catalog are to be accessed this user catalog must be specified either with a JOBCAT DD statement (the user catalog is used for all steps of the job) or with a STEPCAT DD statement (the user catalog is used for this step only).

In MVS JOBCAT or STEPCAT DD statements are not necessary if the high level qualifier of the data set name is the same name as the name of a user catalog or the ALIAS of one. A STEPCAT DD statement overrides a JOBCAT DD statement.

The JOBCAT DD statement must follow the JOB statement (after a JOBLIB DD statement if available) before the EXEC statement.

The STEPCAT DD statement must follow the EXEC statement and can be placed anywhere among the other DD statements.

JOBCAT and STEPCAT DD statements may be concatenated to specify more than one user catalog to be used for the job or the step.

The format of these statements is as follows:

```
//JOBCAT  DD DSN=usercatalogname,DISP=SHR
//STEPCAT DD DSN=usercatalogname,DISP=SHR
```

8.9.3.1 OS/VS JCL DDNAME/DSNAME SHARING

If a VSAM data set has to be accessed directly and sequentially simultaneously either multiple string processing or DDNAME/DSNAME sharing can be used.

The following applies:

- DDNAME/DSNAME sharing works like multiple string processing from the VSAM point of view
- VSAM maintains integrity
- CIs obtained for UPDATE are 'locked'
- 'Exclusive control error' is returned if access to 'locked' CI is attempted
- multiple copies of the same CI can be in storage

The following is a short example of how to use DDNAME sharing:

```
//VSAM1  DD DSN=cluster,DISP=SHR

  ACB1  ..... DDNAME=VSAM1,MACRF=DDN ...
  ACB2  ..... DDNAME=VSAM1,MACRF=DDN ...
```

The following is a short example of how to use DSNAME sharing:

```
//VSAM1  DD DSN=cluster,DISP=SHR
//VSAM2  DD DSN=cluster,DISP=SHR

  ACB1  ..... DDNAME=VSAM1,MACRF=DSN ...
  ACB2  ..... DDNAME=VSAM2,MACRF=DSN ...
```

8.9.4 OS/VS JCL TO VSAM MACRO RELATIONSHIP

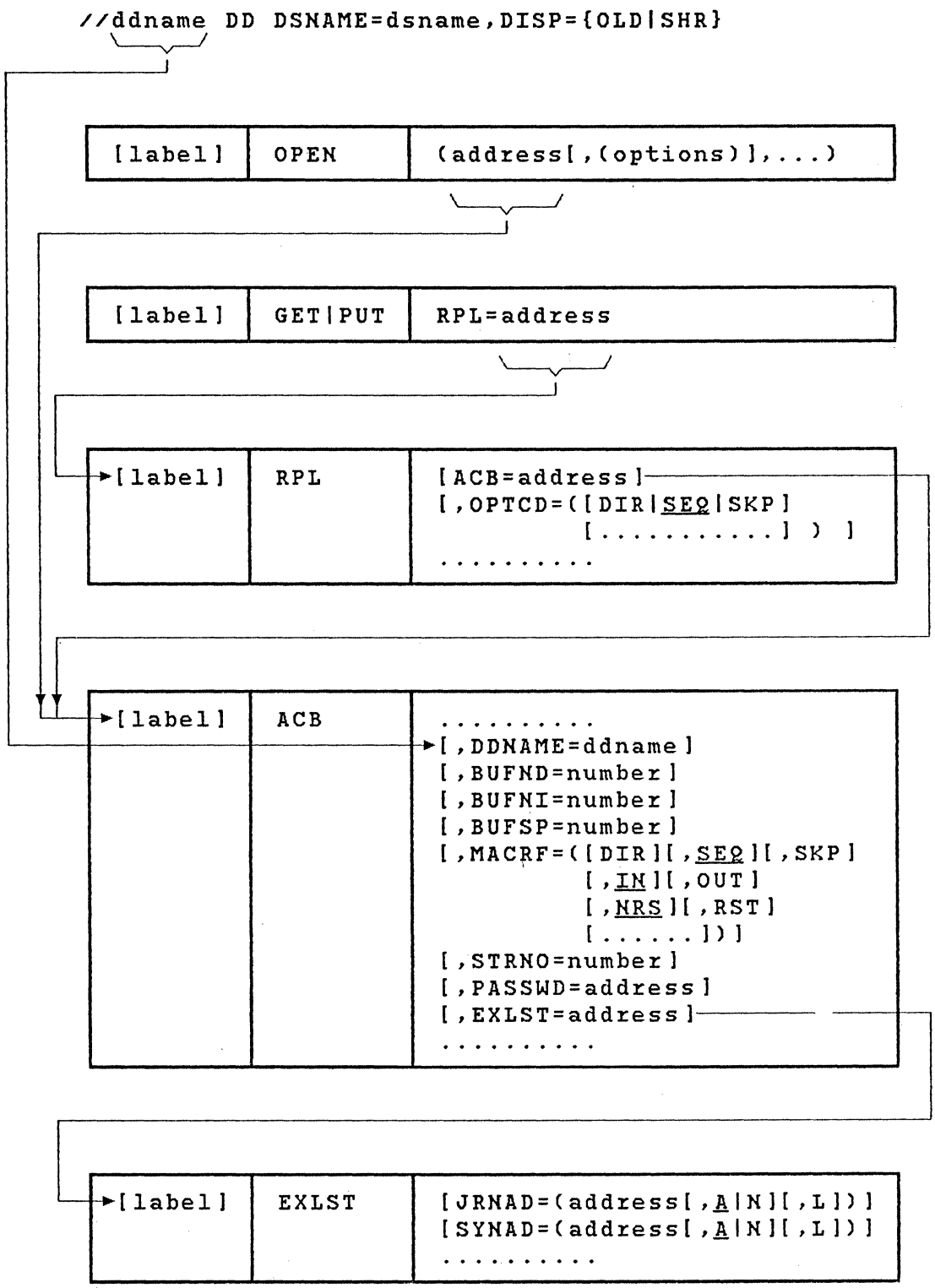


Figure 48. OS/VS JCL to VSAM macro relationship

8.10 SUBSET MOUNT

'Subset Mount' is not part of VSAM and must be used very carefully to avoid mounting and demounting problems. If possible, 'Subset Mount' should be used for read-only operations.

Special problem areas are:

- Index and data component on different device types
- Extending a data set onto a new volume
- Using recoverable catalogs

8.10.1 SUBSET MOUNT (DOS/VS)

In DOS/VS for all but Keyrange data sets, all the volumes of a VSAM data set must be specified in the EXTENT statement. If only a specific volume of a multivolume Keyrange data set should be mounted, only this volume may be specified in the EXTENT statement (so this is the same as OS/VS subset mount).

8.10.2 SUBSET MOUNT (OS/VS)

In OS/VS a VSAM data set and its volumes are found by specifying only the cluster name and, if necessary, the catalog name. OS/VS then uses the catalog information and issues a mount request for all volumes of the data set.

If a multivolume data set is to be opened, normally all volumes are mounted.

If the user knows however, on which volume the required data is stored, (if key ranges are used, it is very easy to determine, which volume contains the data) he can mount only this volume with subset mount (mounting only one volume of a multivolume data set may be desirable, if insufficient drives are available or to reduce mount time).

The following example shows how to use OS/VS subset mount:

- Assuming the data set KSDSKEYR was defined with key ranges using three volumes (V1: keys A-F, V2: keys G-P, V3: keys Q-Z) (see key range example 1 in section 7.2.4).

  If the specific job only uses keys between G and P, only volume V2 need be mounted.

OS/VS JCL for normal and for subset mount:

- Normal mount (all 4 volumes)

```
//DD1  DD  DSN=KSDSKEYR,DISP=OLD
```

- OS/VS subset mount (only volume V2)

```
//DD1  DD  DSN=KSDSKEYR,DISP=OLD,UNIT=3330,
//         VOL=SER=V2,AMP='AMORG'
```

AMP='AMORG' may be specified to specify buffers, etc.

If there is any access to a part of the data set, which is not on volume V2, an error message is issued (no additional volume is mounted).

8.11 VSAM ASSEMBLER MACROS (SHORT DESCRIPTION)

The macros provided to define and process VSAM data sets are divided into control block macros and request macros.

The control block macros are used to define, modify, display, and test the contents of VSAM control blocks and lists. The request macros are used to specify the processing action (read, write, etc.) to be taken on data and index records.

8.11.1 VSAM CONTROL BLOCK MACROS

8.11.1.1 ACB (GENERATE AN ACCESS CONTROL BLOCK)

The ACB macro causes an access control block to be generated during program assembly. One ACB (or GENCB, see section 8.11.1.4 on page 219) macro must be specified in a program for each VSAM data set that is to be processed by the program. More that one ACB can be specified in a program for the same VSAM data set. In this case, the ACB's are connected to the same VSAM control block structure and the same set of I/O buffers is used for all requests issued to the data set (see section 8.9.3.1 on page 214). The access control block for a VSAM data set must be opened before any processing of the data set can occur.

The ACB specifies the following for a VSAM data set: name of the DD statement (OS/VS) or the DLBL 'filename' (DOS/VS) for the data set, address of a list of exit routine addresses for user-written exit routines, buffer space requirements, the password required for the type of processing to be done, all processing options to be used with the data set (keyed, addressed, and/or control interval, sequential, skip-sequential, and/or direct, etc.), and the number of requests that can be outstanding concurrently for the data set using this ACB.

8.11.1.2 EXLST (GENERATE AN EXIT LIST)

The EXLST macro is used to define a list of the addresses of the user-written exit routines that are to be entered when certain conditions occur during the processing of a VSAM data set. The EXLST macro causes an exit list to be generated during program assembly.

Exit to a user-written routine can be taken when end of data set is reached (EODAD exit), a logical error occurs (LERAD exit), an uncorrectable physical I/O error occurs (SYNAD exit), or to perform a journaling operation (JRNAD exit). Each exit routine can be marked active or inactive. An exit routine that is inactive is not entered when its associated condition occurs. The exits to be used during the processing of a given VSAM data set are specified in its ACB (the address of an EXLST macro can be given). More that one ACB can specify the same EXLST macro.

The journaling exit is taken by VSAM at the following times: whenever a GET, PUT, or ERASE macro is issued to the VSAM data set; each time data is shifted within a control interval or moved to another control interval (key-sequenced data sets only); and each time a physical I/O error occurs.

A user-written journaling routine can be used, therefore, to keep track of any RBA changes for the logical records of a key-sequenced data set, if it is to be processed by RBA, and/or to record the VSAM requests that are processed against a VSAM data set (for recovery and reconstruction purposes, for example).

8.11.1.3 RPL (GENERATE A REQUEST PARAMETER LIST)

An RPL macro is used to generate a request parameter list during program assembly. This list defines a request for processing. Certain request macros (GET, PUT, ERASE, POINT, CHECK, ENDREQ, GETIX, and PUTIX) must specify the address of a request parameter list to indicate the processing to be performed. The same RPL can be specified in more than one type of request macro.

An RPL macro specifies the following: the ACB of the data set with which it is to be used (multiple RPL macros can specify the same ACB); the size and address of a work area if logical records are not to be processed in an I/O buffer; the search argument to be used during direct retrieval, skip-sequential retrieval, and positioning (full key, generic key, RBA, or relative record number); address of an ECB if this is an asynchronous request (optional parameter); the type of processing for this request, such as keyed or addressed, sequential or direct, forward or backward, synchronous or asynchronous request (OS/VS only), etc.

When a synchronous request is specified in the RPL indicated by a GET or PUT macro, control is not returned to the instruction after the GET/PUT macro until processing of the request is completed. The logical record is then available for processing.

In OS/VS when an asynchronous request is specified, control returns to the instruction after the GET/PUT macro as soon as the request has been scheduled. The user must then test for completion of the I/O operation (usually using a CHECK macro). Asynchronous processing of a request permits the overlap of I/O operations with program execution and is particularly useful with skip-sequential and direct processing. Up to 255 asynchronous requests (RPL's) can be outstanding concurrently for the same VSAM data set.

Two or more RPLs can be chained together via a pointer field in the RPL itself. A chained parameter list can be used to read or write several records (one for each RPL in the chain) using one GET or PUT macro instead of multiple macros. Chained parameter lists can be used only to retrieve several existing records or to add several new records. It cannot be used to retrieve-for-update, update, or delete existing records.

8.11.1.4 GENCB (GENERATE A CONTROL BLOCK OR LIST)

The GENCB macro can be used to generate an ACB, EXLST, or RPL during program execution instead of program assembly. The GENCB macros can be used to eliminate changing these control macros and reassembling VSAM programs when control block formats change in new versions of VSAM.

The same parameters can be specified in a GENCB macro as in ACB, EXLST, and RPL macros. However, a GENCB macro can specify that multiple copies of the control block are to be generated and parameter values can be specified in more ways (such as in general registers).

8.11.1.5 MODCB (MODIFY CONTENTS OF CONTROL BLOCK OR LIST)

The MODCB macro is used to change, during program execution, the contents of an unopened ACB, an EXLST, or an inactive RPL (one not currently involved in a processing operation).

8.11.1.6 SHOWCB (DISPLAY CONTENTS OF CONTROL BLOCK/LIST)

The SHOWCB macro issued to place the contents of user-specified fields of an ACB, EXLST, or RPL in a user-specified work area.

8.11.1.7 TESTCB (TEST CONTENTS OF CONTROL BLOCK OR LIST)

The TESTCB macro is used to have VSAM compare a user-specified value with a field in an ACB, EXLST, or RPL. The condition code in the PSW is set to indicate the results of the comparison.

8.11.1.8 SHOWCAT (DISPLAY FIELDS IN A VSAM CATALOG)

The SHOWCAT macro can be used to cause selected fields from the catalog entry for a specified data set to be moved to a user-provided work area. The data set whose catalog entry is being inspected need not be open in order for the SHOWCAT macro to be issued.

8.11.2 VSAM REQUEST MACROS

8.11.2.1 OPEN

A VSAM data set must be opened before it can be processed by other request macros. The OPEN macro provides the same types of processing functions for VSAM data sets as for other types of data sets. OPEN causes the volumes of the VSAM data set to be mounted if necessary, constructs the control blocks required (in addition to those already created by EXLST, ACB, and GENCB macros) for the type of processing to be done, overrides information in the ACB and EXLST with any parameters specified in the DD statement for the data set, causes the loading into virtual storage of any VSAM routines required (in addition to the resident VSAM routines) for the processing specified, and verifies that the password given is correct. Any parameter not specified via job control or the ACB is taken from the catalog entry for the data set.

Both sequential and direct processing can be performed on a VSAM data set using one OPEN macro and one ACB. Closing and reopening of the data set to switch modes, as is required for an ISAM data set, is not necessary.

8.11.2.2 GET

This macro is used for simple retrieval and for retrieval for update (GET for update) operations. The RPL specified by a GET macro indicates whether the request is for a retrieval only or a retrieve and update operation. A record that was retrieved by a GET-for-update request need not be written back if it is not to be changed.

Locate mode (logical record made available in the input buffer) can be specified for retrieval only (GET) and retrieve for update without record length change (GET-for-update) operations. In the latter case, however, the updated record must be placed in a work area before it is rewritten. Move mode (logical record made available in a work area) is supported for all read and write requests and is required for all write (PUT and ERASE) operations.

8.11.2.3 PUT

This macro is used to write a new record in a data set during its creation or to insert a new record in an existing data set. A PUT for update is used to change the contents of an existing record (update it or mark it deleted with a user-defined deletion indication). A PUT-for-update request must be preceded by a GET-for-update request. Write verification (automatic reading by DASD hardware after each write operation) is optional and can be specified with the DEFINE parameter WRITECHECK (not suggested).

8.11.2.4 ERASE

This macro is used to delete a logical record from a key-sequenced or relative record data set. The record is physically removed from the data set. An ERASE macro must be preceded by a GET-for-update macro.

8.11.2.5 POINT

This macro is used to position VSAM to a particular logical record in the data set from which processing is to continue. Positioning can be in a forward or backward direction and a key value (including a relative record number) or RBA (not for RRDS) can be used to identify the logical record at which positioning is set.

8.11.2.6 CHECK (OS/VS ONLY)

This macro is used to cause VSAM to determine whether processing of a specific asynchronous request has been completed and to suspend program execution until processing is completed for an incomplete request. CHECK also causes the appropriate active user-written exit routine to be entered, if necessary, at the completion of the request.

A test for the completion of an asynchronous request can also be made by specifying an ECB in the RPL for the request and testing the completion bit. Completion can be tested using the TESTCB macro (IO=COMPLETE operand) as well. These two completion tests can be used to delay issuing the CHECK macro until the operation is completed so that processing is not suspended by the CHECK macro.

8.11.2.7 ENDREQ

This macro is used to terminate the processing of a chain of requests or a single asynchronous request whose completion is no longer required to to free VSAM from keeping track of a position in a data set. VSAM can maintain knowledge of the same number of positions as the number of requests that can be outstanding concurrently (specified in the ACB or GENCB macro).

8.11.2.8 GETIX AND PUTIX (OS/VS ONLY)

These macros are used to process an index component of a key-sequenced data set (special application).

8.11.2.9 CLOSE

The CLOSE macro provides the same types of processing functions for VSAM data sets as for other types of data sets. It causes VSAM to write any unwritten data or index records remaining in the output buffers if their contents have changed, update the catalog entry for the data set, if necessary (if the location of the end-of-file indicator has changed, for example), and write SMF records if SMF is being used. The access method control block(s) for the data set (such as the ACB's) are restored to what they were before the data set was opened and virtual storage that was obtained during OPEN processing for additional VSAM control blocks and VSAM routines is released.

Once a VSAM data set has been closed, it must be reopened before any additional processing can be performed on it. A CLOSE macro with TYPE=T (OS/VS) or TCLOSE (DOS/VS) (temporary CLOSE) can be issued to cause VSAM to complete any outstanding I/O operations, update the catalog if necessary, and write any required SMF records. Processing can continue after a temporary CLOSE without the issuing of an OPEN macro.

8.11.3 VSAM MACROS FOR SHARED RESOURCES

This is an interface to VSAM that is designed to be used in a data base/data communications environment. Five macros are available. Shared resources enable the user to share I/O buffers, I/O-related control blocks, and channel programs among several VSAM data sets and permit management of I/O buffers. The sharing of I/O resources and the buffer management available can speed up the direct processing of VSAM data sets whose activity is unpredictable and the processing of one transaction that requires access to several data sets.

8.11.3.1 BLDVRP AND DLVRP

The BLDVRP and DLVRP macros are provided to build and delete, respectively, a shared resource pool. A resource pool can be shared by the VSAM data sets being processed in the same partition.

8.11.3.2 WRTBFR

The WRTBFR macro causes the writing of a buffer and can be used when deferred processing is specified in the ACB. When deferred

processing is specified, VSAM does not write a buffer after a PUT for direct processing is issued.

8.11.3.3 SCHBFR AND MRKBFR (OS/VS ONLY)

The SCHBFR macro is provided to search the shared buffer pool for a particular range of RBAs (locate a buffer) and the MRKBFR macro causes a buffer to be marked for output without issuing a PUT for update. Details regarding the use of these five macros are contained in OS/VS Virtual Storage Access Method: Options for Advanced Applications.

8.11.4 CONCURRENT PROCESSING (STRINGS)

Note that several parts of a VSAM data set can be accessed concurrently via sequential and direct processing by a program or its subtasks using the same ACB without the necessity of closing and reopening the data set. Each request is processed independently and asynchronously with respect to all other outstanding requests. This is called concurrent request processing and is made possible by the fact that VSAM can keep account of multiple positions in the data set at one time. The number of concurrent requests that can be outstanding is specified in the ACB but is extended by VSAM during processing if necessary.

Concurrently outstanding requests for a data set can be any combination of sequential and direct processing requests. Each outstanding request can specify one RPL or a list of RPLs (chained RPLs) and synchronous or asynchronous processing (OS/VS only). When a request consists of a list of RPLs, the first RPL in the list determines whether synchronous or asynchronous processing is performed for the request.

When synchronous processing is requested in the first RPL, control is not returned to the user until all requests in the list have been processed. When asynchronous processing is specified in the first RPL, control is returned to the user as soon as the chained request is accepted by VSAM, and the processing status of the list must be checked by the user by issuing a CHECK macro for each RPL in the list.

For additional information about string processing see also page 191.

9.0 SYSTEM CONSIDERATIONS

All storage figures given in this chapter are the best approximations available and may differ from user to user.

9.1 WHERE IS VSAM IN MAIN STORAGE

9.1.1 ACCESS METHOD SERVICES AND VSAM IN STORAGE (DOS/VS)

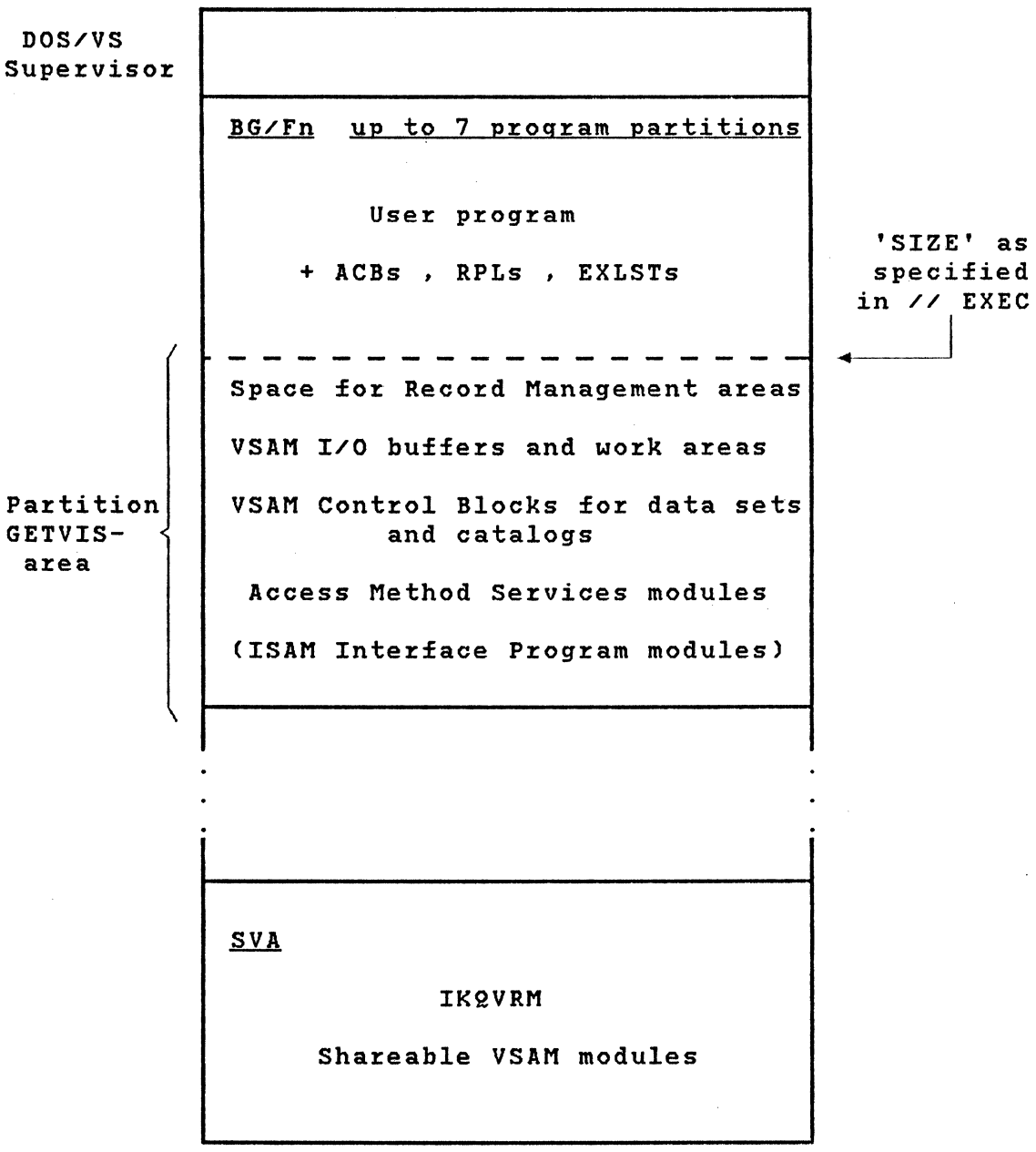


Figure 49. Where is VSAM in storage (DOS/VS)

9.1.2 ACCESS METHOD SERVICES AND VSAM IN STORAGE (VS1/SVS)

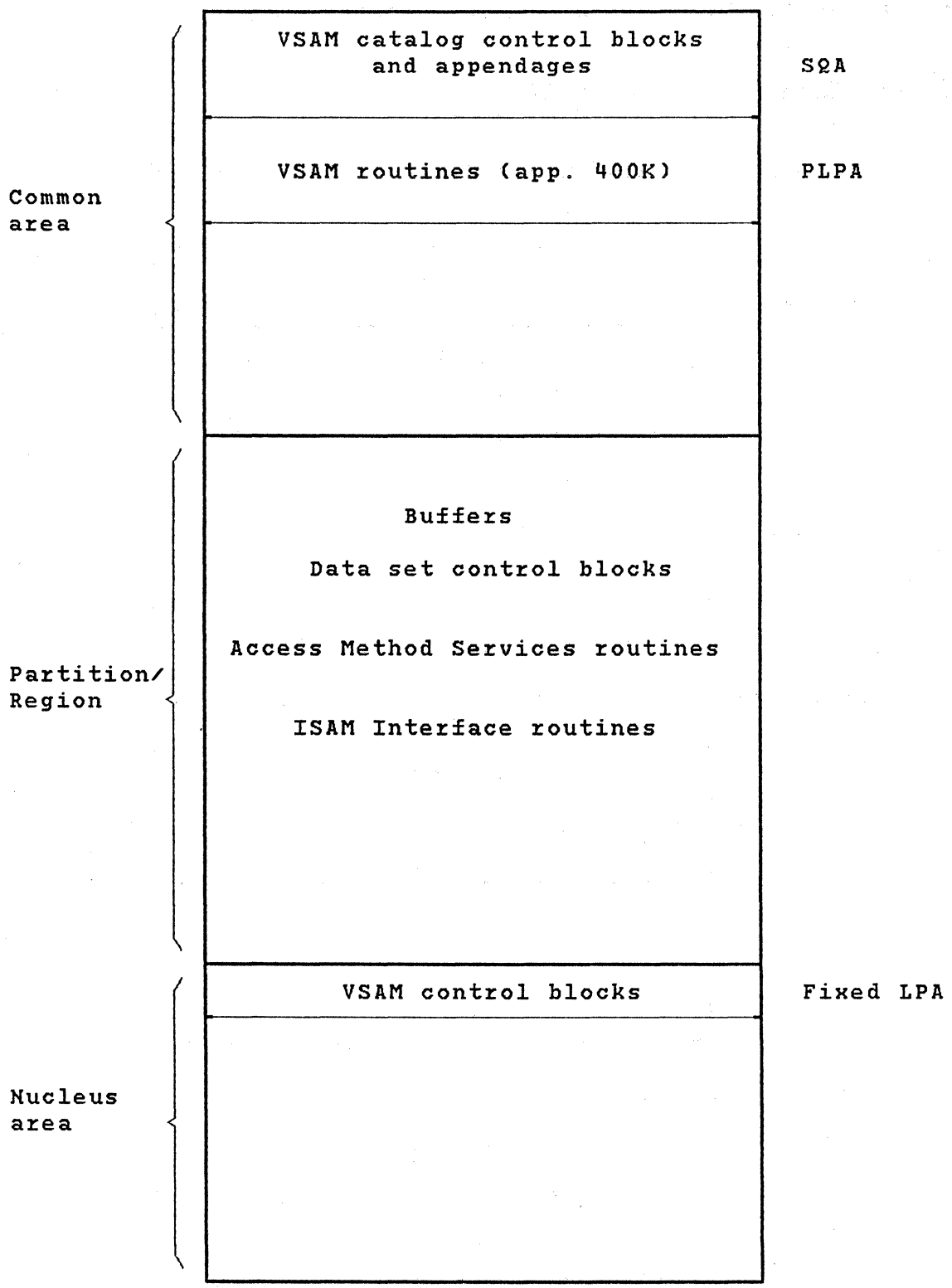


Figure 50. Where is VSAM in storage (VS1/SVS)

9.1.3 ACCESS METHOD SERVICES AND VSAM IN STORAGE (MVS)

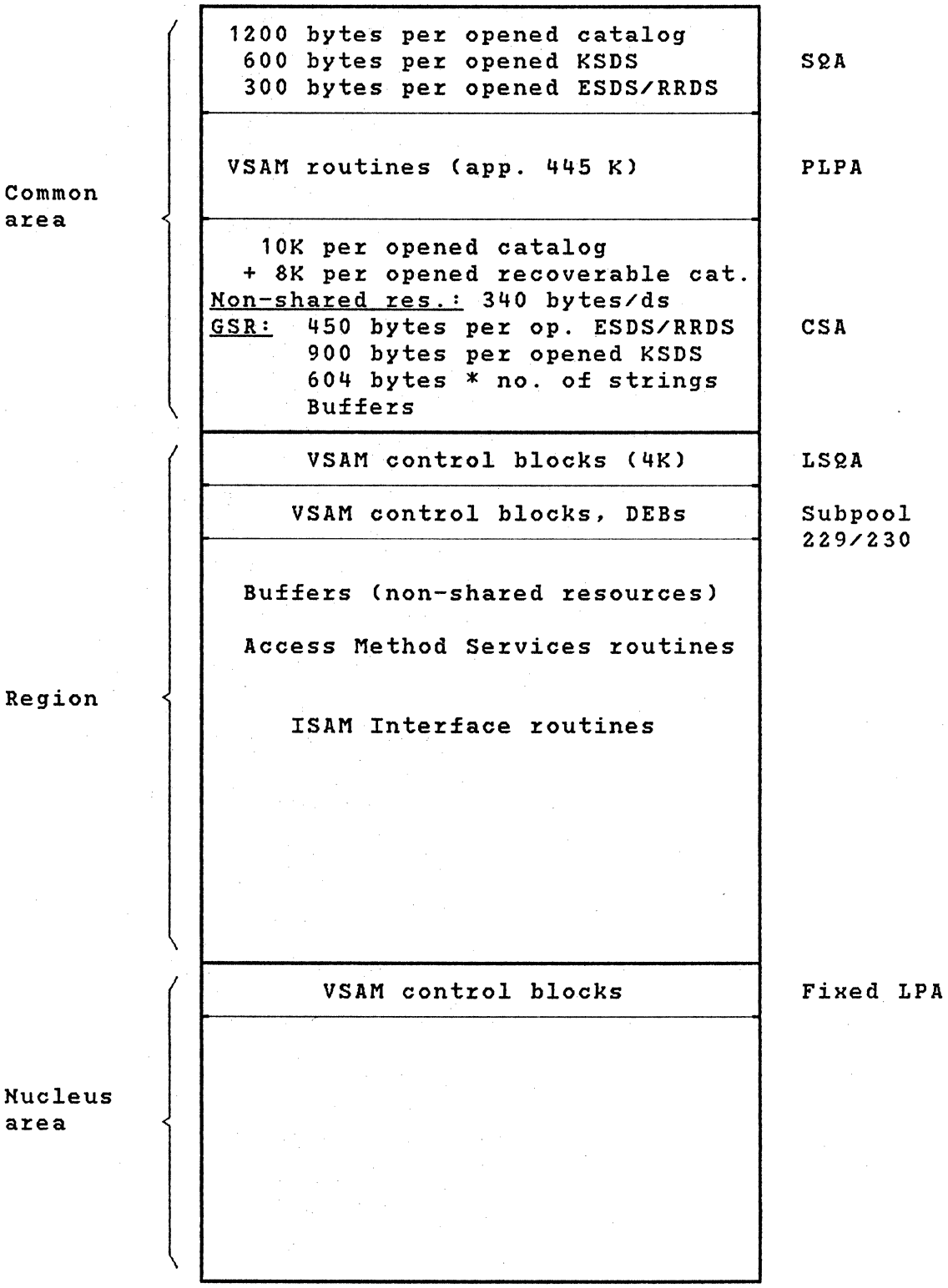


Figure 51. Where is VSAM in storage (MVS)

9.2 VSAM WORKING SETS

9.2.1 MINIMUM WORKING SET FOR A SINGLE KSDS (DOS/VS)

The virtual storage requirements of VSAM are big, but it is designed to operate efficiently in a VS environment with relatively small Real Storage (Working Set) requirements. For further information see the manual 'DOS/VS System Generation', GC33-5377).

The following conditions are assumed:

- One EXTENT for the data component
- SHAREOPTION(4) not specified (SHAREOPTIONs are described on page 95)
- The key is 4 bytes long
- One extent for the sequence set and the index set
- 2 data buffers and 1 index buffer (KSDS data set)
- The ACB and RPL are created in this sequence via GENCB, leaving the space allocation up to VSAM (ACB, RPL, and GENCB are VSAM macros and are described on page 217)

The minimum working set for a single VSAM data set (KSDS, ESDS, RRDS) can be calculated as follows :

- IKQVRM (VSAM record management)	22 K[1]
- (if loading/extending the file	4 K)
- Control blocks and channel program area	4 K
	26-30 K + VSAM buffers

[1] per partition or system if in SVA

If an alternate index is processed as an end-use object, that is, without its related base cluster, it is treated as a KSDS data set.

For each additional data set in the partition, the following has to be added to this minimum :

- Control blocks and channel program area	4 K
- VSAM buffers	n K

9.2.1.1 WORKING SET FOR ALTERNATE INDEX PROCESSING(DOS/VS)

If a path is established between an alternate index and its base cluster, the working set requirements for processing the path are as follows :

-	IKQVRM		28 K per partition
-	Path entry	min.	2 K
-	Base cluster	min.	4 K
-	Other alternate indexes in the upgrade set	min.	2 K per each one
-	Buffers for the base cluster, the AIX path and each remaining alternate index with the UPGRADE attribute belonging to the base cluster (see also section 8.1.5)		m K

Notes:

1. All alternate indexes of an upgrade set (except that of the path entry) share a common set of buffers whose size is that of the largest buffers specified.

2. When opening an alternate index and its base cluster through a NOUPDATE path, no other alternate index is opened, even if its a member of the same upgrade set (parameter NOUPDATE is explained on page 110). Therefore, only space for the path entry and the base cluster has to be provided unless the base cluster is opened with NOUPDATE.

3. If a base cluster is not processed via a path, but has an upgrade set assigned to it, space for the base cluster itself and for the upgrade set members (alternate indexes) has to be provided.

9.2.2 MINIMUM WORKING SET FOR A SINGLE KSDS (MVS)

The virtual storage requirements of VSAM are big, but it is designed to operate efficiently in a VS environment with relatively small Real Storage (Working Set) requirements.

The following conditions are assumed:

- One EXTENT for the data component
- SHAREOPTION(4) not specified (SHAREOPTIONs are described on page 95)
- The key is 4 bytes long
- One extent for the sequence set and the index set
- 2 data buffers and 1 index buffer (KSDS data set)
- The ACB and RPL are created in this sequence via GENCB, leaving the space allocation up to VSAM (ACB, RPL, and GENCB are VSAM macros and are described on page 217)

The minimum working set for a single VSAM data set (KSDS, ESDS, RRDS) can be calculated as follows :

- VSAM record management	36	K[1] PLPA
- (if loading/extending the file	12	K)
- Control blocks and channel program area	3.8	K
	39.8-51.8	K + VSAM buffers

[1] PLPA is shared by all regions

If an alternate index is processed as an end-use object, that is, without its related base cluster, it is treated as a KSDS data set.

For each additional data set in the region, the following has to be added to this minimum :

- Control blocks and channel program area	3.8 K
- VSAM buffers	n K

9.2.2.1 WORKING SET FOR ALTERNATE INDEX PROCESSING (MVS)

If a path is established between an alternate index and its base cluster, the working set requirements for processing the path are as follows :

-	VSAM Record management	48 K[1]	PLPA
-	Path entry min.	4 K[2]	
-	Base cluster min.	3.8 K[2]	
-	Other alternate indexes in the upgrade set min.	4 K[2]	per each one
-	Buffers for the base cluster, the AIX path and each remaining alternate index with the UPGRADE attribute belonging to the base cluster (see also section 8.1.5)	m K	

[1] PLPA is shared by all regions
[2] includes channel program areas

Notes:

1. All alternate indexes of an upgrade set (except that of the path entry) share a common set of buffers whose size is that of the largest buffers specified.

2. When opening an alternate index and its base cluster through a NOUPDATE path, no other alternate index is opened, even if its a member of the same upgrade set (parameter NOUPDATE is explained on page 110). Therefore, only space for the path entry and the base cluster has to be provided unless the base cluster is opened with NOUPDATE.

3. If a base cluster is not processed via a path, but has an upgrade set assigned to it, space for the base cluster itself and for the upgrade set members (alternate indexes) has to be provided.

9.2.3 MINIMUM WORKING SET FOR A SINGLE KSDS (SVS)

The virtual storage requirements of VSAM are big, but it is designed to operate efficiently in a VS environment with relatively small Real Storage (Working Set) requirements.

The following conditions are assumed:

- One EXTENT for the data component
- SHAREOPTION(4) not specified (SHAREOPTIONs are described on page 95)
- The key is 4 bytes long
- One extent for the sequence set and the index set
- 2 data buffers and 1 index buffer (KSDS data set)
- The ACB and RPL are created in this sequence via GENCB, leaving the space allocation up to VSAM (ACB, RPL, and GENCB are VSAM macros and are described on page 217)

The minimum working set for a single VSAM data set (KSDS, ESDS, RRDS) can be calculated as follows :

- VSAM record management	36 K[1]	PLPA
- (if loading/extending the file	12 K)	
- Control blocks and channel program area	3.3 K	
	39.3-51.3 K	+ VSAM buffers

[1] PLPA is shared by all regions

If an alternate index is processed as an end-use object, that is, without its related base cluster, it is treated as a KSDS data set.

For each additional data set in the region, the following has to be added to this minimum :

- Control blocks and channel program area	3.3 K
- VSAM buffers	n K

9.2.3.1 WORKING SET FOR ALTERNATE INDEX PROCESSING (SVS)

If a path is established between an alternate index and its base cluster, the working set requirements for processing the path are as follows :

-	VSAM Record management	48 K[1]	PLPA
-	Path entry min.	3.4 K[2]	
-	Base cluster min.	3.3 K[2]	
-	Other alternate indexes in the upgrade set min.	3.4 K[2]	per each one
-	Buffers for the base cluster, the AIX path and each remaining alternate index with the UPGRADE attribute belonging to the base cluster (see also section 8.1.5)	m K	

[1] PLPA is shared by all regions
[2] includes channel program areas

Notes:

1. All alternate indexes of an upgrade set (except that of the path entry) share a common set of buffers whose size is that of the largest buffers specified.

2. When opening an alternate index and its base cluster through a NOUPDATE path, no other alternate index is opened, even if its a member of the same upgrade set (parameter NOUPDATE is explained on page 110). Therefore, only space for the path entry and the base cluster has to be provided unless the base cluster is opened with NOUPDATE.

3. If a base cluster is not processed via a path, but has an upgrade set assigned to it, space for the base cluster itself and for the upgrade set members (alternate indexes) has to be provided.

9.2.4 MINIMUM WORKING SET FOR A SINGLE KSDS (VS1)

The virtual storage requirements of VSAM are big, but it is designed to operate efficiently in a VS environment with relatively small Real Storage (Working Set) requirements.

The following conditions are assumed:

- One EXTENT for the data component
- SHAREOPTION(4) not specified (SHAREOPTIONs are described on page 95)
- The key is 4 bytes long
- One extent for the sequence set and the index set
- 2 data buffers and 1 index buffer (KSDS data set)
- The ACB and RPL are created in this sequence via GENCB, leaving the space allocation up to VSAM (ACB, RPL, and GENCB are VSAM macros and are described on page 217)

The minimum working set for a single VSAM data set (KSDS, ESDS, RRDS) can be calculated as follows :

- VSAM record management	36 K[1] PLPA
- (if loading/extending the file	12 K)
- Control blocks and channel program area	3.3 K
	39.3-51.3 K + VSAM buffers

[1] PLPA is shared by all partitions

If an alternate index is processed as an end-use object, that is, without its related base cluster, it is treated as a KSDS data set.

For each additional data set in the region, the following has to be added to this minimum :

- Control blocks and channel program area	3.3 K
- VSAM buffers	n K

9.2.4.1 WORKING SET FOR ALTERNATE INDEX PROCESSING (VS1)

If a path is established between an alternate index and its base cluster, the working set requirements for processing the path are as follows :

-	VSAM Record management	48 K[1]	PLPA
-	Path entry min.	3.4 K[2]	
-	Base cluster min.	3.3 K[2]	
-	Other alternate indexes in the upgrade set min.	3.4 K[2]	per each one
-	Buffers for the base cluster, the AIX path and each remaining alternate index with the UPGRADE attribute belonging to the base cluster (see also section 8.1.5)	m K	

[1] PLPA is shared by all regions
[2] includes channel program areas

Notes:

1. All alternate indexes of an upgrade set (except that of the path entry) share a common set of buffers whose size is that of the largest buffers specified.

2. When opening an alternate index and its base cluster through a NOUPDATE path, no other alternate index is opened, even if its a member of the same upgrade set (parameter NOUPDATE is explained on page 110). Therefore, only space for the path entry and the base cluster has to be provided unless the base cluster is opened with NOUPDATE.

3. If a base cluster is not processed via a path, but has an upgrade set assigned to it, space for the base cluster itself and for the upgrade set members (alternate indexes) has to be provided.

9.2.5 VSAM VIRTUAL STORAGE REQUIREMENTS (DOS/VS)

Although most of the storage required for VSAM and Access Method Services is in SVA, additional virtual storage is required in the user's address space for control blocks, buffers, Access Method Services, and if used, the ISAM interface routines.

The following information is based on DOS/VS Release 34. For further information see DOS/VS2 System Generation, GC33-5377-6.

- Access Method Services

Any function except BLDINDEX, EXPORTRA, RESETCAT	= 116K-166K
BLDINDEX (without internal sort area)	= 138K
EXPORTRA	= 414K
RESETCAT	= 238K

- Main storage requirements per opened catalog

Catalog with 3K buffer (nonrecoverable on 3330)	= 10K
Catalog with 8K buffer (nonrecoverable on 3330)	= 18K
Catalog with 3K buffer (recoverable on 3330)	= 14K
Catalog with 8K buffer (recoverable on 3330)	= 23K

- Catalog management control blocks (for catalog access)

OPEN/CLOSE	= 1K
LOCATE, UPDATE, DEFINE NONVSAM,DELETE SPACE, DELETE catalog, LISTCAT	= 5K
DEFINE (common)	= 6-7K
DEFINE catalog, ALTER	= 14K

- Record management control blocks (example)

  This is an example for a KSDS data set:

  1. Key length is 8
  2. Number of data buffers is 2 (2048)
  3. Number of index buffer is 1 (512)
  4. No IMBED, no KEYRANGEs
  5. Disk is IBM 3330

Total amount used for control blocks	= 4 K bytes
Total amount used for buffers (2*2048+1*512)	= 4608 bytes

9.2.6 VSAM VIRTUAL STORAGE REQUIREMENTS (MVS)

Although most of the storage required for VSAM and Access Method Services is in the PLPA, additional virtual storage is required in the user's address space for control blocks, buffers, Access Method Services, and if used, the ISAM interface routines.

The following information is based on VS2 Release 3.7. For further information see OS/VS2 Storage Estimates, GC28-0604-4.

- Access Method Services

Any function except BLDINDEX, EXPORTRA, RESETCAT	=	220K
BLDINDEX (without internal sort area)	=	170K
EXPORTRA	=	445K
RESETCAT	=	270K

- Main storage requirements per opened catalog

		CSA + SQA
Catalog with 3K buffer (nonrecoverable on 3330)	=	10K 1K
Catalog with 8K buffer (nonrecoverable on 3330)	=	18K 1K
Catalog with 3K buffer (recoverable on 3330)	=	18K 1K
Catalog with 8K buffer (recoverable on 3330)	=	24K 1K
VSAM LSQA requirement	=	4K

- Catalog management control blocks (for catalog access)

OPEN/CLOSE	=	1K
LOCATE, UPDATE, DEFINE NONVSAM,DELETE SPACE, DELETE catalog, LISTCAT	=	5K
DEFINE (common)	=	6-7K
DEFINE catalog, ALTER	=	14K

- Record management control blocks (example)

This is an example for a KSDS data set:

1. Key length is 8
2. Number of data buffers is 2 (2048)
3. Number of index buffer is 1 (512)
4. No IMBED, no KEYRANGEs
5. Disk is IBM 3330

Total amount used for control blocks	=	5 K bytes
Total amount used for buffers (2*2048+1*512)	=	4608 bytes

9.2.7 VSAM VIRTUAL STORAGE REQUIREMENTS (SVS)

Although most of the storage required for VSAM and Access Method Services is in the PLPA, additional virtual storage is required in the user's address space for control blocks, buffers, Access Method Services, and if used, the ISAM interface routines.

The following information is based on VS2 Release 1.7 with VSAM ICR. For further information see OS/VS2 SVS Independent Component: Planning for Enhanced VSAM, GC26-3869.

- Access Method Services

Any function except BLDINDEX, EXPORTRA, RESETCAT	= 204K
BLDINDEX (without internal sort area)	= 204K
EXPORTRA	= 444K
RESETCAT	= 390K

- Main storage requirements per opened catalog

Catalog with 3K buffer (nonrecoverable on 3330)	= 10K
Catalog with 8K buffer (nonrecoverable on 3330)	= 18K
Catalog with 3K buffer (recoverable on 3330)	= 14K
Catalog with 8K buffer (recoverable on 3330)	= 23K

- Catalog management control blocks (for catalog access)

OPEN/CLOSE	= 1K
LOCATE, UPDATE, DEFINE NONVSAM,DELETE SPACE, DELETE catalog, LISTCAT	= 5K
DEFINE (common)	= 6-7K
DEFINE catalog, ALTER	= 14K

- Record management control blocks (example)

  This is an example for a KSDS data set:

  1. Key length is 8
  2. Number of data buffers is 2 (2048)
  3. Number of index buffer is 1 (512)
  4. No IMBED, no KEYRANGEs
  5. Disk is IBM 3330

Total amount used for control blocks	= 5 K bytes
Total amount used for buffers (2*2048+1*512)	= 4608 bytes

9.2.8 VSAM VIRTUAL STORAGE REQUIREMENTS (VS1)

Although most of the storage required for VSAM and Access Method Services is in the PLPA, additional virtual storage is required in the user's address space for control blocks, buffers, Access Method Services, and if used, the ISAM interface routines.

The following information is based on VS1 Release 6. For further information see OS/VS1 Storage Estimates, GC24-5094-6.

- Access Method Services

Any function except BLDINDEX, EXPORTRA, RESETCAT	= 220K
BLDINDEX (without internal sort area)	= 170K
EXPORTRA	= 445K
RESETCAT	= 270K

- Main storage requirements per opened catalog

Catalog with 3K buffer (nonrecoverable on 3330)	= 10K
Catalog with 8K buffer (nonrecoverable on 3330)	= 18K
Catalog with 3K buffer (recoverable on 3330)	= 14K
Catalog with 8K buffer (recoverable on 3330)	= 23K

- Catalog management control blocks (for catalog access)

OPEN/CLOSE	= 1K
LOCATE, UPDATE, DEFINE NONVSAM,DELETE SPACE, DELETE catalog, LISTCAT	= 5K
DEFINE (common)	= 6-7K
DEFINE catalog, ALTER	= 14K

- Record management control blocks (example)

  This is an example for a KSDS data set:

  1. Key length is 8
  2. Number of data buffers is 2 (2048)
  3. Number of index buffer is 1 (512)
  4. No IMBED, no KEYRANGEs
  5. Disk is IBM 3330

Total amount used for control blocks	= 4 K bytes
Total amount used for buffers (2*2048+1*512)	= 4608 bytes

A.0 APPENDIX A. VSAM CATALOG MISCELLANEOUS

A.1 VSAM CATALOG KEYRANGES

A.1.1 THE HIGH-KEYRANGE (HKR)

As described in the previous section, each VSAM catalog is a KEYRANGE data set (normal Keyranges are described on page 91). The second Keyrange called the 'High-Keyrange' (HKR) or the 'True Name Section' is where the actual data set names, volume names, etc. reside (Keyrange X'40' - X'FF').
Each CI here is 512 bytes (the same as throughout the data and index components of the catalog) and contains 44 byte data set names and 3 byte LKR CI pointers. Each LKR CI pointer associated with each data set entry contains the number of a CI in the Low-Keyrange (LKR) (see section A.1.2) where the actual data and information with regard to the particular entry itself resides. Thus the HKR functions as an additional index to the LKR record.

Given a data set name the index component is searched to find the entry in the HKR. Once there, Catalog Management can pick up the LKR CI pointer multiply it by 512 (CI size) to get the RBA value, add it to the beginning of the LKR extent, and find the actual data relative to the data set name.

Catalog Management uses regular keyed direct VSAM Record Management processing to do I/O to the HKR. The HKR is subject to CI and CA splits (unlike the LKR) and therefore the catalog should be subject to periodic reorganization by the user.

A.1.2 THE LOW-KEYRANGE (LKR)

The LKR is actually the first extent of the VSAM catalog data space (Keyrange X'00' - X'3F'). It is where the body of information relating to data sets is stored. It consists of a series of pre-formatted 512 byte data CIs each of which contain the appropriate data set, volume, etc., information.
An entire 512 byte CI is wholly used to contain a catalog entry (if there is not enough room available, extension records are built). A KSDS cluster, for example occupies three 512 byte CIs (cluster, data and index entry).

The Catalog Self Describing Records (see next section A.1.4) reside in the first 15-20 (dependent on device type) CIs of the LKR. Catalog Management when it is opening the catalog accesses these records via its own channel program and builds the catalog control blocks needed. Control Interval processing (CNV) (see description on page 74) is used to do I/O to the rest of the LKR. Since CNV processing is done to a series of preformatted CIs, the LKR is not subject to CI and CA splits.

If a record needs to be retrieved, it can be done via the CI pointer retrieved from the HKR. If a record needs to be written, it is first written in the LKR. Then the appropriate HKR record pointing to it is written.

A.1.3 CATLOG HKR AND LKR LOGIC

As described in the previous sections, the catalog consists of 3 parts, the index, the HKR and the LKR. The index points to the HKR and the HKR points to the LKR (see figure 52). The HKR is used like a second index.

This structure was chosen for the following reasons:

1. As shown in figure 52, some records in the LKR contain CI number pointers pointing to each other. Due to these pointers, CI or CA splits are not allowed, since otherwise all pointers must be rewritten after a split.
2. KSDS allows direct access using an index to find a specific entry, so the advantages of a KSDS were included.

Since an ESDS structure (LKR) does not have an index and a KSDS data component allows splits, both these functions have been combined using a HKR where splits are allowed. Record Management routines are used to access the HKR via the index. Catalog Management routines convert the LKR CI pointer in the HKR record (via its own channel programs).

Preformatting: The catalog areas are preformatted in the different systems at different times (preformatting can take a considerable amount of time depending on the size of the catalog):

- MVS all catalog primary allocation space is preformatted when first entry is stored in catalog.
- DOS/VS,VS1,SVS one CA is preformatted the others when needed.

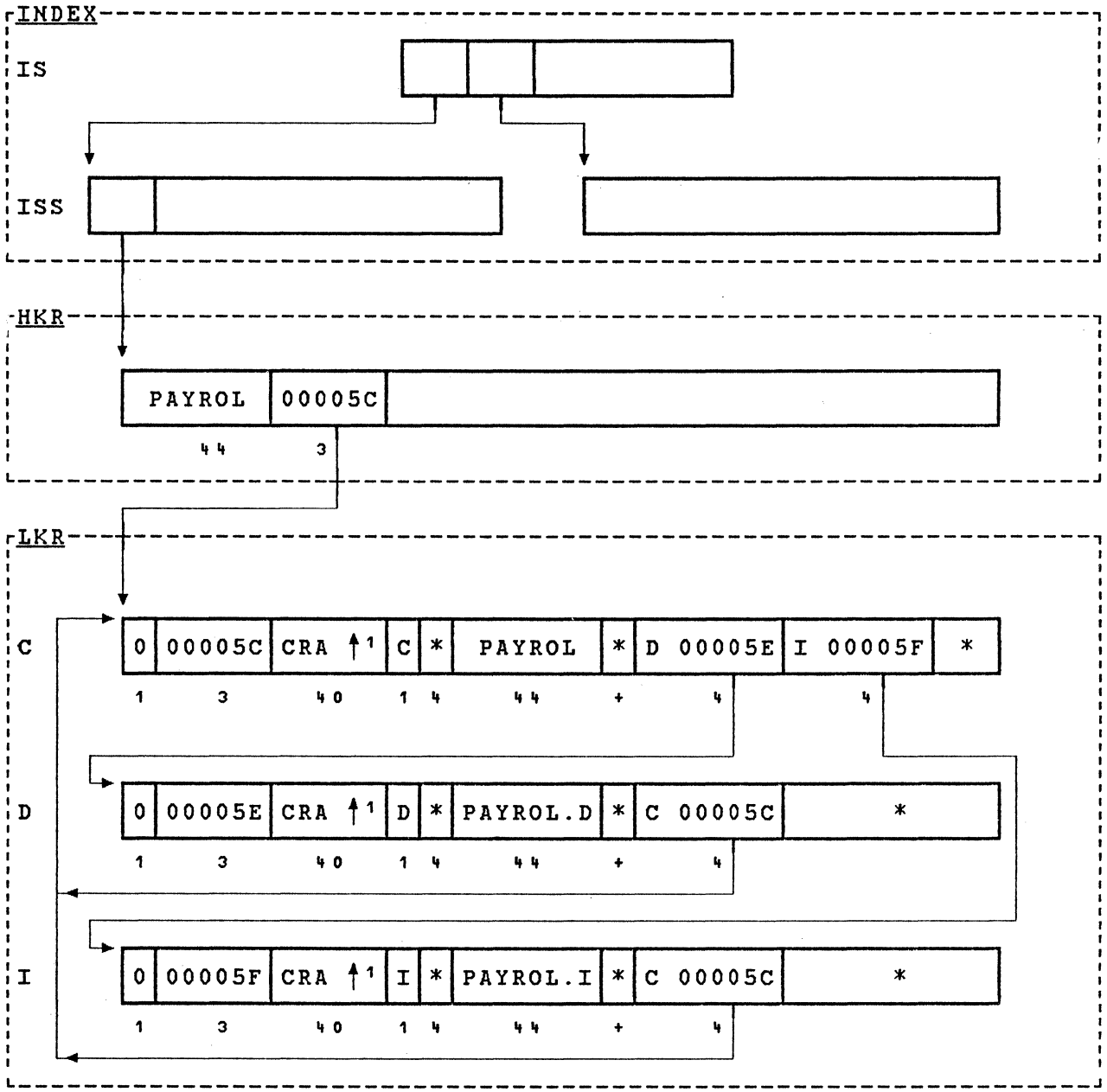
A.1.4 THE SELF DESCRIBING RECORDS

Each VSAM data set must be cataloged in a VSAM catalog. All the information with regard to the VSAM data set needed for open, close, EOV, allocation, etc, is kept in a VSAM catalog.

A VSAM catalog being a special KSDS, must be cataloged somewhere. One cannot 'catalog' a catalog. What is used instead are self-describing records in each catalog. These records describe the catalog itself, exactly like a catalog would describe a VSAM data set.

These records (described in the appropriate VSAM or Catalog Management PLMs) reside in the first few CIs of the LKR. They describe the space the catalog resides in, its extent per each Keyrange, where the free records are, what type of device the catalog resides on, etc. If these records are bad, then the catalog is bad and the message "Catalog Unavailable" is issued.

Figure 52 shows the relation of catalog entries (records) for a KSDS cluster 'PAYROL' with its data component 'PAYROL.D' and its index component 'PAYROL.I'. All CIs are 512 bytes in length. The index CIs have been shortened due to limited space.



C = Cluster entry
D = Data component entry
I = Index component entry
\+ = variable length
* = Miscellaneous fields (see PLM)
↑1 = these fields contain CRA references (recoverable catalog only)

HKR = High Keyrange
LKR = Low Keyrange
IS = Index Set
ISS = Index Sequence set

Figure 52. VSAM catalog structure

A.2 TIMESTAMPS, VTOC

The VSAM catalog records five different timestamps. The most important and the only one discussed in this manual is the 'volume timestamp' (most other timestamps are used internally for identification).

A.2.1 VOLUME TIMESTAMP

The purpose of this timestamp is to check that the catalog and its owned volumes are on the same level from the space allocation standpoint.

This timestamp is located in the Format 4 DSCB (see page 245, note 6) as well as in the volume record (see page 159).

Actually, this timestamp is recorded twice in the Format-4, at offset X'4C' (page 245 note 3) as well as at X'57'(see page 245 note 6). The former one (at X'4C') is used by Version 1 VSAM and IEHDASDR; it is completely ignored by the checking algorithm in Enhanced VSAM.

The volume timestamp update is treated differently for a recoverable catalog as compared to nonrecoverable catalog. In a recoverable catalog, when space on a volume is altered the first time after the CRA is opened, the timestamp in the Format 4 DSCB and the volume record in the VSAM catalog will be updated. Subsequent alteration in VSAM data space will not cause the volume timestamp to be updated. Any allocation change in a data space (DEFINE or DELETE), which changes the space bit map (this is a field in the volume record where VSAM records which VSAM data space tracks are free or occupied) changes the volume timestamp (nonrecoverable catalog only).

The volume timestamp checking algorithm in Enhanced VSAM is different from Version 1. In Version 1 VSAM the old timestamp field in the F4 DSCB (X'4C' - X'53) was checked against the catalog volume timestamp (see page 159) for equal or high. In Enhanced VSAM, the check is for exact equal at volume mount time. A mismatch will cause the UCBAMV bit not to be set. If there is no mismatch, this bit will be set ON ('1') by VSAM. This indicator will be checked by VSAM OPEN.

A timestamp mismatch can be encountered in VSAM usage and will cause serious operational problems. This out-of-sync condition is usually caused by restoring a back-level volume or catalog. Timestamp mismatch will prevent opening of all VSAM data set on that volume.

On page 254 the timestamp mismatch problem is discussed in more detail.

A.2.2 VTOC F1 AND F4 DSCB'S (LABELS)

The following is part of a printout of the VTOC taken after the LISTCAT ALL example (see section 7.9.2 on page 142).

DOS/VS example (input):

```
// JOB VTOC
// ASSGN SYS004,X'191'
// ASSGN SYS005,X'00E'
// PAUSE REPLY NO IF MSG 8V96D IS ISSUED
// EXEC LVTOC
/&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIMOLI2 EXEC PGM=IEHLIST
//SYSPRINT DD SYSOUT=A
//DD1      DD UNIT=3330,DISP=OLD,VOL=SER=WTVSAM
//SYSIN  DD *
   LISTVTOC DUMP,VOL=3330=WTVSAM
/*
```

The output format has been modified to fit on this paper size.

CONTENTS OF VTOC ON VOL WTVSAM

```
   0 . . . 4 . . . . 9 . . . .14 . . . .19 . / / .29 . . . .34 . . . .39 . . . .
44 . . . .49 . . . .54 . . . .59 . . . .64 . / / .74 . . . .79 . . . .84 . . . .89 . . . .94 . . . .
99 . . . 104 . . . 109 . . . 114 . . . 119 . / / 129 . . . 134 . . . 139          DSCB ADDR (CCHHR)
```

1. Format 4 DSCB (VTOC descriptor):

```
FORMAT 4 DSCB
-------------
      1
  040404040404040404040404040404040404040404040/ /040404040404040404040404040404
F400000003050245019400OE007708010000019B00133/ /00271C8BDE7F9BA51F90008000378BDE7F9BA51F900000000000
--                                                ----------------++-----+++++++++++++++++
 2                                                          3       4  5        6
000000000000010000000030000001100000000000000/ /00000000000000000000000          0000000301
```

2. Format 1 DSCB (VSAM catalog):

```
Z9999994.VSAMDSPC.T8BDE7F4.TAFA2AB8
-----------------------------------
                7
F1E6E3E5E2C1D400014D016363016D010000C9C2D4D6E/ /4040000000000000000008C000100000000000010C000000100
--                                                                    ----                --
 8                                                                     9                  10
0000336D000081000001000000030012000000000000/ /00000000000000000000000          000000030
```

3. Format 1 DSCB (VSAM data space):

```
Z9999992.VSAMDSPC.T8BDE7F5.T0C9DD78
-----------------------------------
               11
F1E6E3E5E2C1D400014D016363016D010000C9C2D4D6E/ /4040000000000000000008C000100000000000010C000000200
--                                                                    ----                --
12                                                                    13                  14
0000336D000081000004000000670012000000000000/ /00000000000000000000000          0000000304
```

The numbers are explained on the next page.

Legend:

1 = This line was added by the utility program to identify the Format-4 DSCB (Label).

2 = Format-4 identifier

3 = bytes 76-83 (X'4C'-'53').
'VSAM old timestamp' used by Non-Enhanced VSAM and IEHDASDR. This timestamp is updated but not checked in systems using Enhanced VSAM (DOS/VS ≥R.31, MVS ≥R.3.6, SVS with ICR, VS1 ≥R.4).

4 = Byte 84 (X'54') bit 0.
'VSAM ownership bit'. This volume is owned by a VSAM catalog. Whenever a VSAM catalog is defined or a VSAM catalog allocates space on a volume, bit '0' is set ON.

5 = bytes 85-86 (X'55'-'56').
Start of CRA (track offset in hex). 0037 = cyl 2 track 17 (dec).

6 = bytes 87-94 (X'57-'5E').
'VSAM volume timestamp'. This is the catalog/volume timestamp.

7 = bytes 0-43 (X'00'-'2B').
This is the identification of the VSAM catalog:
Z9999994 = the '4' at the end identifies this VSAM object as a VSAM catalog. In DOS/VS, SVS and VS1 this is the identification for a user catalog (the master catalog uses a '6' instead of a '4'). In MVS the master and user catalog identifier is '4'.
Txxxxxxx.Txxxxxxx
is the timestamp when this F-1 DSCB (Label) was created.

8 = Byte 44 (X'2C').
Format-1 identifier

9 = bytes 82-83 (X'52'-'53').
Data set organization (0008 = VSAM)

10 = Byte 93 (X'5D') bit 3.
'Data set security bit' (in OS/VS systems also 'OS/VS password protection bit'). This bit prevents other access methods from accessing this data space.

11 = bytes 0-43 (X'00'-'2B').
This is the identification of a VSAM data space:
Z9999992 = the '2' at the end identifies this VSAM object as a suballocatable VSAM data space.

12 = Byte 44 (X'2C').
Format-1 identifier

13 = bytes 82-83 (X'52'-'53').
Data set organization (0008 = VSAM)

14 = Byte 93 (X'5D') bit 3.
'Data set security bit' (in OS/VS systems also 'OS/VS password protection bit').
This bit prevents other access methods from accessing this data space.

A.3 HOW TO CALCULATE USED SPACE IN A VSAM CATALOG

As described in section 7.4 on page 102 the space for a catalog can be calculated using the worksheet of the Access Method Services manual. Sometimes, however it might be necessary to calculate the used space in a catalog as a base for defining a new catalog.

Normally the statistics fields show the number of records or used RBA-values, but for a catalog these fields in a LISTCAT output of a catalog are meaningless and cannot be used for space calculations.

The only possible way to find the number of occupied records (CIs) in the catalog 'Low-Keyrange' area (usually about 90% of the catalog space), is to analyze the catalog control record, CCR (see also VSAM Logic manuals).

The following is an example of how to print the CCR-record of a catalog to estimate the used space.

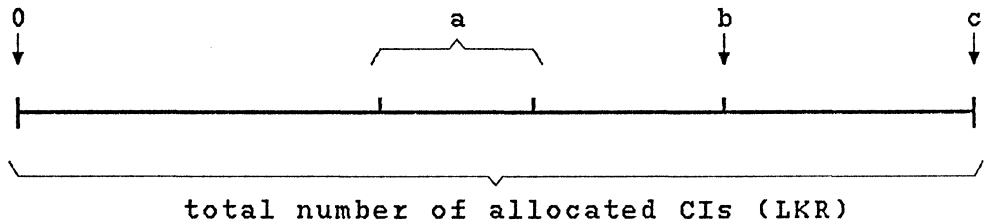
The format of the important CCR-record fields is:

```
X'00'-'2B'= key (44 bytes)
X'01'-'03'= 3, the CI number of this record
X'2C'     = X'D3' = 'L' (id of CCR record)
X'2D'-'2F'= (c)number of highest CI in the extent
X'30'-'32'= (b)number of next free CI (not previously assigned) CI
X'33'-'35'= (a)number of deleted (previously assigned but freed
              by deletion) CIs (they need not to be contiguous)
```

The approximate catalog size can be estimated by calculating the number of CIs used in the catalog Low-Keyrange, which is about 90% of the actual catalog size, using this formula:

Catalog Low-Keyrange = b - a + 1

The following is a graphic to show how the values 'a' - 'c' are used.



The CCR is the fourth CI record of any catalog and can be printed using Access Method Services commands as follows:

DOS/VS example (input):

```
// JOB PRINT UCAT (PARTIAL)
// DLBL UCAT,'PRIMER.UCAT1',,VSAM
// EXTENT SYS010,WTVSAM
// ASSGN SYS010,DISK,VOL=WTVSAM,SHR
// EXEC IDCAMS,SIZE=AUTO
         PRINT  INFILE (UCAT)        -
                SKIP (3)             -
                COUNT (1)
/&
```

OS/VS example (input):

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIMO562 EXEC PGM=IDCAMS
//STEPCAT  DD DSN=PRIMER.UCAT1,DISP=SHR
//UCAT     DD DSN=PRIMER.UCAT1,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
        PRINT     INFILE (UCAT/UCATMRPW)    -
                  SKIP  (3)                 -
                  COUNT (1)
/*
```

Output of Access Method Services (DOS/VS and OS/VS are similar):

```
LISTING OF DS -PRIMER.UCAT1

KEY OF RECORD - 0000000300000000000000000000000000000000000000000000000000000000000000000000000000000000//
0000  00000003 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....................//
0020  00000000 00000000 00000000 D300018F 00018D00 01660001 72000318 00000320 *..........L........//
0040  00000320 00000322 00000370 00000370 00000002 00000078 0000008C 0000008C *.....................//
0060  0000008E 0000008E 00000000 00000000 00000000 00000000 00000000 00000000 *.....................//
0080  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....................//
00A0  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....................//
00C0  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....................//
00E0  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....................//
0100  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....................//
0120  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....................//
0140  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....................//
0160  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....................//
0180  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....................//
01A0  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....................//
01C0  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....................//
01E0  00000000 00000000 00000000 00000000 00000000 00000000 00                 *.....................//

IDC0005I NUMBER OF RECORDS PROCESSED WAS 1
IDC0001I FUNCTION COMPL, HIGHEST COND.CODE WAS 0

IDC0002I IDCAMS PROC COMPL. MAX. COND.CODE WAS 0
```

Note: The output lines were truncated on the right side due to limited page size.

The approximate catalog size can now be calculated by adding the three logical parts:

- Low-Keyrange

  The number of occupied CIs in 'Low-Keyrange' is 40, since:

  b - a + 1 = X'18D' - X'166' = X'27' + 1 = 40

  Each CI is 512 bytes in length. Use the device characteristics table in section 8.2 on page 196 to determine the number of tracks:

  device type = 3330 = 20 * 512 bytes per track

  40 entries = 2 tracks for data in Low-Keyrange used

  Add 1 track per data CA (for the imbedded sequence set record).

  The data CA sizes in a catalog are as follows:

  2305, 3330, 3350 = 3 tracks (2 tracks + 1 track sequence set)
  2314, 3340 = 5 tracks (4 tracks + 1 track sequence set)

  The allocation units in a catalog are one data control area + one track.

  Total space used for Low-Keyrange = 3 tracks

- High-Keyrange

  The High-Keyrange size is about 10% (DOS/VS Rel.34 20%) of Low-Keyrange (minimum 1 allocation unit).

- Index set

  The index set size is 1 allocation unit (an allocation unit is 1 CA (3 or 5 tracks depending on device type) + 1 track).

Total calculation summary for the catalog:

- Low-Keyrange	= 3 tracks
- High-Keyrange	= 3 tracks
- index set	= 3 tracks
Total catalog space used	= 9 tracks

A.4 CATALOG RECOVERY

A.4.1 INTRODUCTION

Unlike nonVSAM, each VSAM data set must be cataloged in a VSAM catalog. In order to process a VSAM data set, that data set's catalog entry must be accessible. A damaged catalog will prevent accessibility of VSAM data sets even though the data may be in perfectly good condition.

In an MVS environment, a damaged catalog can also affect nonVSAM users. It is not uncommon to find an MVS installation having a VSAM catalog that contains several thousand nonVSAM entries. Damage to such a catalog does not normally prevent accessibility of nonVSAM data sets, however, the changes that have to be made in JCL procedures and VS2 TSO C-lists may impact the operation of an installation.

A much more common problem than the damaged catalog is an out-of-sync condition where the catalog or its owned volumes is down-level. A down-level situation is usually created by restoring an earlier version of the catalog or its owned volumes (see also description of DUMP/RESTORE on page 255).

Such out-of-sync conditions will usually prevent opening of the VSAM data set on that particular volume. For all practical purposes, this condition is just as unacceptable as the damaged catalog as far as the particular application is concerned.

With Enhanced VSAM a recovery facility is available for VSAM catalogs. It enables VSAM data sets and catalog entries for both VSAM and nonVSAM data sets to be recovered in the event that a VSAM catalog cannot be read for any reason. Use of the recovery facility for a VSAM catalog is specified via the RECOVERABLE attribute. Use of this facility is optional.

When a catalog is recoverable, catalog information for each data set described by the catalog is recorded in both the catalog and usually in the catalog recovery area (CRA) on the first volume of the data set on which a data space is defined. Thus, each volume identified by a recoverable catalog contains its own catalog recovery information.

A CRA is automatically reserved on a volume by VSAM when the first data space allocation occurs for the volume. Initially, one cylinder is allocated. If this space becomes filled, one additional cylinder is allocated each time a cylinder is filled. A CRA can contain a maximum of 16 cylinders.

The location of the CRA is specified in the Format-4 DSCB (Label) for the volume and is not indicated in the associated catalog (see page 49 and 245). Whenever an entry in a recoverable catalog is updated, the corresponding catalog information in the catalog recovery area of the affected volume is also automatically updated.

This means the affected volume must be mounted.

In DOS/VS systems if the first data space on a volume is a catalog with no DATA or INDEX parameter specified, the CRA is not suballocated from the data space. Therefore, the space value specified in the EXTENT control statement must be increased by 1 cylinder for the CRA.

In OS/VS systems if the first data space on a volume is a unique data space the CRA is not suballocated from the data space. The OS/VS DADSM routines try to dynamically allocate one additional cylinder for the CRA. For catalogs, the CRA is always suballocated from the catalog data space.

If the catalog is down-level, a current copy of the entries are still available through the CRA. On the other hand, if the volume is down-level (and, therefore, also the data set and the CRA), the data and the associated catalog entries can be unloaded (using the CRA) and reloaded back into the catalog. This will synchronize the catalog with its owned volumes.

Two points are worth noting:

1. The CRA, although containing duplicated records of the catalog, cannot be processed as a catalog.

2. With Enhanced VSAM, the synchronization of the catalog with its owned volume(s) will require data movement via the EXPORTRA/IMPORTRA commands (described later).

With DOS/VS Release 33, MVS Data Management Selectable Unit, SVS with VSAM ICR, and VS/1 Release 6 a new Access Method Services command, RESETCAT, is introduced. This command will take the entries in the CRA and use them to update (insert, delete, replace) the catalog. This synchronization process does not require data movement, thus providing a faster way to recover the catalog than EXPORTRA/IMPORTRA.

In DOS/VS nonVSAM data sets cannot be defined in a recoverable catalog.

A.4.2 CRA OPEN AND CLOSE

A CRA is opened when it is first needed (when CRA records have to be changed, added or deleted), and will remain open as long as the user catalog is open. Because of this fact, the CRA volume must remain mounted during that period of time (until the user catalog is closed).

The CRA is not referenced when the catalog is read.

A.4.3 RECOVERABLE OR NONRECOVERABLE CATALOG

A.4.3.1 MASTER CATALOG

It is the consensus today that the master catalog should be kept 'pure'. In other words, the master catalog should contain only the following kinds of data set entries:

- User catalog connector entries

Additional entries for MVS systems only:

- ALIAS entries
- CVOL connectors and ALIASes
- System data sets such as LINKLIB
- System work data sets such as Page Space

The only way that a master catalog can be kept 'pure' and not be cluttered with user data sets is through the implementation of a VSAM password at the update level (see description on page 52).

With a 'pure' master catalog established, the only advantage that we can conceive in using a recoverable catalog is the facility of EXPORTRA ENTRIES. This may be useful in a VS2 TSO environment where it is not unusual to find hundreds of ALIASes (userid) relating to the user catalog. The EXPORTRA/IMPORTRA command will allow us to back up and restore all the entries in a straightforward manner.

A.4.3.2 USER CATALOG

For the user catalog, the selection of recoverable versus nonrecoverable is not as clear-cut. Before a user catalog is defined, the following questions should be addressed:

1. Is recovery of the catalog an important issue?

   It is conceivable that VSAM data sets are used for temporary files, which minimize the importance of catalog recovery.

2. Is good catalog performance important to the installation or application?

   The catalog to be used for VS2 TSO data sets may want to consider nonrecoverable for performance reasons. Performance tests have shown that defining a KSDS took considerably longer with a recoverable than a nonrecoverable catalog.

3. Are the missing Access Method Services functions for a recoverable catalog important to the installation?

   The following functions are (partially) implemented for recoverable catalog:

	Nonrecoverable	Recoverable
UNLOAD/RELOAD via REPRO[1]	yes	partially supported
DELETE NOSCRATCH[2]	yes	no
Catalog to catalog copy[2]	yes	no

   Legend:

   [1] Unload/reload of a recoverable catalog via REPRO will work except that the CRA will be ignored in both directions.

   [2] These commands are supported in MVS only.

4. Are the added functions in EXPORTRA, IMPORTRA and RESETCAT playing a role in selecting recoverable catalog?

   We are referring to features that are not directly related to recoverability. An example of this is EXPORTRA allowing you to unload a nonVSAM entry and all its associated ALIASes, whereas EXPORT does not support nonVSAM data sets.

The most important question to raise is the first one. If recovery of the catalog and synchronization of the catalog with its owned volumes are germane to the operation of the installation, then recoverable catalog is the only way to go.

A.5 CATALOG BACKUP

Generally, there are two ways to backup a VSAM catalog:

1. Unload/reload the catalog using REPRO
2. Dump the catalog volume with utilities:
   - DOS/VS: BACKUP/RESTORE (see utilities manual GC33-5381)
   - OS/VS : IEHDASDR (see utilities manual GC26-3901 (VS1) or GC26-3902 (VS2))
     DRWDASDR (Program Product 5740-UT1)

A.5.1 REPRO (CATALOG UNLOAD/RELOAD)

REPRO can be used to unload and reload a VSAM catalog. The unloaded version called portable catalog here, can be reloaded back into an existing or new catalog. The new catalog must

- have the same name, same devicetype, same volume serial number as the catalog prior to unload.

- be large enough in its primary allocation to accommodate all the extents of the portable catalog.

Insertion, deletion, and replacement will take place for the target catalog. The content of the target catalog after reload will have the same content as the portable catalog with one exception; when reloading into a new catalog, the volume record of the target catalog's volume is retained.

Unload/reload will ignore the CRA in both directions and essentially will treat recoverable and nonrecoverable catalogs alike. When reloading into a new recoverable catalog, the CRA which is empty except for self-defining records, will be out of step with the catalog. RESETCAT should be executed immediately after reload is performed (since RESETCAT deletes all entries associated with the catalog volume, all data sets on the catalog volume should be saved with EXPORTRA before the RELOAD CATALOG operation is started).

In addition to ignoring the CRA, reload will not attempt to synchronize the Format-4 'volume timestamp' and the catalog 'volume timestamp' (see section A.2 on page 244).

This will create an out-of-sync condition that will preclude the opening of VSAM data sets on the catalog volume.

If the catalog is to be opened as a data set, as in the cases of LISTCRA COMPARE and RESETCAT, VSAM OPEN will fail with a volume timestamp mismatch. The only way to get around the problem is to SUPERZAP the Format-4 'volume timestamp' timestamp (see page 245 note 6) (for DOS/VS systems use IKQVDU, see description on page 179; for OS/VS systems use H/AMASPZAP, see description on page 270).

This superzapping undoubtedly is a violation of VSAM integrity, but if the action is confined to LISTCRA and RESETCAT, there should be no unpredictable and undesirable results. It should be pointed out that the above discussion is applicable only to the catalog volume. Timestamp out-of-sync condition on a catalog volume without a catalog will be corrected by RESETCAT.

Even with the restrictions of not updating the CRA and F4DSCB, any out-of-sync condition can be corrected by RESETCAT. Therefore, unload/reload is a viable method in catalog backup and restore (since RESETCAT deletes all entries associated with the catalog volume, all data sets on the catalog volume should be saved with EXPORTRA before the RELOAD CATALOG operation is started) or backup with catalog.

One final point regarding reload: if LISTCAT ALL is executed in the same job step as reload, probably only a part of the catalog is listed since the catalog has not yet been closed.

Examples on how to use REPRO for catalog unload/reload are included on pages 171, 172.

A.5.2 DUMP/RESTORE (OS/VS UTILITY)

A.5.2.1 DUMP/RESTORE VOLUMES (OS/VS)

IEHDASDR dump obviously will take more time than unload, especially on a device such as 3350. 3850 users should also be aware that 3330V (which is a virtual 3330) is an unsupported device for IEHDASDR.

IEHDASDR restore has the distinct advantage of providing the same level of F4DSCB and CRA as expected by VSAM catalog management. On the other hand, IEHDASDR will restore all data sets to an earlier level which may not be acceptable. Furthermore, the time involved in restoring a pack may be prohibitive in urgent recovery situations. Its shortcoming may outweigh its advantages.

As VSAM objects are protected against any nonVSAM access method by setting the 'data set security bit' ('OS/VS password protection bit') in the F1-DSCB (Label) X'5D' to 'ON' (not to be mixed up with VSAM password protection, which is stored in the VSAM catalog only), special setup is necessary for volumes containing VSAM objects when using IEHDASDR or DRWDASDR (PP 5740-UT1) (see the appropriate manuals).

The following are examples on how to use IEHDASDR:

The first jobs dumps the volume WTVSAM. This job was executed after the LISTCAT ALL example (see section 7.9.2 on page 142).

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIMODU2 EXEC PGM=IEHDASDR
//STEPCAT DD DSNAME=PRIMER.UCAT1,DISP=OLD
//SYSPRINT DD SYSOUT=A
//DISK1    DD UNIT=3330,DISP=OLD,VOL=SER=WTVSAM
//TAPE1    DD VOL=(,RETAIN,,,SER=057454),UNIT=3400-6,
//         LABEL=(1,SL),
//         DSN=DUMPVS1,DISP=(NEW,PASS)
//SYSIN  DD *
 DUMP FROMDD=DISK1,TODD=TAPE1
/*
```

The output is as follows:

```
IEH806I  DUMP TO    DDNAME=TAPE1    IS COMPLETED
IEH839I  HIGHEST RETURN CODE ENCOUNTERED WAS 00
```

Note:

The //STEPCAT DD statement is necessary when a VSAM object resides on the volume (otherwise a 913 ABEND is issued).

The operator was prompted to enter the VSAM catalog master password (the catalog was defined with passwords (see page 102).

The second Job restores the volume.

It is assumed that there is a valid VSAM catalog with the name PRIMER.UCAT1 on the volume. For the JCL specification see the appropriate Utilities manual.

```
//PRIMER   JOB WTSC,IBM,MSGLEVEL=1
//PRIMORE2 EXEC PGM=IEHDASDR
//STEPCAT DD DSNAME=PRIMER.UCAT1,DISP=OLD
//SYSPRINT DD SYSOUT=A
//TAPE1    DD VOL=(,RETAIN,,,SER=057454),UNIT=3400-6,
//         LABEL=(1,SL),
//         DSN=DUMPVS1,DISP=OLD
//SYSIN  DD *
 RESTORE TODD=STEPCAT,FROMDD=TAPE1,PURGE=YES
/*
```

The output is as follows:

```
IEH806I  RESTORE TO DDNAME=STEPCAT  IS COMPLETE. VOLUME SERIAL NO.=WTVSAM
IEH839I  HIGHEST RETURN CODE ENCOUNTERED WAS 00
```

A.6 CATALOG DEVICE CONVERSION

The only utility currently available to do direct device conversions of VSAM catalogs is REPRO 'COPYCAT' (see section B.3 on page 272). This however is an MVS exclusive and only works for nonrecoverable catalogs. The following also discusses other alternatives with RCATs and other SCPs.

If a catalog needs to be reorganized, or its allocation needs to be changed, a catalog copy function needs to be performed. The considerations involved are generally the same as device conversion and are mentioned below.

A.6.1 NONRECOVERABLE CATALOG DEVICE CONVERSION (MVS)

REPRO 'COPYCAT' (section B.3 on page 272) can be directly used to move a catalog from one device to another. A new empty catalog can be defined on the new volume and the old catalog copied into it. The new catalog must have a different name, be on a different volume and be large enough to contain all the extents of the old catalog. Additionally both catalogs must be nonrecoverable.

Following the 'COPYCAT' function the user must EXPORT DISCONNECT the original catalog and delete the cluster entries that described the original catalog and which now resides in the new catalog. This function will make the old catalog space available, clean up the volume's F4 DSCB and write a new Volume Record for the old volume in the new catalog. The new catalog now owns the old volume.

If the Master Catalog (MCAT) device is being converted or copied, the SYSCTLG member of the SYS1.NUCLEUS must be changed to point to the new MCAT device, followed by an IPL (see description on page 81). Subsequent to the IPL the DELETE and EXPORT DISCONNECT action must be taken.

When 'COPYCAT' is being used for device conversion, the user must define a new catalog with a different name than the original and it must also reside on a different volser number. The reasons for this are because the DEFINE process of the new catalog involves writing an entry in the MCAT which points to the new catalog. This entry cannot obviously have the same name as the original source catalog. Since the source and target volumes must also be mounted when the actual copying is being done, the target volume must also have a different volser. If the user wants to retain the same volser and/or retain the same catalog name, there is no easy straightforward utility command available to accomplish the task.

What has to be done is the execution of two sets of 'COPYCAT'. The first step would be a 'COPYCAT' to an interim catalog on any scratch volume. The entire 'COPYCAT' process including EXPORT DISCONNECT and DELETE CLUSTER of the original sources must be executed here. The second step would be the definition of the new catalog with the same catalog name on the new volume of the desired devicetype and bearing the same volser name, followed by the 'COPYCAT' function from the interim catalog on the scratch volume to the new devicetype.

An additional consideration is the question of the UCAT associated ALIAS names. These ALIAS names do not reside in the UCAT itself. They are associated with the user catalog record (U record) in the MCAT that points to the UCAT volume.

When the catalog copy is done and the EXPORT DISCONNECT command is issued against the original source catalog (and this must be done), it deletes the U record and associated ALIAS entries from the MCAT. The new version of the UCAT is left without its ALIAS names. As a consequence of this it is generally a good idea to do a LISTCAT of the original source catalog's U entry in the MCAT, before 'COPYCAT' is done. The output of this run can be directed to SYSPRINT so a hardcopy of the ALIAS names is produced or it can be directed to some tape or disk data set for subsequent retrieval.

Subsequent to the 'COPYCAT' procedure, the user, using the LISTCAT output, can either manually, (via the Access Method Services DEFINE ALIAS command), or via a user-written program, (that retrieves the ALIAS names from the output data set of LISTCAT and generates DEFINE ALIAS commands) rebuild the ALIAS names for the new UCAT.

A.6.2 NONRECOVERABLE CATALOG DEVICE CONVERSION

REPRO 'COPYCAT' not being available with DOS/VS, SVS, and VS1, the only choice the user has for device conversion is via EXPORT/IMPORT of individual data sets. The REPRO Unload/Reload function available here will not help since it has a requirement that the reload must be done to a catalog with the same name and devicetype.

Each data set in the catalog being converted must be backed up. This can be done either via REPRO or EXPORT. Once everything is out of the catalog being converted, it can be deleted and removed. A DEFINE for the new catalog must be issued, on the new device type, and the data set entries moved back into it. If REPRO was used for backup, the data set can be manually redefined in the new catalog followed by the REPRO of the data. If EXPORT was used the data set can now be imported. IMPORT, generally speaking, is nothing more than a DEFINE followed by a REPRO issued by Access Method Services.

With Enhanced VSAM (DOS/VS Rel. 31, VS1 Rel.4 and up, SVS Rel. 1.7 with ICR, and MVS Rel. 3.0 with ICR and up) a data set can be predefined and then IMPORTed into. If the data set, as it is defined in the catalog is empty (IMPORT checks this) the existing definition in the catalog will be used. If the data set is indicated as nonempty in the catalog and is being IMPORTed into, IMPORT will issue a delete, followed by a DEFINE based on information on the portable tape data set produced by EXPORT, and then REPRO the data set in.

During the IMPORT process, since sequential VSAM I/O is being done, any KSDS being imported will be reorganized. The important point to remember here is that some of the important performance options (such as FREESPACE) originally specified for the data set should probably be changed. If for example FREESPACE (50 50) was originally specified for the data set in anticipation of insert activity subsequent to the original load process, since these records probably now do exist in the data set, the FREESPACE parameter should be reduced. Since the REPRO or IMPORT functions are essentially reload functions, users should make a point of rechecking the parameters that are specifiable at DEFINE time and make the appropriate changes (with IMPORT the FREESPACE value can only be changed if the data set is imported into an empty cluster).

Note: Since device types have different cylinder capacities the space parameters must be checked for accuracy on the new device type.

Since the data sets involved are being totally reloaded into the new catalog the catalog statistics information is also changed. It is as if we have a new data set with no accumulated statistical information available.

Since EXPORT/IMPORT does not support nonVSAM data sets, any nonVSAM data sets cataloged in the old catalog must be manually recataloged.

A.6.3 RECOVERABLE CATALOG DEVICE CONVERSION

Prior to the availability of the Access Method Services RESETCAT command, the only tool available to do this kind of conversion were either EXPORT/IMPORT or EXPORTRA/IMPORTRA. The consideration relative to these commands are the same as was discussed in the previous section. EXPORTRA/IMPORTRA essentially do the same as EXPORT/IMPORT but they do it through the catalog entries in the CRAs on each owned volume.

The major differences are as follows:

- With EXPORTRA one can point to a certain CRA and issue a EXPORTRA ALL command rather than individually EXPORT or EXPORTRA each entry. This involves, of course, less work by the user.

- EXPORTRA also supports nonVSAM names cataloged in CRAs and will also move these to the portability tape (or disk). Unlike EXPORT/IMPORT, the user does not have to manually reenter the nonVSAM entries; IMPORTRA can be used instead.

- IMPORTRA always does a delete followed by a redefine. With IMPORT if the existing catalog entry for the data set being imported indicated that it was empty, the existing definition would be used. This is not the case with IMPORTRA.

- If an EXPORTRA ALL was originally coded, there is no tool available to do individual IMPORTRA entries (DSN). It is an all or nothing proposition.

- Both EXPORTRA and IMPORTRA only support recoverable catalogs which are only available in Enhanced VSAM. Until Enhanced VSAM support became available in SVS (with ICR), none of what is discussed here was available to SVS users.

  The Access Method Services command RESETCAT became available in DOS/VS Rel. 33, in VS1 Rel.6, in SVS with the VSAM ICR, and in MVS via SU 8. Its primary function is to rebuild a catalog through the duplicated entries in the owned volume CRAs. Using this information a catalog is rebuilt.

  Although not designed to be used as a conversion tool, it can be used to do so. If the RESETCAT function is directed to a new catalog, the entries in this catalog are rebuilt via the CRA contents. RESETCAT cannot be used against an MVS live master catalog. It requires the exclusive control of the catalog being reset which is not possible in the case of the MVS MCAT.

  The steps involved in using RESETCAT to do device conversion for a recoverable user catalog are listed below. The new catalog can be on a different devicetype, different volume serial number and can also have different space allocation values. The only restriction is that it must have the same name as the catalog that originally produced the CRAs involved.

Resetting into a new RCAT involves the steps listed below. As an example let us assume that a UCAT called CAT1 which is on volume OLDVOL is being moved to volume NEWVOL. CAT1 owns volume VOL1 and VOL2 in addition to OLDVOL.

Action	Explanation
1. Using IEHDASDR (FASTCOPY for DOS/VS) backup all volumes involved.	If an error condition occurs during RESETCAT, then the volumes can be restored.
2. LISTCAT ENTRIES (CAT1) ALL CAT (master catalog name) (OS/VS only).	List the user catalog entry in the MCAT to get a list of all its ALIAS names.
3. EXPORT DISCONNECT CAT1.	Deletes the MCAT entry for CAT1. Also deletes ALIAS names for CAT1.
4. DEFINE new catalog called CAT1 which is recoverable on volume NEWVOL.	
5. RESETCAT into CAT1 on NEWVOL pointing to the CRAs on VOL1, VOL2 and OLDVOL.	Rebuild CAT1 based on the information in CRAs on all owned volumes, we now have a new RCAT on NEWVOL.

At the completion of this process, the user should proceed to thoroughly check the new catalog. The space which the old catalog occupied on OLDVOL is now available for suballocation and is owned by the new catalog.

A.7 CATALOG REORGANIZATION

The High-Keyrange portion of a catalog could require periodic reorganization. It currently consists of 10% (DOS/VS Rel.34 20%) of the total allocated space of a catalog and it is subject to CI and CA splits. The reorganization of the catalog essentially involves the rewriting of all its contents. This can be done if a new catalog is defined and a recent backup copy of the catalog is restored into it.

In MVS the REPRO 'COPYCAT' function can be used to copy a nonrecoverable catalog into another catalog (see section B.3 on page 272). The disadvantages of 'COPYCAT', as described earlier, is that the new catalog will have a new name unless 'COPYCAT' is executed twice with the first execution producing an interim version of the catalog.

REPRO Unload/Reload, if the reload is done to a new catalog, can be used to reorganize the catalog. It requires that the catalog being reloaded have the same name and device type and volser as the

originally unloaded catalog. It also works with RCATs as opposed to 'COPYCAT' that does not. It can also be used to rebuild a catalog if it has run out of extents or if the user wants to consolidate all of the extents into one extent (for UNLOAD/RELOAD considerations see also section A.5.1 on page 254).

It is always necessary to insure that the particular catalog involved in an Unload/Reload operation is quiesced. If this is not done, since a backup/restore operation is involved, out of sync conditions could arise that could make either the catalog or some data sets unusable.

A.7.1 REORGANIZING A NONRECOVERABLE CATALOG

The steps are as follows:

	Action	Explanation
1.	LISTCAT ALL the UCAT	To have a list of catalog entries.
2.	LISTCAT ENTRIES (UCATname) ALL CAT(master catalog name) (OS/VS only).	To get the catalog ALIAS names.
3.	EXPORT all of the VSAM data sets that are wholly or partially contained on the catalog volume.	
4.	Unload the catalog to tape or disk using REPRO.	
5.	EXPORT DISCONNECT the catalog.	To remove its name and ALIAS names from MCAT.
6.	ALTER REMOVEVOLUMES on the catalog volume to clean up the volume.	Function is not available in old VSAM. Volume clean up can be done via super-zapping OFF the VSAM ownership-bit in the F4 DSCB followed by IEHPROGM SCRATCH.
7.	DEFINE new catalog on the same volume, with same name. Allocation amounts could be larger.	
8.	Reload the catalog (REPRO).	
9.	IMPORT all data sets EXPORTed in step 2 above.	
10.	Manually restore the ALIAS names for the UCAT (OS/VS only).	

A.7.2 REORGANIZING A RECOVERABLE CATALOG

Reorganizing a recoverable catalog involves some additional steps, since Unload/Reload opens a catalog as a data set and is not aware of CRAs.

	Action	Explanation
1.	LISTCAT ALL user catalog.	To have a list of catalog entries.
2.	LISTCAT ENTRIES (UCATname) ALL CAT(master catalog name) (OS/VS only).	To get a list of the UCATs ALIAS names out of the master catalog.
3.	EXPORTRA ALL out of CRA on catalog volume.	To capture all of VSAM and nonVSAM data sets on the catalog volume.
4.	EXPORTRA ENT (multivolume data set) of data sets that wholly or partially reside on catalog volume.	Get the multi volume data set names from step 1 above.
5.	Unload the catalog to tape.	Use Access Method Services REPRO.
6.	EXPORT DISCONNECT catalog from master catalog.	
7.	ALTER REMOVEVOLUMES catalog volume.	Cleans up pack and deletes all VSAM information.
8.	DEFINE new RCAT on same volume.	Could have larger extents.
9.	Reload into new catalog.	Use Access Method Services REPRO.
10.	IMPORTRA all data sets exported in steps 2 and 3 above.	Restores all VSAM and nonVSAM data sets.
11.	Rebuild the ALIAS names for UCAT.	

A.8 CHANGING THE ALLOCATION OF A NONRECOVERABLE CATALOG.

It is always a good idea to give a catalog the capability of extending into secondary extents if it needs to do so. A surprisingly large number of VSAM catalog users have not done so.

Once a decision is made to reorganize a catalog or to redefine it with secondary allocation amounts specified, the user faces the problem of what to do with possibly a large number of ALIAS names associated with the catalog. These ALIAS names will be deleted when the UCAT is EXPORT DISCONNECTed and can always be manually reentered in the MCAT, but if quite a few of these are involved (such as in a TSO environment) the process could take a long time. A user could always write a utility that would take LISTCAT output and generate DEFINE ALIAS commands (OS/VS only). Failing this, the process mentioned below can be used for nonrecoverable UCATs.

Since the process described below involves backup and restore of the MCAT and the UCAT, the access to these objects should be absolutely minimized. Any changes made to these after they have been unloaded will not be reflected in the catalogs after they have been restored. Of particular interest, in the case of the MCAT, are MVS PAGE data set and SYS1.STGINDEX. Activity against these data sets should be minimal (hence a quiesced standalone environment) to prevent their extension between the MCAT unload and subsequent reload.

Action	Explanation
1. Quiesce the system.	The following steps involve MCAT backup/restore. It is necessary to have a stand-alone quiesced system to do the following.
2. Backup MCAT and UCAT volumes.	For recovery purposes if an error condition occurs.
3. Unload MCAT to tape.	This step copies all of the UCAT's ALIAS names to tape.
4. EXPORT all VSAM multi or single volume data sets on UCAT volume.	
5. Unload UCAT to tape.	
6. EXPORT DISCONNECT UCAT.	Deletes UCAT entry and its ALIAS names from MCAT.
7. ALTER REMOVEVOLUMES UCAT volume.	Cleans up UCAT volume.
8. DEFINE new UCAT.	Should have secondary allocation specified and be larger than all extents of original.
9. Reload into UCAT.	Restores UCAT to original contents and reorganizes HKK.
10. IMPORT data sets EXPORTed in Step 5.	
11. Reload MCAT.	Reloads MCAT and restores UCAT's original record and its associated ALIAS names.

A.9 CHANGING THE NAME OF A VSAM CATALOG

The ALTER NEWNAME command of Access Method Services does not support VSAM catalog name changes. If a catalog name is specified on this command, catalog management checks the old name against the catalog name and if they match, the command will be rejected. The primary reason for this rejection is for integrity purposes. Catalog management cannot allow a catalog name change if there is a possibility that it might be shared with another region, address space or CPU.

Currently the only means available to change the catalog name, is via its deletion and redefinition. The user must of course capture all the entries in the catalog, via EXPORT, EXPORTRA or REPRO, essentially emptying it out, prior to deleting and redefining it. This is a very time consuming task. Additionally if the catalog is nonrecoverable, EXPORT does not support nonVSAM entries either, so subsequently these entries must be manually recataloged. (EXPORTRA for recoverable catalogs does support nonVSAM entries.)

In MVS, with nonrecoverable catalogs, 'COPYCAT' can be used simply to copy the catalog into a new one with the new desired name. 'COPYCAT' however is not available in DOS/VS, SVS, VS1 nor does it support recoverable catalogs.

B.0 APPENDIX B. OS/VS CATALOG CONSIDERATIONS

B.1 USING OS/VS SERVICE AIDS AND UTILITIES WITH VSAM

B.1.1 PRINTING A VSAM CATALOG WITH OS/VS UTILITIES

B.1.1.1 PRINT PARTS OF A CATALOG WITH IEHDASDR/DRWDASDR

The IEHDASDR/DRWDASDR utility can be used to print all catalog extents including index and sequence sets.

If the volume is owned by a user catalog, a STEPCAT or JOBCAT DD-statement must be provided.

- The output of LISTCAT (starting on page 142, contains the catalog extents as follows (see also description on page 148).

```
section:   CLUSTER ------- PRIMER.UCAT1
group:     DATA ------- VSAM.CATALOG.BASE.DATA.RECORD
```

  - The first 'VOLUME' group (describing the 'Low-Keyrange') contains the beginning of the catalog:

```
EXTENTS   LOW-CCHH----X'00010000'
```

  - The second 'VOLUME' group (describing the 'High-Keyrange') contains the end of the catalog (if no extents exist):

```
EXTENTS   HIGH-CCHH---X'00020010'
```

- To print the catalog use the following job:

```
//LIST        EXEC      PGM=IEHDASDR
//STEPCAT     DD        DSN=PRIMER.UCAT1,DISP=SHR
//SYSPRINT    DD        SYSOUT=A
//DD1         DD        UNIT=3330,VOL=SER=WTVSAM,DISP=SHR
//SYSIN       DD        *
     DUMP FROMDD=DD1,                                          X
          TODD=SYSPRINT,                                       X
          BEGIN=00010000,                                      X
          END=00020010
/*
```

B.1.1.2 PRINT PARTS OF A CATALOG WITH A/HMASPZAP

The AMASPZAP (MVS, SVS), or HMASPZAP (VS1) program can be used to print catalog extents, including index and sequence sets.

A STEPCAT or JOBCAT DD-statement must be provided.

- Obtain the begin and end of the catalog as described in section B.1.1.1 on page 267).

```
LOW-CCHH----X'00020000'
HIGH-CCHH---X'00020020'
```

- Execute utility IEHLIST with the LISTVTOC function (see page 245).

  The printout shows the catalog data space name as follows:

```
Z9999994.VSAMDSPC.TXXXXXXX.TYYYYYYY
```

- To print the catalog the following job may be used:

```
//LIST      EXEC   PGM=AMASPZAP
//STEPCAT   DD     DSN=PRIMER.UCAT1,DISP=SHR
//SYSPRINT  DD     SYSOUT=A
//SYSLIB    DD     DSN=Z99999994.VSAMDSPC.TXXXXXXX.TYYYYYYY,
//                 UNIT=3330,VOL=SER=WTVSAM,DISP=OLD
//SYSIN     DD     *
   ABSDUMPT  00010000 00010010        WILL PRINT SPECIFIC EXTENTS
   OR
   ABSDUMPT ALL          WILL PRINT THE ENTIRE CATALOG DATA SPACE
/*
```

B.1.2 HOW TO CHANGE OWNERSHIP BITS AND VOLUME TIMESTAMP:

For all functions described in the next three sections DOS/VS users may use the IKQVDU utility (described on page 179).

The functions described in the next three sections are a violation of VSAM data set security and protection and should only be used in critical situations.

A description of the 'data set security bit' is included in section 2.10.2 on page 45.

A description of the 'VSAM ownership bit' is included in section 2.10.1 on page 44.

A description of the 'VSAM volume timestamp' is included in section A.2 on page 244.

B.1.2.1 TURN OFF THE 'DATA SET SECURITY BIT' (OS/VS)

When the catalog cannot be opened due to I/O errors or damaged self-defining records, the only way to print the catalog without STEPCAT is turning off the 'data set security bit' (in OS/VS systems also called 'OS/VS password protection bit') in the Format 1 DSCB (see page 245 note 10) for the catalog data space.

The following procedure can be used to turn off the 'data set security bit' :

- First execute utility IEHLIST with the LISTVTOC function to get the 'cccchhhhrr' of the FORMAT-1 DSCB of the catalog space with the name 'Z9999994.VSAMDSPC.Txxxxxxx.Tyyyyyyy' (see section A.2.2 on page 245).
- Then turn the 'data set security bit' OFF with the following job:

```
//ZAP          EXEC     PGM=AMASPZAP
//SYSPRINT     DD       SYSOUT=A
//SYSLIB       DD       DSN=FORMAT4.DSCB,UNIT=3330,
//                      VOL=SER=WTVSAM,DISP=OLD,DCB=KEYLEN=44
//SYSIN        DD       *
  CCHHR     CCCCHHHHRR          /*OBTAIN FROM FORMAT-1 DSCB*/
  VER       005D      10C0
  REP       005D      00C0      /*TURN OFF 'DATA SET SECURITY BIT' */
/*
```

Note:

The SYSLIB to be specified is always 'FORMAT4.DSCB' even though the Format-1 DSCB is modified.

B.1.2.2 TURN OFF THE 'VSAM OWNERSHIP BIT' (OS/VS)

If a disk contains VSAM objects which should be deleted and the VSAM catalog which owned the volume is not accessible or not available any more, one way to use the volume for VSAM objects is to turn the VSAM ownership bit off.

In DOS/VS the IKQVDU utility (described on page 179) can be used. In OS/VS the following method or ALTER REMOVEVOLUMES (see example on page 181) can be used.

The 'VSAM ownership bit' in the Format-4 DSCB is shown on page 245 note 4.

```
//ZAP         EXEC     PGM=AMASPZAP
//SYSPRINT    DD       SYSOUT=A
//SYSLIB      DD       DSN=FORMAT4.DSCB,UNIT=3330,
//                     VOL=SER=WTVSAM,DISP=OLD,DCB=KEYLEN=44
//SYSIN       DD       *
  CCHHR     CCCCHHHHRR          /*OBTAIN FROM FORMAT-4 DSCB*/
  VER       0054      80
  REP       0054      00        /*TURN OFF VSAM OWNERSHIP BIT */
/*
```

B.1.2.3 MODIFY THE VSAM VOLUME TIMESTAMP (OS/VS)

As described in section A.5.1 on page 254 a catalog may be out of sync with its volume(s) after either reloading a backlevel catalog or a backlevel volume containing the catalog.

To enable access to the VSAM data sets for recovery purposes the F4 'volume timestamp' (see page 245) must be changed to exactly the same value as the 'volume timestamp' in the catalog's volume record (see page 159).

The catalog timestamp is shown on page 159.

The VTOC volume timestamp is shown on page 245 note 6.

The Format-4 'volume timestamp' may be modified with the following job:

```
//ZAP         EXEC     PGM=AMASPZAP
//SYSPRINT    DD       SYSOUT=A
//SYSLIB      DD       DSN=FORMAT4.DSCB,UNIT=3330,
//                     VOL=SER=WTVSAM,DISP=OLD,DCB=KEYLEN=44
//SYSIN       DD       *
  CCHHR     CCCCHHHHRR                    /* OBTAIN FROM LISTVTOC */
  VER       0057      XXXXXXXXYYYYYYYY    /* OBTAIN FROM LISTVTOC */
  REP       0057      XXXXXXXXYYYYYYYY    /* OBTAIN FROM LISTCAT  */
/*
```

B.2 CATALOG PERFORMANCE CONSIDERATIONS (OS/VS)

As described in detail in the following sections, VSAM catalog management uses multi-string direct I/O to read the catalog.

The default buffer space (3K) allows to build 2 strings.

If a catalog is used by many users/programs additional strings may improve performance dramatically.

The following table shows the relation of buffer space, defined with the BUFSP parameter at DEFINE time and the number of resulting strings.

BUFSP	Strings	Index Buffer	Data Buffer
3 K	2	3	3
4 K	3	4	4
5 K	4	5	5
6 K	5	6	6
7 K	6	7	7
8 K	7	8	8
9 K	7	10	8
10 K	7	12	8

Theoretically there is no limit in specifying a large amount of BUFSP. This is not yet documented in Access Method Services Manuals. The specified buffer space in excess of 8K is added to number of index buffers.

When using shared UCB to access a catalog from multiple CPUs all buffers are refreshed for each access. This may decrease performance when using too many buffers.

B.3 COPY CATALOG - CATALOG (REPRO) (MVS ONLY)

The following terms are used:

COPYCAT = is a terminology used loosely to describe the REPRO function which copies a VSAM catalog into another VSAM catalog (MVS only)
CRA = Catalog Recovery Area
MCAT = VSAM Master Catalog
RCAT = VSAM Recoverable Catalog (user or master catalog)
UCAT = VSAM User Catalog

'COPYCAT' is an MVS exclusive function. It allows an MVS user to copy a catalog directly into another new catalog. The target catalog must be a new empty catalog, but it could have a different name, volser and devicetype than the source catalog. Thus it is a very nice tool to be used by MVS users for catalog device conversion or backup.

'COPYCAT' only supports regular (nonrecoverable) catalogs. If either the source or the target is recoverable, 'COPYCAT' stops. The reason for this is that 'COPYCAT' opens both catalogs as VSAM data sets and uses regular VSAM GET/PUT logic. Thus it bypasses any kind of CRA processing which of course, if 'COPYCAT' with recoverable catalogs were allowed, would cause catalog-CRA out of sync conditions. Hence no support for recoverable catalogs (here called 'RCAT') at all.

In the process of copying, 'COPYCAT' copies the self describing records of the original source catalog also (it is simply doing CI by CI reads and writes.) It is up to the user to delete these entries subsequent to the copy function. The user must also realize that until such a point as the original source Catalog is deleted, we have 2 catalogs owning the volumes and data sets involved. This is a severe integrity exposure that should be immediately remedied by the user deleting the source catalog's cluster entry in the new target catalog. This DELETE run cleans up the F4 DSCB VTOC on the original catalog indicating that the catalog is gone and frees up the original catalog space. It also deletes the source catalog self-describing records and cluster entries in target. The MVS Access Method Services SRL (see page 2) describes this function further under the topic of "Copy - Catalog Procedure" in the chapter "Copying and Printing".

B.4 OS/VS CVOL CATALOG VERSUS VSAM CATALOG

The discussion is designed to give the reader enough pertinent information such that an appropriate decision can be made in deciding between VSAM User Catalogs (UCATs) and OS/VS CVOLs. It will however point out important considerations that can be used by any SCP user to determine the contents of each type of catalog.

The following terms are used:

CVOL = OS/VS System Catalog
HKR = High-Keyrange
LKR = Low-Keyrange
MCAT = VSAM Master Catalog
RCAT = VSAM Recoverable Catalog (user or master catalog)
UCAT = VSAM User Catalog

This discussion does not reference any direct performance figures. Benchmark figures in this type of comparison would be meaningless since performance, as it will be pointed out, is dependent on the contents of each catalog and the type of processing done. Any UCAT oriented discussion naturally applies to the MVS MCAT also since the MCAT is nothing other than a UCAT that is designated by the user as an MCAT.

B.4.1 VSAM USER CATALOG ADVANTAGES (OS/VS)

The following section discusses the more favorable aspects of a VSAM catalog.

- A VSAM catalog's index structure is a balanced tree. Additions or deletions to the catalog do not change the structure, i.e. gaps are not introduced and fragmentation does not occur. So the performance of the catalog does not necessarily degrade with additions or deletions (in some cases the DEFINE performance may decrease significantly if many VSAM data sets are defined on the same volume).

- A VSAM catalog supports both VSAM and nonVSAM entries.

- VSAM catalogs can be shared between virtual SCP systems with total integrity.

- VSAM catalogs are also totally portable between systems. A MCAT can be taken over to another system and used as an MCAT or connected as a UCAT to the existing MCAT. The incompatibilities in terms of portability or sharing possibilities are only relative to Recoverable Catalogs (RCAT) in cases where an SCP does not support RCATs (SVS until April 77).

- VSAM catalogs certainly do not suffer from lack of utility support. In fact if there is a problem it is the other way around where there are many utilities that a user has to know about and learn.

- If we use VSAM catalogs we have the capability of using Recoverable Catalogs (RCAT) which is certainly a useful feature to have. They, unlike other catalogs, give you the capability of having a dynamic backup structure such that at any given point in time your backup is totally up to date and in sync with your catalog. We additionally have a comprehensive set of utilities to access this backup structure for recovery purposes. All of this is available without any major loss in performance.

- JOBCAT/STEPCAT DD cards give a user the capability of directing a request to a particular catalog. This reduces the search path length and is optimum in terms of performance.

- One can specify additional new fields for even nonVSAM data sets in a VSAM catalog. Owner ID and retention period are new fields in addition to the normal volser, devicetype information.

- In terms of protection VSAM catalogs have the added function of password protection. Three levels of passwords, Master, Update and Read levels can be specified for a catalog. The fourth type, CI level password, does not really have any meaning with VSAM catalogs (see also description of passwords on page 52).

B.4.2 VSAM USER CATALOG DISADVANTAGES (OS/VS)

The following is a compilation of unfavorable items relative to VSAM catalogs.

- A lot of users, specially MVS MCAT users would like a relaxation of volume ownership rules. These rules limit the ownership of a volume to only one VSAM catalog. This rule only applies to VSAM data sets. NonVSAM data sets on an owned VSAM pack can of course be cataloged in any other catalog.

- VSAM catalogs generally end up being very large objects. Especially compared with OS/VS CVOLs they are much larger. There is a good deal of space in the Low-Keyrange of VSAM catalogs specially for nonVSAM entries. But then again for VSAM entries there is a great deal more information recorded than what we have been used to.

  The big size of a catalog, apart from obvious disk space considerations, makes the DASD arm seek span larger which is a performance consideration.

- The number of I/Os required to locate anything in a VSAM catalog is more than likely a minimum of 3. As an example to locate a nonVSAM single volume entry, catalog management must read the following:

  1. High level index record

     No I/O. This is kept in storage after the very first access.

  2. An intermediate level index record if the number of index levels are greater than 2

  3. A sequence set record. This however can be in storage in an index buffer

  4. The High-Keyrange record

5. The Low-Keyrange record.

- Catalog management uses VSAM multi-string direct I/O to read the catalog. This type of I/O invalidates the data buffer after the buffer is released, i.e. when the I/O is done, so the likelihood of finding any valid data associated with the string being used is very low. So at least the HKR and LKR records must be read. The sequence set record will probably have to be read since multiple strings do not share sequence set buffers hence the likelihood of a string finding a sequence set record associated with its buffer is minimal.

- VSAM KSDS's currently suffer from a phenomenon,explained below, which in general is referred to as the "Space Reclamation problem." A VSAM catalog being a KSDS suffers from this problem. The problem is, however, limited only to the HKR.

  For every data CI in a KSDS, there is an entry in a sequence set record. This entry contains the compressed key of the record with the highest key in the CI. Keys, when they arrive into the index component, are compressed relative to what key had arrived prior to them and what will come after them. They are front and rear compressed. One can have a 255 byte key that gets compressed down to 1 byte by the time it gets into the sequence set record. Because of the compression they actually express a range of possible keys as opposed to an actual key associated with the particular CI.

  With the current implementation, all the records in a CI can be erased so that it is totally empty, however the sequence set pointer is not removed. As long as the sequence set pointer remains where it is, the only way one can put a record into this empty CI, is via coming in with a record whose key fits within the range described by the sequence set pointer. If this is never the case, the CI will always sit there, taking space and never being used. The space will never be reclaimed. Hence the "Space Reclamation Problem."

  In normal KSDS type of processing, it being considered a randomly retrieved or accessed kind of object, there is no problem of space reclamation. Random access means just that. The likelihood of coming in with a key that fits in the range described with the sequence set record is very good. In certain cases however, we might have this problem. If we are always inserting records that have successively higher key values and never coming in with lower keys we will have this problem.

  VSAM catalogs suffer from this problem when GDGs are cataloged in them and the GDG generation numbers are never reused. Each generation of a GDG has a key higher than the previous key. If we delete a GDG base and never reuse those same generation numbers (unlikely situation') we will have the wasted space in the HKR and only the HKR.

  The problem used to be worse in the old version of VSAM. Generated VSAM names always consisted of 8 bytes of the time of

day clock value. So they were always in ascending sequence. Their subsequent deletions did not do any good because other generated keys always had a higher value. Enhanced VSAM fixed this particular problem by taking the TOD value and breaking it in half, putting one half in the front of the name and the other half on the other end, thus making the name totally random.

If a user currently uses GDGs the way it is explained above, his/her choices are twofold:

(1) Do not put GDGs in VSAM catalogs;
(2) Periodically reorganize the catalog via the Repro Access Method Services command. More on this later (see section A.7 on page 261).

- There is no GDG support in VS1/SVS VSAM catalogs. If a GDG Base or new generation is cataloged in a VSAM catalog during an MVS migration process and the catalog is subsequently moved back to VS1/SVS, the new GDG entries must be manually recataloged in an appropriate CVOL. Additionally there is no ALIAS name support in SVS/VS1 VSAM catalogs.

- ALIAS support is an MVS exclusive support which is mainly used for CVOL or UCAT entries. Data set names cataloged in MVS CVOLs or UCATS are mostly accessed through qualified data set names that have the ALIAS of the CVOL (or UCAT) or the catalog name as a first level qualifier. This support is not needed in SVS/VS1 because ALIAS support for CVOLs has always been available in these systems and of course STEPCAT/JOBCAT DD cards are the only means of accessing VSAM catalogs even if the entries are qualified.

- VSAM catalogs require a good deal of virtual storage for buffers and control blocks. In an SVS environment all of the catalog control blocks are fixed for the duration of the OPEN. In VS1 and MVS (Enhanced VSAM) these control blocks and buffers are not fixed; they are pageable. These required control blocks and buffers get fixed only for the duration of the I/O.

  Generally speaking, for a KSDS, the amount of storage needed for control blocks is equal to 4560 + 1560 (S - 1) where S it the number of strings. A VSAM catalog being a KSDS, therefore requires a minimum of 4560 + 2560, i.e. 7120 bytes of control blocks. This is for the case when bufferspace of 3K is specified which translates into 2 strings. The 3K of buffers is on top of this which makes it approximately 10K of virtual storage. Of this amount only 224 x S (MVS) or 72 x S (VS1) is always fixed. The rest is pageable common storage.

  To summarize, a VSAM catalog requires anywhere from 10K (2 strings) to 20K (7 strings) of virtual storage. This storage comes out of protected common storage areas. CSA in MVS and SQA in VS1. Too many open catalogs will have the effect of extending the users address space or partition size (see also chapter 9.0 starting on page 225, where VSAM working set and storage

sizes are described).

- Without Recoverable VSAM catalogs (RCATs), the act of cataloging an entry in a VSAM catalog was nothing more than making an entry into the catalog. The volume containing the data set did not even have to be around. With RCAT since the catalog entries are duplicated in the CRAs on each owned volume, all of the volumes specified must be mounted. Anytime a CRA is accessed the volume must be mounted. In MVS these volumes are dynamically allocated and mounted. In VS1 we need to specify DD cards so the scheduler can mount and access them.

- Compared to CVOLs, VSAM catalogs suffer from limited access problems.

  For update processing, writing into or updating, VSAM catalogs allow only single access. This says that while a requester is writing, every other requester must wait. For nonupdate type of requests, this restriction does not apply, but we still have a maximum limitation. The limitation is based on the STRING number. Each string specifies a concurrent request. If our string number is 2 we can only have 2 requesters concurrently reading. If the string number is 7 the limit is for 7 requesters and no more.

B.4.3 CVOL ADVANTAGES (OS/VS)

Prior to the availability of the Data Management SU (SU 8), CVOL support in MVS was minimal. SU 8 has restored CVOL support to the level available in SVS, VS1, etc. subject, however, to the single catalog structure of MVS (as opposed to dual catalog structure of SVS/VS1 where we have an OS/VS System Catalog and a VSAM Master Catalog). Some of these differences have been discussed and the rest will be pointed out shortly.

- CVOLs are typically much smaller objects than VSAM catalogs. This in turn means that the seek span of the DASD arm when accessing a CVOL can be much smaller, which in turn could translate into better performance

- CVOLs are migratable between not only virtual SCP's but also MVT and other non-virtual control programs.

- There is no limit for concurrent READ access to CVOLs in one CPU. This is contrasted to VSAM catalogs that have a maximum limit of concurrency of access of 7.

- CVOLs do not suffer from the so called "space reclamation" problem of VSAM catalogs as described earlier (section B.4.2 on page 274). They have total space reusage. Although this saves DASD space for the user, it can pose other problems that will be discussed in section B.4.4 on page 278).

- CVOL entries are stored with the key field on the DASD device. Therefore searches are done on the DASD device by DASD key.

VSAM catalogs have an index structure which is read into storage and searched via the CPU. CVOL searches mean more DASD and channel time, whereas VSAM searches involve more CPU time.

- CVOLs performance is dependent on the size of the catalog, the number of index levels and the number of names at each level. The smaller the size the smaller the DASD arm seek span. The smaller the number of index levels and entries at each level, the lesser the search time and therefore the faster the access.

- It is possible to chain CVOLs such that a data set can be found via searching multiple CVOLs chained together. Another way to point this out is to indicate that a CVOL can point to CVOL and so on. VSAM catalogs can not point to other VSAM catalogs (no chaining) with the exception of the Master catalog pointing to user catalogs or to a CVOL (MVS only). User catalogs can not point to user catalogs.

B.4.4 CVOL DISADVANTAGES (OS/VS)

- CVOLs do not support VSAM data sets. VSAM catalogs support both VSAM and nonVSAM.

- Although, as pointed out earlier, CVOLs have total space reuse, this can be done at the expense of some performance. It is entirely possible that over a period of time with additions and/or deletions, CVOLs can become fragmented to the point that the DASD arm can "ping-pong" back and forth between cylinders far apart. This is a characteristic of large very active old CVOLs. The obvious solution to the problem is eventual reorganization, which leads us into the next topic.

- Certainly, compared to VSAM catalogs, CVOLs suffer from the lack of comprehensive utility support. The only utility available for any CVOL activity in terms of backup, recovery, reorganization etc. is IEHMOVE which is not exactly near and dear to too many hearts. Most shops have their own user-written version of a utility to backup and reorganize CVOLs.

- System access to CVOL entries, in an MVS environment, is through qualified data set names. By system access this means scheduler allocation activity via JCL. Generally speaking access in MVS is always through the MCAT via qualified data set names and "SYSCTLG" pointers in the MCAT.

  The only exception is where a user through IEHPROGM or some user code, writes a CAMLST macro specifying a CVOL parameter. Access in this case is directly to the particular CVOL, but even in this case the SYSCTLG entry in the MCAT must exist, since the scheduler still accesses the MCAT to find out the CVOL devicetype. This essentially indicates that all data set names in a CVOL must be qualified and the first level qualifier must be an ALIAS name of the CVOL and reflected in the MCAT.

C.0 APPENDIX C

C.1 DEVICE TYPE TRANSLATE TABLE

The following device type translate table has been extracted from Access Method Services manual and is valid for all systems:

LISTCAT Code	Device Type
3000 8001	9 TRK TAPE
3040 200A	3340 (35M/70M)
3050 2006	2305-1
3050 2007	2305-2
3050 2009	3330-1,2
3050 200B	3350
3050 200D	3330-11
3058 2009	3330V
3080 8001	7 TRK TAPE
30C0 2008	2314/2319

C.2 LIST OF ABBREVIATIONS

ACB = Access Method Control Block (VSAM DCB equivalent)
AIX = Alternate Index
AMS = Access Method Services (abbreviation not used any more)
CA = Control Area
CI = Control Interval
CIDF = Control Interval Definition Field
CNV = Control Interval (used for control interval access)
CRA = Catalog Recovery Area
DASD = Direct Access Storage Device
DCB = Data Control Block (OS/VS control block to describe a nonVSAM data set)
DSCB = Data Set Control Block (in the VTOC)
ESDS = Entry Sequenced Data Set
HKR = High-Keyrange
ICI = Improved Control Interval access
IS = Index Set (part of the index)
KSDS = Key Sequenced Data Set
LKR = Low-Keyrange
LR = Logical Record
MCAT = VSAM Master Catalog
RBA = Relative Byte Address
RCAT = VSAM Recoverable Catalog (user or master catalog)
RDF = Record Definition Field
RPL = Request Parameter List (VSAM control block)
RRDS = Relative Record Data Set
SS = Sequence Set (part of the index)
UCAT = VSAM User Catalog
VTOC = Volume Table Of Contents (disk directory)

INDEX:

VSAM PRIMER AND REFERENCE G320-5774-01

Your comments Please . . .

Reply requested

Yes ☐

No ☐

Name _____

Job Title _____

Address _____

Thank you for your cooperation

YOUR COMMENTS PLEASE.....

Your comments on the other side of this form will help us improve future editions of this publication. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material.

fold fold

IBM World Trade Corporation
World Trade Systems Center
Department 471
P.O. Box 50020
San Jose, California 95150
U.S.A.

fold fold

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
360 Hamilton Avenue, White Plains, New York 10601
(international)

Printed in U.S.A.