GC26-4066-0
File No. S370-30

Program Product

MVS/370 VSAM Users Guide

Data Facility Product 5665-295

Release 1.1

IBM

First Edition (April 1983)

This edition, as amended by technical newsletter GN26-8093, applies to Release 1.1 of MVS/370 Data Facility Product, Program Product 5665-295, and to any subsequent releases until otherwise indicated in new editions or technical newsletters.

The changes for this edition are summarized under "Summary of Amendments" following the preface. Specific changes are indicated by a vertical bar to the left of the change. These bars will be deleted at any subsequent republication of the page affected. Editorial changes that have no technical significance are not noted.

Changes are periodically made to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest IBM System/370 and 4300 Processors Bibliography, GC20-0001, for the editions that are applicable and current.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM program product in this publication is not intended to state or imply that only IBM's program product may be used. Any functionally equivalent program may be used instead.

Publications are not stocked at the address given below; requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, P.O. Box 50020, Programming Publishing, San Jose, California, U.S.A. 95150. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1983

GC26-4066-0
File No. S370-30

Program Product

MVS/370 VSAM Users Guide

Data Facility Product 5665-295

Release 1.0

IBM

First Edition (April 1983)

This edition applies to Release 1.0 of MVS/370 Data Facility Product, Program Product 5665-295, and to any subsequent releases until otherwise indicated in new editions or technical newsletters.

Changes are periodically made to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest IBM System/370 and 4300 Processors Bibliography, GC20-0001, for the editions that are applicable and current.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM program product in this publication is not intended to state or imply that only IBM's program product may be used. Any functionally equivalent program may be used instead.

Publications are not stocked at the address given below; requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, P.O. Box 50020, Programming Publishing, San Jose, California, U.S.A. 95150. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1983

PREFACE

This publication describes the use of virtual storage access method (VSAM). It is intended for programmers who use VSAM macro instructions, access method services commands, or JCL to process VSAM data sets. The publication has the following major divisions:

- "VSAM Data Sets and Records" describes the types of VSAM data sets and the data set's structure.
- "Data Set Procedures" describes the procedures used to define, alter, delete, copy, print, and recover VSAM and non-VSAM data sets.
- "VSAM Performance Considerations" describes many of the concepts and parameters that influence VSAM performance.
- "Security" discusses the types of security arrangements that are available with VSAM.
- "Sharing a VSAM Data Set" discusses how VSAM data sets may be shared by different operating systems, by different jobs in a single operating system, and by different subtasks in an address space.
- "Options for Advanced Applications" discusses applications that might be effective in the development of high performance programs and system control programs and how to prestage data for use in certain Mass Storage System applications.
- "Job Control Language" describes JCL for VSAM, including DD statements for catalogs and the DD AMP parameter.
- "User-Written Exit Routines" describes how to write exit routines to be used with VSAM.
- "Invoking Access Method Services from a Problem Program" gives an example of how to use the access method services commands in a user program.
- "Using ISAM Programming with VSAM" describes how data sets can be converted to VSAM's format and processed using an ISAM processing program.
- "Glossary" defines VSAM terms.
- "Index" is a subject index to this publication.

CONVENTIONS USED IN THE PUBLICATION

The conventions used in this publication for writing the macro and JCL statements indicate whether an operand is optional, how to specify the value for an operand, and how to punctuate a macro or statement. The conventions are:

- Expressions enclosed in brackets, [ ], are optional.
- Items separated by an OR sign, |, and enclosed in braces, { }, are alternatives, one of which must be specified.
- An underlined bold item, <u>**ITEM**</u>, is the default when you do not specify an operand.
- Ellipses, ..., indicate that you may repeat the preceding item.

- Capitalized **BOLD** expressions, parentheses, commas, and equal signs must be entered as shown. Unless otherwise noted, parentheses are not required if you specify only one item.

- Underlined lowercase expressions are variables; you may specify a variable.

VSAM AND ACCESS METHOD SERVICES PUBLICATIONS

The VSAM and access method services publications are:

- MVS/370 VSAM Reference, GC26-4074, provides information about the VSAM macro instructions.

- MVS/370 Access Method Services Reference for the Integrated Catalog Facility, GC26-4051, contains a description of the commands that are used to copy, print, and load data sets. The relationships among components, the structure of components, the use of the catalog, and access method services commands to define and delete data sets, list catalog entries, and move data sets from one operating system to another are described.

- MVS/370 Access Method Services Reference for VSAM Catalogs, GC26-4059, contains a description of the commands that are used to copy, print, and load data sets. The relationships among components, the structure of components, the use of the catalog, and access method services commands to define and delete data sets, list catalog entries, and move data sets from one operating system to another are described.

- MVS/370 Data Facilities Planning Guide, GC26-4052, describes planning considerations for VSAM.

- MVS/370 Catalog Users Guide, GC26-4053, describes the use of ICF catalogs, with an appendix that covers VSAM catalogs and OS CVOLs.

- MVS/370 Catalog Diagnosis Reference, SY26-3915, describes the process by which a program component and type of failure can be determined.

- MVS/370 Access Method Services Logic, LY26-3912, describes the internal logic of access method services.

- MVS/370 VSAM Logic, LY26-3928, describes the internal logic of VSAM.

RELATED PUBLICATIONS

The reader also needs to be familiar with some of the information presented in the following publications:

- MVS/370 Data Management Services, GC26-4058, presents basic concepts such as access methods, direct access storage, and the distinction between data set organization and data set processing.

- OS/VS2 MVS JCL, GC28-0692, describes the JCL parameters referred to in this publication and describes dynamic allocation.

- OS/VS2 System Programming Library: Job Management, GC28-0627, describes dynamic allocation.

- OS/VS2 TSO Command Language Reference, GC28-0646, and OS/VS2 TSO Terminal User's Guide, GC28-0645, describe the time sharing option.

- OS/VS System Modification Program (SMP) System Programmers Guide, GC28-0673

- MVS/370 System Generation Reference, GC26-4063
- MVS/370 Checkpoint/Restart, GC26-4054
- OS/VS Message Library: VS2 System Messages, GC38-1002
- MVS/370 Utilities, GC26-4065
- OS/VS2 Data Areas, SYB8-0606
- OS/VS2 System Programming Library: Debugging Handbook, GC28-1047, GC28-1048, and GC28-1049
- OS/VS2 MVS Resource Access Control Facility (RACF): General Information Manual, GC28-0722

SUMMARY OF AMMENDMENTS

RELEASE 1.1, OCTOBER 1983

NEW PROGRAMMING SUPPORT

Description of Resource Access Control Facility (RACF) Release 1.5 Data Management Support.

CONTENTS

FIGURES

INTRODUCTION

VSAM supports both direct and sequential processing to give you direct access to records in any order and sequential access to records that follow one another. You can identify a record for retrieval by its key (a unique value in a predefined field in the record), by its displacement from the beginning of the data set, or by its relative record number. These alternative types of access and access options enable you to design a program to suit your requirements for processing data.

VSAM DATA SETS AND RECORDS

Three data set organizations are used for VSAM and include key-sequenced data set, entry-sequenced data sets, and relative record data sets.

KEY-SEQUENCED DATA SETS

A key-sequenced data set (KSDS) always includes an index, which is a mechanism for keeping track of records. A record is identified for retrieval by its key (a unique value in a predefined field in the record).

Key-sequenced data format consists of an index, logical records, and data sets. A key-sequenced data set contains one or more logical records that may have a fixed length or may vary in length. A logical record is a unit of information grouped together based on its contents or function (for example, a personnel file), not its physical attributes (for example, the size of the data set).

Each logical record contains a unique, embedded key. The index contains key information and a pointer to logical records in a data set. VSAM keeps track of records by the key field so you only need to refer to a record by its key field and not in some location-dependent manner. The records in a key-sequenced data set may be accessed using keyed access, addressed access, and control interval access. The relative byte address of records can change in a key-sequenced data set when records are added, deleted, shortened, or lengthened.

ENTRY-SEQUENCED DATA SET

In an entry-sequenced data set (ESDS), a record is identified for retrieval by its displacement from the beginning of the data set.

Entry-sequenced data format consists of logical records and data sets. An entry-sequenced data set contains one or more logical records that are fixed or variable in length. Entry-sequenced data sets do not have an index as key-sequenced data sets do. The logical records are sequenced in the data set by the time of their arrival. Records may be added at the end and may be updated, but not extended or erased. The relative byte address (RBA) is used to locate the record. Once a record is added to an entry-sequenced data set, it stays there and keeps its original relative byte address.

An entry-sequenced data set is appropriate for applications that require no special ordering of data by the contents of a record. To retrieve records randomly from an entry-sequenced data set, you must keep track of the record's relative byte address and associate relative byte addresses with the contents of records.

RELATIVE RECORD DATA SET

In a relative record data set (RRDS), a record is identified for retrieval by its relative record number.

Relative record data format consists of one logical record in a fixed-length slot. Each slot has a unique relative record number, with the slots sequenced by the ascending relative record numbers from 1 to n.

A logical record is placed in the specified slot by a user-supplied relative record number. If a number is not specified, VSAM supplies a number of 1 for the initial record load or a number that is 1 greater than the last available number for subsequent inserts. A record may be inserted into any empty slot. Records can be retrieved by a relative record number or in the order of their physical sequence.

You can use a relative record data set in much the same way as a direct access method data set in which the data records are not ordered by their contents or their entry sequence.

CONTROL INTERVALS

Logical records for VSAM data sets are structured and stored differently than logical records for non-VSAM data sets.

In VSAM, disk space is divided on a direct access storage device (for example, an IBM 3380 etc.) into segments of information called control intervals. A control interval is a fixed-length area of auxiliary storage that contains logical records. A control interval is a contiguous area that may vary in size from one data set to another. The size may be specified by a user or, as a default, VSAM will establish the size. In order to use a data set (for example, to update or delete a record), the control interval that contains the logical record is moved from disk storage to virtual storage.

A control interval may contain two types of information fields. One type of field is the control interval definition field (CIDF) that exists for each control interval and contains the information for that control interval. The other type may be one or more record definition fields (RDF) that indicate, for example, the length of records, or how many adjacent records have the same length.

All control intervals must contain a control interval definition field, but may also contain logical records and unused space, or logical records only, or unused space only.

CONTROL AREAS

Control intervals are grouped into fixed-length contiguous areas called control areas. VSAM uses the control areas to receive or hold the control intervals of a particular data set in an auxiliary storage space.

VSAM determines the number of control areas based on the number of control intervals. When a control area is full and a data set needs to be extended, VSAM extends the data set and may create a new control area.

SPANNED RECORDS

Sometimes a record is larger than the control interval. These records do not need to be broken apart or reformatted, because VSAM allows you to specify a spanned parameter when defining a data set. This parameter allows a record to extend across or span the control interval. These spanned records apply only to key-sequenced or entry-sequenced data sets.

The spanned record starts at the beginning of one control interval and fills that control interval, then spans to the next control interval. A record may span several control intervals; however, a record cannot span a control area.

If the spanned parameter was specified when the data set was defined, VSAM determines when to span the control interval.

CLUSTERS

When a data set is created, it is defined as a cluster. A cluster can be a key-sequenced data set, which consists of a data component and an index component, or it can be an entry-sequenced or relative record data set, which consists of only a data component.

ALTERNATE INDEXES

An alternate index provides a way to gain access to a related data set so you do not need to keep multiple copies of the same information organized in different ways for different applications. For example, a data set containing payroll information indexed by employee number can also be indexed by other fields such as employee name or department number.

An alternate index accesses the data set in the same way the prime index of a key-sequenced data set does. The data set over which the alternate index is built is the base cluster. The base cluster can be a key-sequenced or an entry-sequenced data set, but not a relative record or reusable data set.

USING VSAM AS AN ACCESS METHOD

VSAM uses access method services or VSAM macro instructions to process data sets and catalogs.

Access method services is a program to use with VSAM to create and maintain data sets and catalogs that includes a set of commands along with various parameters. Access method services commands are defined in Access Method Services Reference.

Access method services commands allow you to:

- Define, alter, and delete data sets
- List catalog entries
- Copy and print data sets
- Construct alternate indexes for VSAM data sets
- Diagnose data structures to prevent the use of invalid data
- Move data sets from one operating system to another or from one device type to another
- Aid in recovery from damage to data sets or catalogs

The macro instructions provided by VSAM 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 parameter lists. The request macros are used to specify the processing action; for example, read or write, to be taken on data and index records. The macro instructions are defined in VSAM Reference.

The macro instructions are used to:

- Build control blocks
- Modify, display, and test the contents of control blocks
- Store, retrieve, and erase records, to position VSAM in a data set, to suspend processing, and to terminate requests
- Develop high performance programs and system control programs
- Prestage data for use in certain MSS applications to reduce the number of cylinder faults incurred during processing
- Connect or disconnect data sets

VSAM OPTIONS

Information is provided about advanced applications of VSAM. The topics include:

- Gaining access to control intervals
- I/O buffering
- Constructing parameter lists for the macros that generate, modify, and examine control blocks at execution
- Processing the index as data
- Sharing resources

CHAPTER 1. VSAM DATA SETS AND RECORDS

VSAM has key-sequenced, entry-sequenced, and relative record data sets. The primary difference among the three is the way the data sets are structured.

Records are loaded into a key-sequenced data set in the order defined by the collating sequence of the contents of the key field in each of the records. Each record has a unique value in the key field, such as employee number or invoice number. VSAM uses an index to insert a new record into the data set in key sequence.

Records are loaded into an entry-sequenced data set regardless of the contents of the records. Their sequence is determined by the order in which they are physically received in the data set or their entry sequence. New records are stored at the end of the data set.

Records are loaded into a relative record data set in relative record number sequence. The data set is a string of fixed-length slots, each identified by a relative record number. When a record is inserted, you can assign the relative record number or allow VSAM to assign the next available number in sequence. No index is used.

When you create a data set, you define it, together with its index, if any, in a cluster. A cluster may be a key-sequenced data set, which consists of a data component and an index component, or an entry-sequenced or relative record data set, which consists of only a data component. Further information on data set entities may be found in "Other VSAM Data Set Entities" on page 11.

KEY-SEQUENCED DATA SETS

A key-sequenced data set is always defined with an index that relates key values to the relative locations of the data records in a data set. (This index is the prime index, in contrast to alternate indexes, which are discussed later.)

A key in the index is taken from a record's key field, whose size and position are the same for every record in the data set, and whose value cannot be altered. VSAM uses an index to locate a record for retrieval and to locate the collating position for insertion.

An index has one or more levels, each of which is a set of records that contains entries giving the location of the records in the next lower level. The index records in the lowest level are the sequence set; they give the location of control intervals containing the data records. The records in all the higher levels are the index set; they give the location of index records. The highest level always has only a single record. The index of a data set with a few control intervals for a single sequence-set record has only one level: the sequence set itself.

Figure 1 on page 6 illustrates the levels of a prime index and shows the relationship between a sequence-set index record and a control area. The figure shows that the highest-level index record (A) controls the entire next level (records B through Z); each sequence-set index record controls a control area.

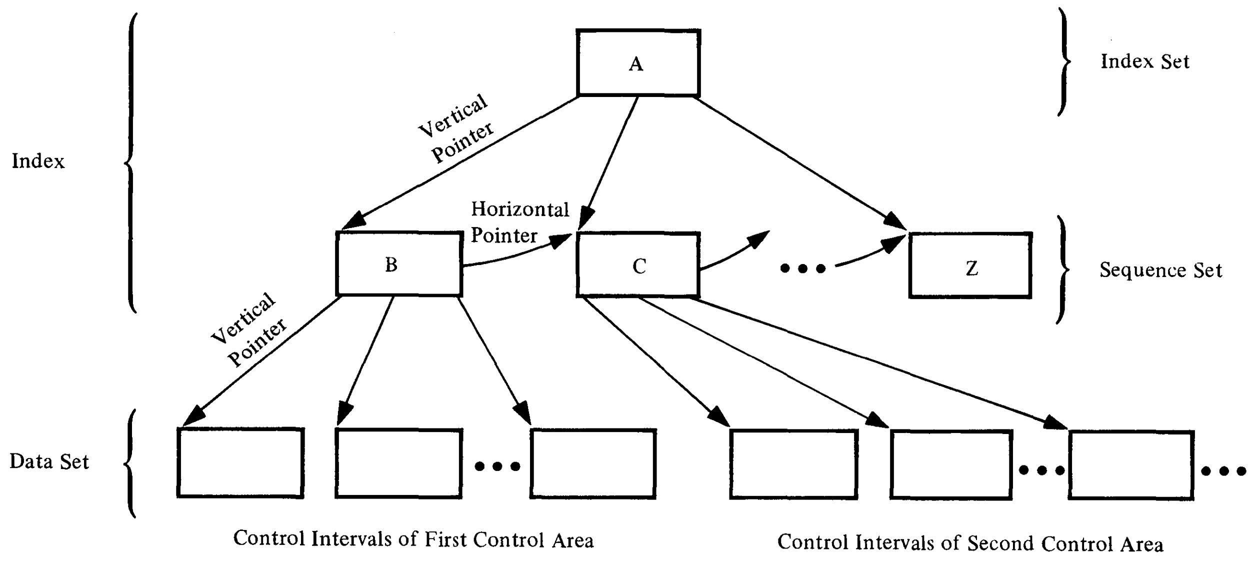


Figure 1. Relationship among the Levels of a Prime Index and a Data Set

An entry in an index-set record consists of the highest key that an index record in the next lower level contains, and a pointer to the beginning of that index record. An entry in a sequence-set record consists of the highest key in a control interval of the data component, paired with a pointer to the beginning of that control interval. Not all data records have sequence-set entries; there is only one entry for each control interval in the data set.

For direct access by key, VSAM follows vertical pointers from the highest level down to the sequence set to find a vertical pointer to data; for sequential access by key, VSAM refers only to the sequence set. It uses a horizontal pointer in a sequence-set record to get from that sequence-set record to the next record in collating sequence to find vertical pointers to data. Figure 1 shows both vertical pointers and horizontal pointers.

VSAM uses a method of key compression to eliminate from the front and the back of a key those characters that aren't necessary to distinguish it from the adjacent keys. Compression helps achieve a smaller index by reducing the size of keys in index entries. Key compression increases the capacity of an index entry and allows access to many more records for a given index record. The number of entries in a sequence set record is limited by the number of control intervals in a control area and the compression characteristics of the entry keys.

When you define a key-sequenced data set, you can specify how free space is to be distributed in one of two ways:

- Leave some space at the end of all the used control intervals or
- Leave some control intervals completely empty

The amount of free space in a used control interval and the number of free control intervals in a control area are independent of each other. You may specify the amount of free space. The selection of free space values depends on whether your program does direct processing, sequential processing, or both. Figure 2 shows how free space might be set aside in each control area of a data set. In addition to entries for used control intervals, the sequence-set record for a control area keeps a record of free control intervals.

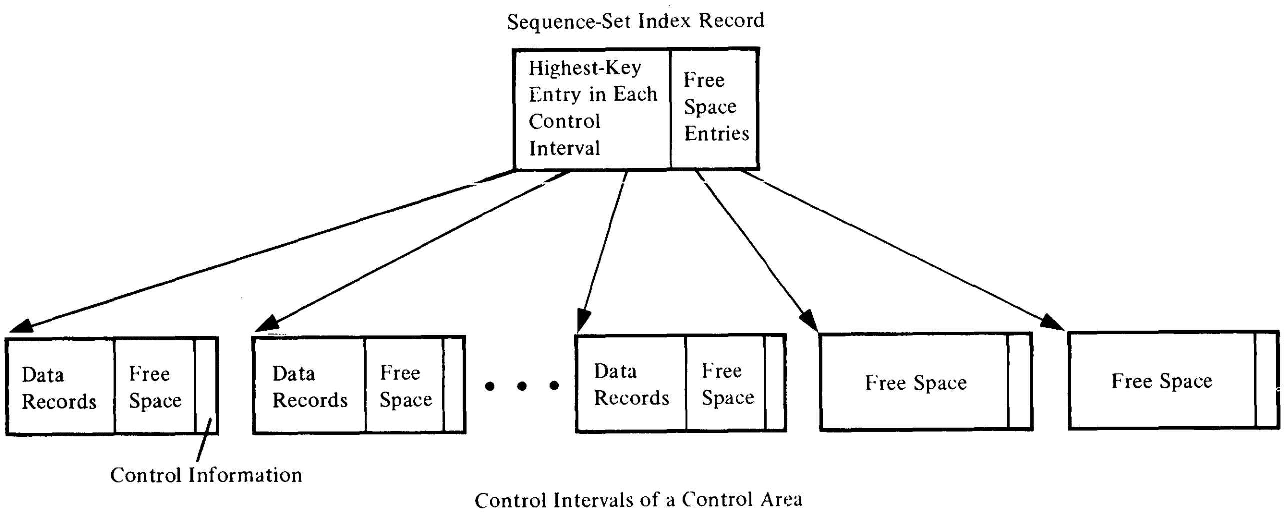


Figure 2. Distribution of Free Space in a Key-Sequenced Data Set

Besides the space that you distribute when you create a key-sequenced data set, space that becomes available within a control interval when a record is shortened or deleted from the data set is automatically reclaimed by VSAM and can be used when a record is lengthened in place or directly inserted into the control interval.

Reclaiming space and using distributed free space may cause RBAs of some records to change. As Figure 2 illustrates, free space within a used control interval is between the data in the front and the control information in the back. If a record is deleted or shortened, any succeeding records in the control interval are moved to the left and their RBAs are changed so the space vacated can be combined with the free space already in the control interval. Any insertion or lengthening causes any succeeding records in the control interval to be moved to the right into free space and their RBAs to be changed.

If the record to be inserted will not fit in the control interval, VSAM will move data and update the index to allow the record to be inserted. This process is called control interval split and control area split.

For additional information about processing key-sequenced data sets, see "Chapter 2. Data Set Procedures" on page 21.

ENTRY-SEQUENCED DATA SETS

A prime index is not associated with an entry-sequenced data set. When a record is loaded or added, VSAM indicates its relative byte address (RBA). You must keep track of the RBAs of the records if you want to gain access to them by direct processing. One way to keep track is to build an index.

Sequential access with an entry-sequenced data set is similar to that of QSAM (queued sequential access method).

For additional information about processing entry-sequenced data sets, see "Chapter 2. Data Set Procedures" on page 21.

RELATIVE RECORD DATA SETS

A relative record data set has no index. It has a string of fixed-length slots that have a relative record number from 1 to n, the maximum number of records that can be stored in the data set. Each record occupies a slot and is stored and retrieved by the relative record number of the slot. Relative record number 9 in Figure 3, for example, occupies the ninth slot, even though all the slots between 1 and 8 are not filled.

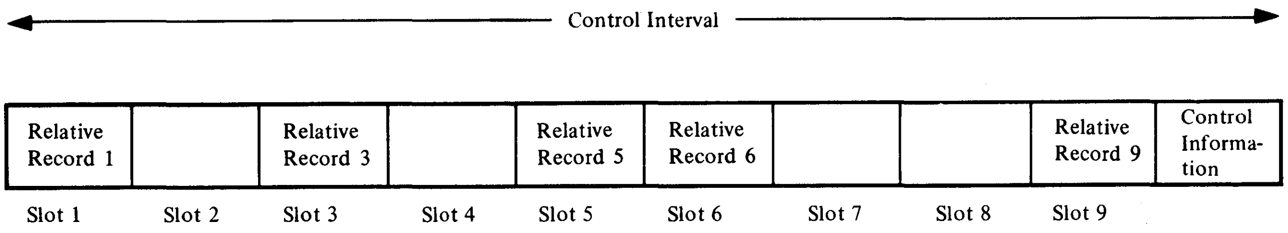


Figure 3. The First Control Interval within a Relative Record Data Set

Records in a relative record data set are grouped together in control intervals. Each control interval contains the same number of slots; the size is the record length specified when defining the data set. The number of slots in a control interval is determined by the control interval size and the record length.

Because the slot can contain data or be empty, a data record can be inserted, deleted, or moved without affecting the position of other data records in the relative record data set. Records can be retrieved sequentially or directly, based on its relative byte address (RBA).

Additional information about processing relative record data sets appears under "Processing VSAM Data Sets" on page 15.

CONTROL INTERVALS

VSAM stores the records of each type of data set in a a control interval. A control interval is a continuous area of auxiliary storage that VSAM uses to store data records and the control information that describes the records. The control interval is the unit of information VSAM transfers between virtual and auxiliary storage. The size may vary from one data set to another, but, for a given data set, the size of each control interval is fixed, either by VSAM or by you, within limits acceptable to VSAM. VSAM chooses the size based on the type of direct access storage device used to store the data set, the size of your data records, and the smallest amount of virtual storage space your processing program provides for VSAM's I/O buffers.

The information recorded on a track is divided into physical records that are limited by the capacity of a track. The physical-record sizes that VSAM uses begin at 512 bytes and increases by powers of 2 up to 4096 bytes: 512, 1024, 2048,

4096. Control interval size is limited by the requirements that it be a whole number of physical records (1, 2, 3, ..., up to 64, or a maximum size of 32 768 bytes). If it is greater than 8192 bytes, it must be a multiple of 2048. A data set whose control intervals correspond with the tracks of one device might have more or less than one control interval per track if it were stored on a different device. Figure 4 illustrates the independence of control intervals from physical records.

When you define a data set and specify track allocation or specify record allocation requiring less than one cylinder, VSAM uses track allocation.

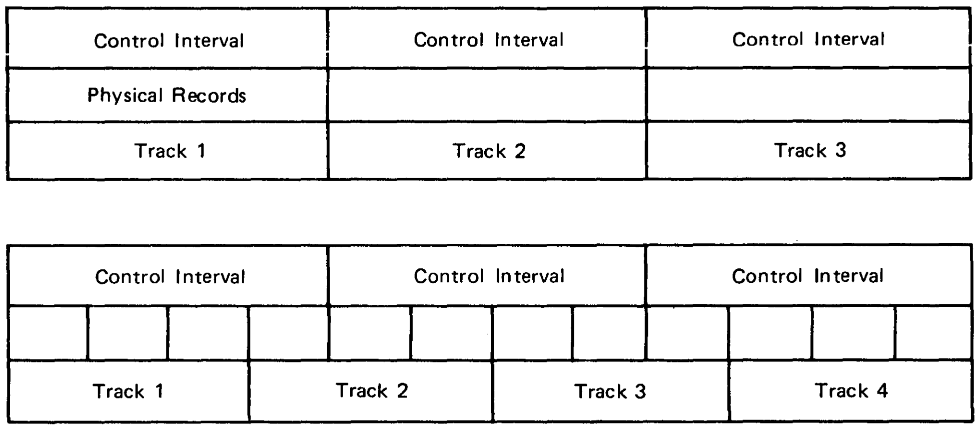


Figure 4. Control Intervals Are Independent of Physical Record Size

A volume may contain areas for VSAM's use and areas for use by other access methods or the operating system. A storage area defined in the volume table of contents for VSAM's exclusive use is called a data space. Each data space on a volume has one or more DSCBs and a unique name.

A data set is stored in a data space or data spaces on one or more volumes on direct access devices of the same type. When you define a data set, you can allocate enough space so some is left at the end of the data set for additions. For new data sets, the amount requested must be available, or DEFINE will terminate. Otherwise, when additional space is needed, VSAM automatically extends the data set by the amount of space indicated in the definition of the data set in the catalog. It can be extended beyond its original size up to 123 extents, or to a maximum size of 2^{32} (approximately 4 290 000 000) bytes.

A data set is made up of control intervals. A group of control intervals makes up a control area. A control area is the unit of a data set that VSAM preformats for data integrity as records are added to the data set. In a key-sequenced data set, control areas are also used for distributing free space throughout the data set as a percentage of control intervals in a control area and for placing portions of the index next to the data set.

The number of control intervals in a control area is fixed by VSAM, with a minimum of two. If 50 were the number chosen, for example, the first 50 control intervals would be the first control area; the next 50 would be the second control area, and so on. Whenever the space for a data set is extended, it is extended by a whole number of control areas. For a key-sequenced data set, the size of a control area is determined by the space allocation request, user-specified or default index and data control interval size, and available buffer space.

RECORDS STORED IN A CONTROL INTERVAL

The records of a key-sequenced or entry-sequenced data set may be fixed or variable in length; the records of a relative record data set are always fixed in length. VSAM treats them all in the same way. It puts control information at the end of a control interval to describe the data records stored in the control interval. The combination of a data record and its control information, though they are not physically adjacent, is called a stored record. When adjacent records are the same length, they share control information. Figure 5 shows how data records and control information are stored in a control interval. The data records are stored at the beginning of a control interval, and control information at the end. For further information on control interval, see "Chapter 6. Options for Advanced Applications" on page 81.

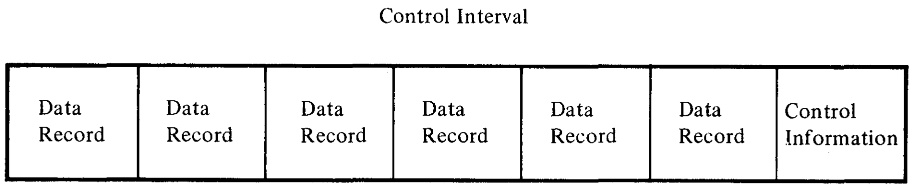


Figure 5. Placement of Data Records and Control Information in a Control Interval

When you define a data set, you should specify enough buffer space so the control intervals in the data set are large enough for the largest stored record.

Key-sequenced and entry-sequenced data set records whose lengths exceed control interval size may cross, or span, one or more control interval boundaries. Such records are called spanned records. A spanned record always begins on a control interval boundary and fills one or more control intervals within a single control area. As shown in Figure 6 on page 11, the control interval that contains the last segment of a spanned record can contain unused space. This free space can be used only to extend the spanned record; it cannot contain all or part of any other record. You must specify your intent to use spanned records when you define the data set.

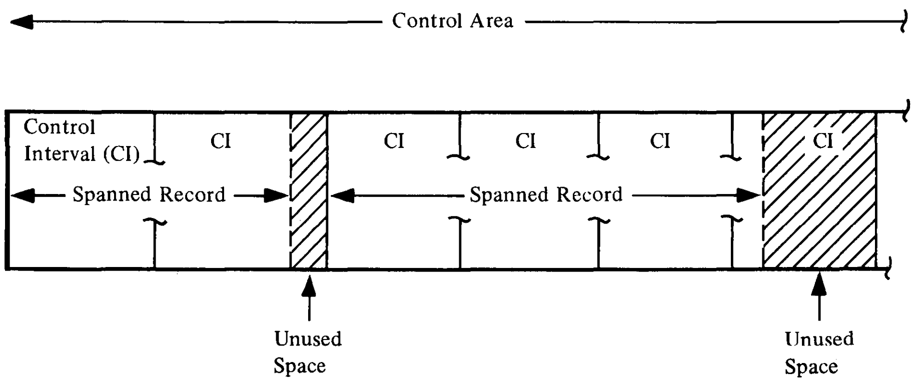


Figure 6. Control Intervals That Contain Spanned Records

A data record is addressed not by its location in terms of the physical attributes of the storage device (such as the number of tracks per cylinder), but by its displacement, in bytes, from the beginning of the data set, called its relative byte address (RBA). The RBA does not depend on how many extents belong to the data set or whether they are in different data spaces or on different volumes. For relative byte addressing, VSAM considers the control intervals in the data set to be contiguous, as though the data set were stored in virtual storage beginning at address 0. For example, the first record in a data set has RBA 0. The second record has an RBA equal to the length of the first record, and so on. The bytes required for intervening control information and free space are included in the RBA value.

OTHER VSAM DATA SET ENTITIES

ALTERNATE INDEXES

An alternate index provides a way to access a related (base) data set, so you do not need multiple copies of the same information for different applications.

You use access method services to define and build one or more alternate indexes over a key-sequenced or an entry-sequenced data set. See "Creating an Alternate Index (DEFINE ALTERNATEINDEX)" on page 31.

CLUSTERS AND ALTERNATE INDEXES

In terms of access, an alternate index performs the same function as the prime index of a key-sequenced data set. The data set over which the alternate index is built is the base cluster. It can be a key-sequenced or an entry-sequenced data set, but not a relative record or a reusable data set.

In structure, the alternate index is similar to a key-sequenced cluster. It consists of an index component and a data component. The index component is identical in structure, format, and function to the prime index of a key-sequenced data set. Each entry in the sequence set of an alternate index index component points to a control interval in the alternate index data component. The format of the alternate index data component is identical to the format of the data portion of a key-sequenced data set. The records in the data component contain an alternate key and one or more pointers to data in the base cluster.

When building an alternate index, the alternate key can be any field in the base data set's records having a fixed length and a fixed position in each record. The alternate key must be in the first segment of a spanned record. For each alternate key, the data component of the alternate index contains a unique record. This record consists of the alternate key itself, followed by a pointer that is the prime key or RBA of the base data record that contains the alternate key. If more than one base data record contains the same alternate key, the alternate index record contains a pointer to each base data record. These duplicate or nonunique keys are discussed in "Alternate Keys" on page 13.

ALTERNATE INDEX PATHS

A path logically relates a cluster and each of its alternate indexes. It provides a way to gain access to the base data through a specific alternate index. You define a path through access method services. You must name it and may give it a password. The path name refers to the base cluster/alternate index pair. When you refer to a path (by way of the OPEN macro, for example), both the cluster and the alternate index are affected (opened). Figure 7 shows how two paths can relate two alternate indexes to a single base cluster.

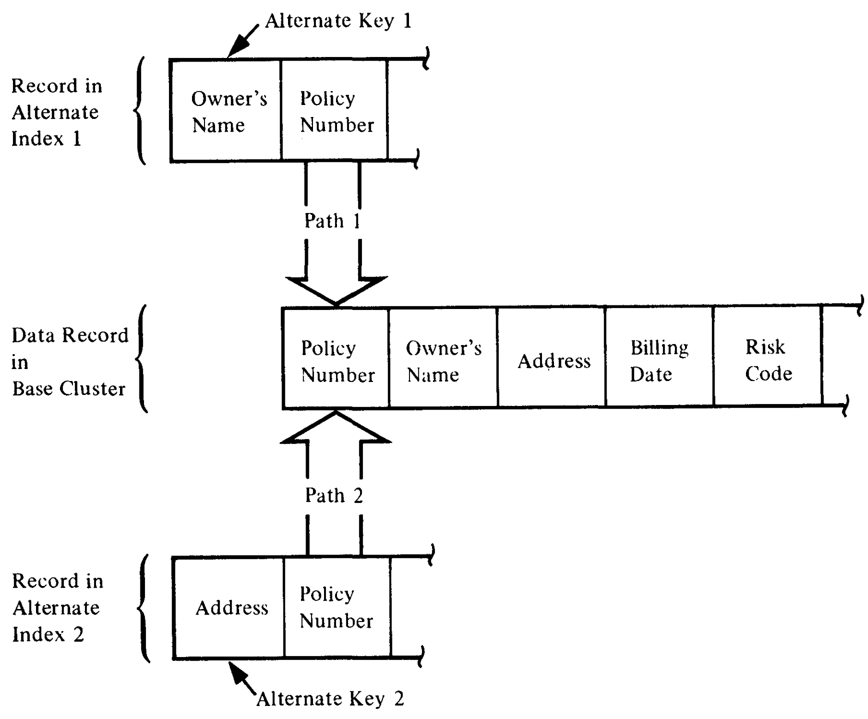


Figure 7. Two Data Sets Over a Single Key-Sequenced Data Set

ALTERNATE INDEX RECORDS

Each record in the data component of an alternate index is of variable length and contains system header information, the alternate key, and at least one pointer to a base data record.

System Header Information

System header information is fixed length and indicates:

- Whether the alternate index record contains prime keys or RBA pointers and unique or nonunique keys
- The length of each pointer
- The length of the alternate key
- The number of pointers

Alternate Keys

Unless the base data records span control intervals, any field in the base data records that has a fixed length and a fixed position within the record can be an alternate key. The alternate key must be in the first control interval of a spanned record. When an alternate index is created, the alternate keys are extracted from the base data records and placed in collating sequence. If you build several alternate indexes over a base cluster, the alternate key fields of the different alternate indexes may overlap each other in the base data records. They can also overlap the prime key.

Keys in the index component of an alternate index or of a key-sequenced base cluster are compressed. Keys in the data component of an alternate index are not compressed, the entire key is represented in the alternate index data record.

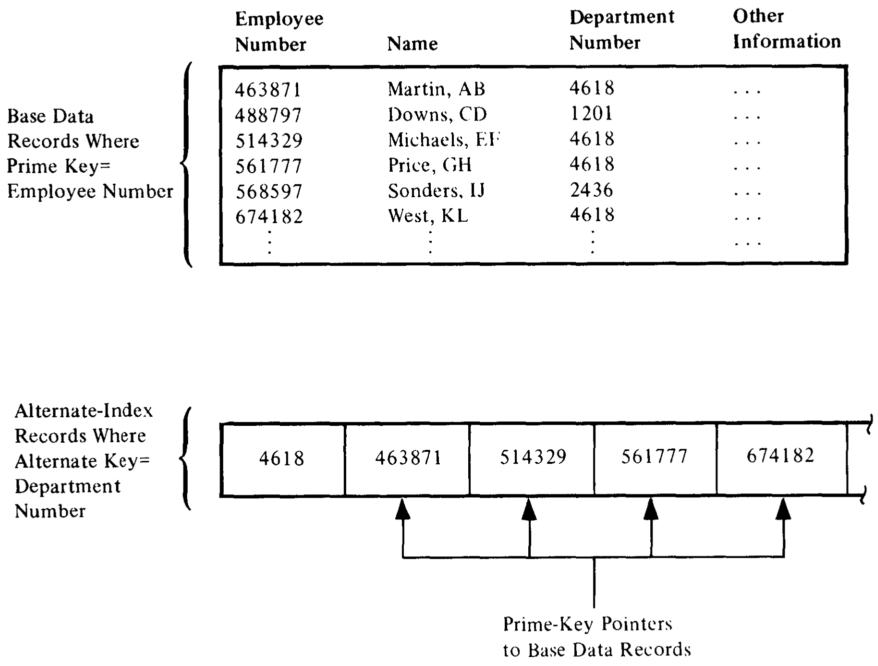
An alternate key may refer to more than one record in the base cluster. For example, if an alternate index is established by department number over a payroll data set organized by employee number, there will be several employees with the same department number, as shown in Figure 8. Several prime-key pointers (employee numbers) are in the alternate index record: one for each occurrence of the alternate key (department number) in the base data set. When multiple pointers are associated with a given alternate key value, the alternate key is said to be nonunique; if only one pointer is associated with the alternate key, it is unique.

Alternate Index Pointers

An alternate index uses prime keys if the base cluster is a key-sequenced data set and RBAs if the base cluster is an entry-sequenced data set.

For a nonunique key, such as department number in Figure 8, multiple pointers are associated with it. The pointers are ordered by their arrival times. If a base data record is updated with a key change (for example, an employee number in Figure 8 is changed), or if a new record is inserted with the same alternate key value (department number in Figure 8), the new prime key pointer is added to the end of the alternate index record. In the case of a key change, the old pointer is deleted.

A prime-key pointer has the same length as the prime key field of the base data record it points to. The maximum number of pointers that can be associated with a given alternate key is 32 767, provided the maximum possible record length is not exceeded.


Base Data Records Where Prime Key= Employee Number
Alternate-Index Records Where Alternate Key= Department Number
4618
463871
514329
561777
674182
Prime-Key Pointers to Base Data Records

Employee Number	Name	Department Number	Other Information
463871	Martin, AB	4618	. . .
488797	Downs, CD	1201	. . .
514329	Michaels, EF	4618	. . .
561777	Price, GH	4618	. . .
568597	Sonders, IJ	2436	. . .
674182	West, KL	4618	. . .
⋮	⋮	⋮	. . .

Figure 8. Nonunique Alternate Keys

ALTERNATE INDEX MAINTENANCE

VSAM assumes alternate indexes are synchronized with the base cluster at all times and makes no synchronization checks during open processing; therefore, all structural changes made to a base cluster must be reflected in its alternate index or indexes. This maintenance is called index upgrade. You can maintain your own alternate indexes or you can have VSAM maintain them. When the alternate index is defined with the UPGRADE attribute of the DEFINE command, VSAM updates the alternate index immediately, when there is a change to the associated base data cluster. VSAM opens all the UPGRADE alternate indexes for a base cluster whenever the base cluster is opened for output (but not control interval processing).

All the alternate indexes of a given base cluster that have the UPGRADE attribute belong to the upgrade set. The upgrade set is updated whenever a base data record is inserted, erased, or updated. The upgrading is part of a request and VSAM completes it before returning control to your program. If the upgrade fails because of a logical error, VSAM attempts to nullify any modifications made to the base data or to other alternate indexes, and the request that caused the upgrade is rejected.

If you specify NOUPGRADE of the DEFINE command when the alternate index is defined, you must provide a way to reflect insertions, deletions, and changes made to the base cluster in the associated alternate index.

When a path is opened for update, JCL allocates the base cluster and all the alternate indexes in the upgrade set. If allocating the alternate indexes is unnecessary, you can specify NOUPDATE of the DEFINE command and cause JCL to allocate only the base cluster. VSAM, in that case, does not automatically upgrade.

The NOUPDATE specification may be nullified if the path is opened for both UPDATE and NOUPDATE concurrently.

PROCESSING VSAM DATA SETS

Processing options include:

- Types of access (keyed or addressed, and sequential, skip sequential, or direct)
- Exit routines for special processing

You can gain access to a data set with a mixture of options. For instance, if you were processing two portions of a data set concurrently, you might process one portion directly, asynchronously, using a work area; you might process the other sequentially, synchronously, in the I/O buffer. You could also alternate among the options to process a data set, switching, for example, from direct to sequential access when you got to a point where you wanted to process records in ascending sequence.

Processing options are specified in macros that generate control blocks when your program is assembled (ACB, EXLST, and RPL macros) or executed (GENCB macro). Each request for some action is associated with a request parameter list, which, in association with other control blocks, supplies the processing options for the request. For a description of the macro instructions and the specifications of the processing options, see VSAM Reference.

When you issue a request for a record, you can either wait until the request is completed to continue processing (synchronous) or go on with processing that is not dependent upon the first request while it is being carried out (asynchronous). Overlapping processing in this way can improve the performance of your job.

VSAM can concurrently keep track of positions in a data set for many requests to a data set. You can thus process many portions of a data set during the same period of time. Such concurrent access may be used to increase throughput, where each request can be processed independently of the others.

The standard request for access retrieves, stores, or deletes a single record. The standard request is described by a parameter list that indicates a single record. By chaining parameter lists together, you can retrieve or store many records with one request. You may not use chained parameter lists to update or delete records; you may use chained parameter lists only to retrieve records or to store new records.

TYPES OF ACCESS

VSAM allows both sequential and direct access for each of its three types of data sets. Sequential access of a record depends on the position, with respect to the key, the relative byte address of the previously processed record, or the relative record number; direct access does not. During sequential access, records retrieved by key are in key sequence, records retrieved by RBA are in entry sequence, and records retrieved by relative record number are in relative record number sequence. To retrieve records after initial positioning, you don't need to specify a key, an RBA, or a relative record number. VSAM automatically retrieves or stores the next record in order, either next in key sequence, next in entry sequence, or next in relative record number sequence, depending on whether you're processing by key, by RBA, or by relative record number.

With direct access, the retrieval or storage of a record is not dependent on the key, the RBA, or the relative record number of any previously retrieved record. You must fully identify the record to be retrieved or stored by key, by RBA, or by relative record number.

GET-previous processing is a variation of normal keyed or addressed sequential processing. Instead of retrieving or updating the next record in ascending sequence (relative to current positioning in the data set), GET-previous processing returns or updates the next record in descending sequence. You can select GET-previous processing for POINT, GET, PUT (update only), and ERASE operations. GET-previous processing is not permitted with control interval or skip-sequential processing.

When GET-previous processing is specified with either a POINT or a GET-direct request, the exact key of the request record must be specified.

VSAM allows a processing program or its subtasks to process a data set with multiple concurrent sequential and/or direct requests, each requiring that VSAM keep track of a position in the data set, with a single opening of the data set. Access can be to the same part or to different parts of a data set.

You can use a reusable VSAM data set as a work file, if the data set does not have an alternate index and is not associated with key ranges. That is, you can treat a filled data set as if it were empty and use it again and again regardless of its old contents. To reuse a data set, you need only to define it as reusable and specify that it be reset when you open it.

For a key-sequenced data set the primary form of access is keyed access, using an index or sequential access. For an entry-sequenced data set without an alternate index, the only forms of access are addressed (using the RBA determined for a record when it was stored in the data set), sequential access, and control-interval access. For a relative record data set, the only forms of access are keyed (using the relative record number as the key) and control interval access. Control interval access is described in "Chapter 6. Options for Advanced Applications" on page 81.

If you use addressed access to process key-sequenced data, you should consider the possibility that RBAs may have changed during previous keyed access.

For examples of keyed and addressed retrieval, storage, deletion, and update, see "Request Macros" in VSAM Reference.

RETRIEVE BY KEY

Keyed sequential access for a key-sequenced data set depends on where the previous macro request positioned VSAM with respect to the key sequence defined by the index. When your program opens the data set for keyed access, VSAM is positioned at the first record in the data set in key sequence to begin keyed sequential processing. The POINT macro instruction positions VSAM at the record whose key you specify. If the key is a leading portion of the key field, a generic key, the record positioned to is the first of the records having the same generic key. A subsequent sequential GET macro retrieves the record VSAM is positioned at. The GET then positions VSAM at the next record in key sequence. VSAM checks positioning when processing modes are changed between requests. The POINT macro can position for either forward or backward processing, depending on whether FWD or BWD was specified for the RPL OPTCD operand.

When you are processing by way of a path, records from the base cluster are returned according to ascending or, if you are retrieving the previous record, descending alternate key values. If there are several records with a nonunique alternate key, those records are returned in the order in which they were entered into the alternate index. VSAM sets a return code in the RPL when there is at least one more record with the same alternate key. For example, if there are three data records with the alternate key 1234, the return code would be set during the retrieval of records one and two and would be reset during retrieval of the third record.

Keyed sequential retrieval for a relative record data set causes the records to be returned in ascending or, if you are retrieving the previous record, descending numerical order, based on the current positioning for the data set. Positioning is established in the same way as for a key-sequenced data set, and the relative record number is treated as a full key. If a deleted record is encountered during sequential retrieval, it is skipped over and the next record is retrieved. The relative record number of the retrieved record is returned in the ARG field of the RPL.

Keyed Skip Sequential: When you indicate the key of the next record to be retrieved during skip-sequential retrieval, VSAM skips to the next record's index entry by using horizontal pointers in the sequence set to get to the appropriate sequence-set index record to scan its entries. The key of the next record to be retrieved must always be higher in sequence than the key of the preceding record retrieved.

Keyed direct retrieval for a key-sequenced data set does not depend on prior positioning; VSAM searches the index from the highest level down to the sequence set to retrieve a record. You can specify the record to be retrieved by supplying one of the following:

- The exact key of the record
- An approximate key, less than or equal to the key field of the record
- A generic key

You can use an approximate specification when you do not know the exact key. If a record actually has the key specified, VSAM retrieves it; otherwise, it retrieves the record with the next higher key. Generic key specification for direct processing causes VSAM to retrieve the first record having that generic key. If you want to retrieve all the records with the generic key, specify NSP in your direct request. That causes VSAM to position itself at the next record in key sequence. You can then retrieve the remaining records sequentially.

When GET-previous processing is specified with either a POINT or a GET-direct request, the exact key of the requested record must be specified.

When you use direct or skip-sequential access to process a path, a record from the base data set is returned according to the alternate key you have specified in the ARG operand of the RPL macro. If the alternate key is not unique, the record which was first entered with that alternate key is returned and a return code (duplicate key) is set in the RPL. To retrieve the remaining records with the same alternate key, specify the NSP option when retrieving the first record with a direct request and then switch to sequential processing.

To use direct or skip-sequential access to process a relative record data set, you must supply the relative record number of the record you want in the ARG operand of the RPL macro. If you request a deleted record, the request will cause a no-record-found logical error.

A relative record data set has no index; VSAM takes the number of the record to be retrieved and calculates the control interval that contains it and its position within the control interval.

DELETE BY KEY

An ERASE macro instruction that follows a GET for update deletes the record that the GET retrieved. A record is physically erased in the data set when you delete it. The space the record occupied is then available as free space.

You can erase a record from the base cluster of a path only if the base cluster is a key-sequenced data set. If the alternate index is in the upgrade set (that is, UPGRADE was specified when the alternate index was defined), it is modified automatically when you erase a record. If the alternate key of the erased record is unique, the alternate index data record with that alternate key is also deleted.

You can erase a record from a relative record data set after you have retrieved the record for update. The record is set to binary zeros and the control information for the record is updated to indicate an empty slot. You can reuse the slot by inserting another record of the same length into it.

STORE BY KEY

To store records in ascending key sequence throughout a data set, you can use sequential, skip-sequential, or direct access. For sequential or skip-sequential processing, VSAM scans the sequence set of the index; for direct processing, VSAM searches the index from top to bottom.

After a data set is created and initially loaded, it must be closed and reopened before update or insert requests can be issued.

A PUT macro instruction stores a record. A PUT for update following a GET for update stores the record that the GET retrieved. To update a record, you must previously have retrieved it for update. A PUT for nonupdate inserts or adds a new record into the data set.

When VSAM detects that two or more records are to be inserted in sequence into a collating position (between two records) in a data set, VSAM uses a technique called mass sequential insertion to buffer the records being inserted, thereby reducing I/O operations. Using sequential instead of direct access in this case enables you to take advantage of this technique. You can also extend your data set (resume loading) by using sequential insertion to add records beyond the highest key or relative record number. There are possible restrictions to extending a data set into a new control area depending on the sharing options you specify. See "Chapter 5. Sharing a VSAM Data Set" on page 69.

Mass sequential insertion observes control interval and control area freespace specifications when the new records are a logical extension of the control interval or control area (that is, when the new records are added beyond the highest key or relative record number used in the control interval or control area).

Sequential insertion in a relative record data set causes a record to be assigned the next available number in sequence, which is the next available relative record number greater than the position established by a previous record. The assigned number is returned in the ARG field of the RPL.

Direct or skip-sequential insertion of a record into a relative record data set causes the record to be placed as specified by the relative record number in the ARG field of the RPL. You must insert the record into a slot that does not contain a record. If the slot specified does contain a record, VSAM sets an error return code in the RPL and rejects the request.

You can insert and update data records in the base cluster by way of a path provided:

- The PUT request does not result in nonunique alternate keys in an alternate index which you have defined with the UNIQUEKEY attribute.

- You do not change the key of reference between the time the record was retrieved for update and the PUT is issued. The prime key is never changed.

If the alternate index is in the upgrade set (that is, you specified UPGRADE when you defined the alternate index), the alternate index is modified automatically when you insert or update a data record in the base cluster. If the updating of the alternate index results in an alternate-index record with no pointers to the base cluster, the alternate-index record is erased. If the updating creates a nonunique key in the alternate index, VSAM sets a nonerror return code in the RPL. If the alternate index has the UNIQUEKEY attribute, VSAM sets an error return code in the RPL and rejects the update request.

RETRIEVE BY ADDRESS

Positioning for addressed sequential retrieval is done by RBA rather than by key. When a processing program opens a data set for addressed access, VSAM is positioned at the record with RBA of zero to begin addressed sequential processing. A POINT positions VSAM for sequential access beginning at the record whose RBA you have indicated. A sequential GET causes VSAM to retrieve the data record at which it is positioned and positions VSAM at the next record in forward or backward direction.

With direct processing, you can optionally specify NSP in your RPL to indicate that the position be maintained following the GET. Your program can then process the subsequent records sequentially in either a forward or backward direction.

Addressed sequential access retrieves records in forward or backward direction. If addressed sequential retrieval is used for a key-sequenced data set, records will not be in their key sequence if there have been control interval or control area splits. Addressed direct retrieval requires that the RBA of each individual record be specified, because previous positioning is not applicable. The address specified for a GET or a POINT must correspond to the beginning of a data record; otherwise, the request is invalid.

DELETE BY ADDRESS

The ERASE macro can be used only with a key-sequenced data set to delete a record that you have previously retrieved for update.

With an entry-sequenced data set, you are responsible for marking a record you consider to be deleted. As far as VSAM is concerned, the record is not deleted. You can reuse the space occupied by a record marked as deleted by retrieving the record for update and storing in its place a new record of the same length.

STORE BY ADDRESS

VSAM does not insert new records into an entry-sequenced data set, but adds them at the end. With addressed access of a key-sequenced data set, VSAM does not insert or add new records.

After a data se[illegible]s created and initially [illegible]ded, it must be closed and reopened before update or addressed direct requests can be issued.

A PUT macro instruction stores a record. A PUT for update following a GET for update stores the record that the GET retrieved. To update a record, you must previously have retrieved it for update. You can update the contents of a record with addressed access, but you cannot alter the record's length. Neither can you alter the prime key field of a record in a key-sequenced data set.

To change the length of a record in an entry-sequenced data set, you must store it either at the end of the data set (as a new record) or in the place of an inactive record of the same length. You are responsible for marking the old version of the record as inactive. Figure 9 compares the three types of VSAM data sets.

Key-Sequenced Data Set	**Entry-Sequenced Data Set**	**Relative Record Data Set**
Records are in collating sequence by key field	Records are in order in which they are entered	Records are in relative record number order
Access is by key through an index or by RBA	Access is by RBA	Access is by relative record number, which is treated like a key
May have one or more alternate indexes	May have one or more alternate indexes	May not have alternate indexes
A record's RBA can change	A record's RBA cannot change	A record's relative record number cannot change
Distributed free space is used for inserting records and changing their length in place	Space at the end of the data set is used for adding records	Empty slots in the data set are used for adding records
Space given up by a deleted or shortened record is automatically reclaimed within a control interval	A record cannot be deleted, but you can reuse its space for a record of the same length	Space given up by a deleted record can be reused
Can have spanned records	Can have spanned records	Cannot have spanned records
Can be reused as a work file unless it has an alternate index, is associated with key ranges.	Can be reused as a work file unless it has an alternate index, is associated with key ranges.	Can be reused as a work file

Figure 9. Comparison of Key-Sequenced, Entry-Sequenced, and Relative Record Data Sets

CHAPTER 2. DATA SET PROCEDURES

USING ACCESS METHOD SERVICES

Access method services is a service program that is used with VSAM. Several commands and parameters are available to establish and maintain catalogs and data sets. Access method services consists of the following functions (commands):

ALTER Allows you to change attributes in previously defined catalog entries, for example, to change data set passwords.

BLDINDEX Creates (loads) an alternate index for a base cluster. The alternate index for the base cluster must have at least one record. BLDINDEX reads all base cluster records and extracts the primary keys (for key-sequenced data set's base cluster) or the RBAs (for entry-sequenced data set's base cluster), and the related alternate keys. This information is then sorted and loaded into the alternate index.

CHKLIST Is not a VSAM command and is not used for normal VSAM processing. However, 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 that were open at the time the checkpoint was taken, identifying the tape data sets that need to be mounted for restart.

CNVTCAT Converts VSAM catalog entries or OS CVOL entries into ICF catalog entries.

DEFINE Creates catalogs and catalog entries for alternate indexes, clusters, paths, and non-VSAM data sets. Entries for aliases, generation data groups, and page spaces can also be created. DEFINE creates the catalog entry for a VSAM object and allocates space for this object.

DELETE Deletes catalogs and catalog entries.

DIAGNOSE Scans the catalog or a VSAM volume data set to validate the data structures to detect structure errors.

EXPORT Unloads VSAM data sets and ICF catalogs for backup and/or transportation. The EXPORT command has, among others, two important options: TEMPORARY and 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 catalog to copy all related information such as cluster entry, data entry, and index entry (for a key-sequenced data set only) from the catalog to the target data set.

This command can also be used to disconnect a user catalog from a master catalog and other user catalogs (if a user catalog is disconnected from the master catalog by deleting its connector entry, it cannot be accessed from that system).

IMPORT Reloads VSAM data sets and ICF catalogs for backup and/or transportation. IMPORT redefines and reloads the data set. This command can also be used to reconnect a user catalog to a master catalog (if a user catalog is disconnected from the master catalog by deleting its connector entry, it cannot be accessed from that system).

LISTCAT Lists the catalog entries or parts of entries about VSAM and non-VSAM data sets. Executing a LISTCAT command after any DEFINE command is recommended.

PRINT Prints data sets, or parts of these data sets. The printout can be obtained in hexadecimal, character, or dump format.

REPRO Transfers data between two data sets or catalogs. If data is to be transferred into an empty data set, it always reorganizes the target data set.

When copying a relative record data set to a nonrelative record data set, empty slots are not copied. The slot numbers of the nonempty slots are lost.

When copying a relative record data set to a relative record data set, each empty slot in the source is copied to the same slot in the target.

The different functions are:

- Add records to the end of an entry-sequenced data set.
- Copy a catalog (move a catalog to another disk).
- Load/copy a VSAM data set.

  ISAM--->VSAM (convert an ISAM data set to VSAM format)
  SAM---->VSAM (load a VSAM data set)
  VSAM--->VSAM (copy/merge data)

- Merge records into a key-sequenced data set or relative record data set.
- Punch or print VSAM data sets.
- Unload/reload a catalog (back up).
- Merge or split ICF catalog entries into another ICF catalog.

VERIFY Ensures that 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). It should be used after on OPEN-error caused by a previous system failure or an abend condition while updating the data set.

VERIFY cannot be used for an empty data set (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 loading of the data set with the SPEED or RECOVERY option specified, or while reloading a reusable data set specifying the REUSE option.

For implicit verify processing, when OPEN detects a condition requiring a VERIFY, it issues the VERIFY itself unless told not to do so by the user. The warning message is still issued. If the VERIFY is done by OPEN, an informational message is issued that tells whether the VERIFY is successful or unsuccessful.

This procedure does not happen when improved control interval processing of the ACB macro or reset processing is specified. Under improved control processing, the "VERIFY required" indicator is left on, so that the verify is done when the data set is opened for processing with other than improved control interval or reset processing. Under reset processing, all previously existing data is removed, so it is not necessary to do a VERIFY.

Note: Implicit verify updates only the in-storage control blocks.

CREATING A CLUSTER (DEFINE CLUSTER)

The DEFINE command creates entries in the catalog for the cluster, its data component, and, if the cluster is indexed, its index component.

Attributes of the components in a cluster can be specified separately from attributes of the cluster.

- If attributes are specified for the cluster as a whole and not the components, the attributes of the cluster (except for its passwords and other protection attributes) apply to the components.
- If an attribute that is applicable to the data or index component is specified for both the cluster and the component, the explicit component specification overrides the cluster's specification.

You specify a name for the cluster when defining it. Generally this name is given as the dsname in JCL. You can, optionally, name the components of a cluster. Naming the data component of an entry-sequenced cluster or the data and index components of a key-sequenced cluster makes it easier to process them individually. For instance, you may open the index of a key-sequenced data set and process it as data. (This processing is described in "Chapter 6. Options for Advanced Applications" on page 81.) To use the ALTER command to modify attributes of any component, identify that component by name. If you do not specify a name for the components of a cluster, VSAM generates a name in the form:

```
clustername.Tbbbbbbb.DFDyyddd.Taaaaaaa.Tbbbbbbb
```

where:

clustername is the first qualifier of the cluster name

yyddd is the date (year and Julian day)

aaaaaaabbbbbbb is the timestamp value

DUPLICATE DATA SET NAMES

It is possible to have the same data set name in more than one catalog. With multiple catalogs you should be sure a data set name in one catalog is not duplicated in another 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 so that its new name duplicates the name of another object in the same catalog. However, this does not prevent duplication of names from one catalog to another. "Order of Catalog Use: DEFINE," in Access Method Services Reference, describes the order in which one of the catalogs available to the system is selected to contain the to-be-defined catalog entry. When you define an object, you should ensure that the catalog the system selects is the catalog you want the object's entry in.

Duplication is not prevented when a user catalog is imported into a system; no check is made to determine whether the imported catalog contains an entry name that another catalog already in the system contains.

Note: If the first qualifier is the same for a qualified data set name (PAYROLL) as for an unqualified data set name (PAYROLL.DATA), the two cannot be placed in the same catalog. For example, PAYROLL and PAYROLL.DATA cannot exist on the same catalog.

DEFINING A KEYRANGE DATA SET

When defining a nonkeyrange key-sequenced data set, relative record data set, or entry-sequenced data set, the primary data volume(s) and index volume(s) must be mounted so space can be obtained from the VTOC via DADSM. If you are defining a keyrange key-sequenced data set, the following rules apply:

- There should be as many volume serial numbers in the volser list as there are keyranges. When a volser number is duplicated in the volser list, more than one keyrange is allocated space on the volume. If you name a data component, a generated name is used for each additional keyrange on a volume. If you do not name the data component, then all names on all volumes are generated. The names of user-named data components appear once on each volume in the Format-1 DSCB. Generated names are used on all volumes for the Format-1 DSCBs, but each name has a unique keyrange qualifier so the user can correlate the Format-1 DSCB and the keyrange. All volumes must be mounted to define the keyrange.

- When there are more volume serial numbers in the volume serial list than there are keyranges, the excess volumes are marked as candidates and are not required to be mounted. The candidate volumes are used for overflow records from any keyrange. The overflow name is the prefix name without the keyrange qualifier.

- If you specify the ORDERED parameter on the DEFINE command, a one-to-one correspondence exists between the volumes in the volser list and the keyranges. The first volume in the volser list contains the first keyrange, the second volume contains the second keyrange, and so on. If a volume cannot be allocated in the order specified by the volser list, the cluster definition is terminated with an error. All volumes that will contain keyranges must be mounted.

For a multivolume keyrange cluster, the name specified on the data component is used for the first keyrange on each volume. If more than one keyrange resides on a volume, a special keyrange qualifier is appended to a generated name.

If a duplicate generated name is found on a volume, the keyrange qualifier starts with the letter "B" or "C" and so on, until a unique name is found.

With a multivolume key-sequenced data set, you may assign data to the various volumes according to ranges of key values. For example: if you have three volumes you might assign records with keys A through E to the first volume, F through M to the second, and N through Z to the third. Keyrange allocation facilitates processing the data set with only one of the volumes mounted: You know which volume contains what records. All the volumes specified for the cluster's data component or for the index component must be of the same device type. However, the data component and the index component can be on different device types.

SPECIFYING CLUSTER INFORMATION

All the necessary descriptive information and the performance, security, and integrity options are specified when you create a cluster. The information can apply to the data, the index, or both. Information for the cluster as a whole is specified in the CLUSTER parameter. Information for data only or index only is specified in parameters of DATA or INDEX.

Descriptive Information

Descriptive information includes:

- Type of data organization
- Whether the cluster is reusable for temporary storage of data
- Average and maximum lengths of data records
- Length and position of the key field in the records of a key-sequenced data set
- Identity of the catalog in which to define the cluster
- Identity of the volume(s) on which space is allocated for the cluster
- Amount of space to allocate for the cluster and whether the cluster is to reside in a separate data space
- Least amount of I/O-buffer space a processing program provides to process the data set

If the space to be provided for buffers is not specified, VSAM determines the control interval size, then sets a buffer space amount equal to the size of two data control intervals plus, for a key-sequenced data set, one index control interval. If the space is specified, the data and index control intervals are limited to sizes that allow the buffer space to hold two data control intervals and one index control interval.

If the values you specify for record length and key length require control intervals too large for the buffer space, the DEFINE command will fail.

Performance Options Information

Information for performance options (usually for an indexed cluster) includes:

- Whether records can span control intervals.
- Whether to replicate index records.
- Whether to place the sequence set of an index adjacent to data.
- Whether to place the cluster's index on a separate volume from data.
- The amount of free space to remain in the data component's control intervals and control areas when the data records are loaded. (This applies only to a key-sequenced cluster's data component)
- The control interval size for VSAM to use (instead of calculating the size.)

In a system with the IBM Mass Storage System, you can also indicate how a cluster or component that is stored on a mass storage volume is to be staged.

Protection and Integrity Information

Information for protection and integrity options includes:

- Passwords and related information
- Identity of your own authorization routine to verify that a requester has the right to gain access to data
- Identity of an I/O error-handling routine (the exception exit routine) that is branched to before the program's SYNAD exit
- Whether to preformat control areas during data record insertion
- Whether to verify that write operations are without error
- Whether and to what extent data is shared among systems, jobs, and subtasks
- Whether to erase the information a data set contains when you delete the data set

In a system with the Mass Storage System, you can indicate whether a cluster or component that is stored on a mass storage volume is to be destaged synchronously or asynchronously with respect to the program that closes it.

PREFORMATTING CONTROL AREAS

When you define a cluster, you can indicate that VSAM is to preformat each control area as records are loaded into the cluster (RECOVERY) or not preformat them, in the interest of performance (SPEED) during initial data set load. Preformatting clears all previous information from the direct access storage area and writes an end-of-file indicator:

- For an entry-sequenced or relative record data set, in every control interval in the control area.
- For a key-sequenced data set, in the first control interval in the control area following the preformatted control area. (The preformatted control area contains free control intervals.)

As records are loaded into a preformatted control area, a following end-of-file indicator indicates how far loading in a noncreate load mode has progressed. If an error occurs that prevents loading from continuing, you can identify the last successfully loaded record and resume loading at that point.

Without preformatting, an end-of-file indicator is written only after the last record is loaded. If an error occurs that prevents loading from continuing, you may not be able to identify the last successfully loaded record. You may have to reload the records from the beginning.

USER RESTRICTIONS DURING CREATE (LOAD) MODE

The terms "create mode," "load mode," and "initial data set load" are synonymously applied to the process of placing records into an empty VSAM data set. This type of processing is initiated when VSAM OPEN is called to open a data set whose high-used RBA is zero. It continues while records are added following the (successful) open and concludes when the data set is closed.

Certain restrictions apply during load mode processing. You should be aware of the following:

- Direct processing is not permitted (except relative record keyed direct).

  Note: If your application calls for direct processing during create mode, you can avoid this restriction by doing the following:

  1. Open the empty data set for create mode processing
  2. Sequentially write one or more records. (These may be "dummy" records.)
  3. Close the data set to terminate create mode processing.
  4. Reopen the data set to do your normal processing.

- Only use PUT and CHECK during create mode.
- Do not use improved control interval processing.
- Terminate create mode via CLOSE before the data set is used for other processing.
- Only specify one string in the ACB (STRNO>1 is not permitted).
- Do not specify local shared resources (LSR) or global shared resources (GSR).
- Data set opened for input is not allowed.

EXIT ROUTINES FOR SPECIAL PROCESSING

An exit is a branch that VSAM takes to an optional user-supplied routine when certain unusual conditions occur or when certain recurrent but unpredictable events happen. Exits are defined for:

- Time stamp processing (IDATMSTP) is used to update the most recently referenced date field in the data set control block (DSCB) for ICF catalogs.
- Logical error (LERAD) is used when the processing program makes an invalid request for access to data.
- Physical error (SYNAD) is used to handle physical error conditions.
- Exception handling (EXCEPTIONEXIT) monitors physical error conditions on a data set basis. This exit is specified via the access method services DEFINE command; it is taken before a SYNAD exit if both are specified.
- End of data set (EODAD) is used when the processing program has attempted to point to or retrieve sequentially a record beyond the last record in the data set.
- Journaling a transaction or keeping track of RBA change (JRNAD) is used to keep track of any change to the RBAs of records.
- Returning to a user's exit routine for special processing (UPAD) before a synchronous VSAM request completes.
- User-security-verification (USVR) is used to make security checks in addition to verification of passwords.

The routine to which VSAM exits may be a subroutine in the processing program or a separate load module. An exit routine is identified as available for use in an exit list associated with one or more access method control blocks. For information on how the exit list is created, modified, tested, and displayed, see VSAM Reference. See "Chapter 8. User-Written Exit Routines" on page 129 for detailed information about the exit routines.

DEFERRED AND FORCED WRITING OF BUFFERS

For integrity reasons, it is sometimes desirable to force the data buffer to be written after a PUT operation. At other times, it is desirable to defer the writing of a buffer as long as possible to improve performance. The following table shows when forced writing will occur and when writing will be deferred, based on the OPTCD at the time the PUT is issued. An ERASE request also follows the same buffer writing rules as the PUT request.

	SEQ	**SKP**	**DIR NSP**	**DIR (NO NSP)**
Force writes				X
Defer writes	X	X	X	

If LSR and GSR deferred writes are not specified, an ENDREQ macro always forces the data buffer to be written.

RECORD INSERTIONS

Record insertions in VSAM data sets occur in several ways:

Type I	PUT DIR,NSP
Type II	PUT DIR,NUP
Type III	PUT SEQ,NUP or NSP
Type IV	PUT SKP,NUP or NSP

Insertions into a key-sequenced data set use the free space provided during the definition of the data set or the free space that develops as a result of control interval and control area splits. Type III insert requests are used to create a data set or to do mass insertions. This type of insertion maintains free space during create mode and during mass insertions. This request type uses the sequential insert strategy. All the other types use the direct insert strategy. If MACRF=SIS is specified in the ACB, all inserts use sequential insert strategy.

Using sequential insert strategy, a record is inserted as follows:

- If the new record goes after the last record of the control interval and the free space limit has not been reached, the new record will go into the existing control interval. If the free space does not exist in the control interval, then a control interval split will occur at the point of insertion.

- If the new record does not belong at the end of the control interval and there is free space in the control interval, it will be placed in sequence into the existing control interval.

Using direct insert strategy, a record is inserted as follows:

- A new record is inserted into an existing control interval if free space exists in the control interval. If no free space exists, the control interval is split in half.

Sequential insert strategy results in better performance than direct insert strategy; fewer I/O operations are required by VSAM. Therefore, when a group of records is to be inserted into a data set between two existing records, the sequential insert strategy should be used. When several groups of records in sequence are to be mass inserted, each group may be preceded by a POINT KEQ to establish positioning.

For an entry-sequenced data set, records can be added only at the end of the data set.

Insertions into a relative record data set go into empty slots.

MULTI-STRING PROCESSING

In multiple string processing, there may be multiple independent RPLs within a region or partition for the same data set. The data set may be shared by a common control block structure by multiple tasks. There are several ACB and RPL arrangements to indicate that multiple string processing will occur:

- In the first ACB opened, STRNO or BSTRNO is greater than 1.
- Multiple ACBs are opened for the same data set within the same partition or region and are connected to the same control block structure.
- Multiple concurrent RPLs are active against the same ACB using asynchronous requests.
- Multiple RPLs are active against the same ACB using synchronous processing with each requiring positioning to be held.

If you are doing multiple string update processing, you need to be aware of VSAM look-aside processing and the rules surrounding exclusive use. Look-aside means VSAM checks its buffers to see if the control interval is already present when referring to an index or data control interval. Look-aside proceeds as follows:

1. For a nonupdate GET request, there is no look-aside across strings. As a result, a down-level copy of the data may be obtained either from buffers attached to this string or from secondary storage.
2. For GET update requests, there is a complete look-aside across all strings associated with the ACB. This may lead to an exclusive control conflict, because of update activity to the same control interval under other strings.

The exclusive use rules are as follows:

1. If a given string obtains a record with a GET UPD request, the control interval is not available for update or insert processing by another string.
2. If a given string is in the process of a control area split caused by an update with length change or an insert, that string obtains exclusive control of the entire control area being split. Other strings cannot process insert or update requests against this control area until the split is complete.

Because VSAM does not queue requests that have exclusive control conflicts, user action is required. If a conflict is encountered, VSAM returns a logical error return code, and you must stop activity and clear the conflict. If the RPL that caused the conflict had exclusive control of a control interval from a previous request, you should issue an ENDREQ against it before you attempt to clear the problem. You can clear the conflict in one of two ways:

1. Queuing until the RPL holding exclusive control of the control interval releases that control and then reissue the request or

2. Issue an ENDREQ against the RPL holding exclusive control to force it to release control immediately.

Note: If your RPL has provided a correctly specified MSGAREA and MSGLGN, the address of the RPL holding exclusive control is provided in the first word of the MSGAREA. Your RPL field, RPLDDDD, will contain the RBA of the requested control interval.

MULTI-STRING INDEX BUFFERS

Each string requires one index buffer. If there are four strings active, there is a minimum of four index buffers. Buffers in excess of the minimum are used for index set control intervals and are shared among the strings. The number of index buffers should be set to the number of strings (STRNO) plus X, where:

X=0, if all strings are sequential;

X=1, if the data set is a 2-level index and any string is not sequential;

X=n, where n is the number of index control intervals in the index set, if any string is doing random accessing, the number of index levels is greater than 2, and if the entire high-level index fits in storage;

X=1 plus the number of nonsequential strings, if the entire high-level index won't conveniently fit in storage.

For example, assume you have the following situation:

- 1024-byte index control interval
- 3-level index
- 50 index control intervals at the second level
- 4 strings doing random processing

Then, set BUFNI=STRNO+1+b=9, where b is the number of nonsequential strings. It is usually best to round this number up to the next 4K-byte multiple. That is, BUFNI=12 (12K bytes of index buffers). If you wanted to keep the entire high-level index in storage, then BUFNI would be set to 55 (50+1+4). This would be rounded to BUFNI=56 and would require 56K bytes for index buffers.

MULTI-STRING DATA BUFFERS

One data buffer per string and one additional buffer are required as a minimum per data set. Extra data buffers are used by sequential strings or for read-ahead on a first come, first served basis. When the extra buffers are released by a string (by issuing ENDREQ or a DIR request that releases positioning), they may be used by another string. Consider this example:

- Three strings (two direct and one sequential)
- 3-level index (five control intervals in the index set)
- 1024-byte index control interval
- 2048-byte data control interval

Set BUFNI=8 (8K for index buffers) and BUFND=6 (12K for data buffers). Each direct string uses one data and one index buffer. The sequential string uses one index and three data buffers. There is one data buffer reserved for insert requests that cause a control interval split, and the index set uses the extra five index buffers.

REQUEST POSITIONING

Some operations retain positioning while others release it. In a similar way, some operations hold onto a buffer and others release it with its contents. The following table shows which RPL options result in the retention of data buffers and positioning, and which options result in the release of data buffers and positioning:

	SEQ	SKP	DIR NSP	DIR (No NSP)	DIR LOC
Buffers and Positioning Retained	X	X	X		X
Buffers and Positioning Released				X	

Notes:

1. GET SEQ for new control intervals releases the previous buffer.
2. The ENDREQ and ERASE DIR macros release data buffers and positioning.
3. Certain options that retain positioning and buffers upon normal completion may not do so if the request fails with an error code. To determine whether or not positioning is maintained in the case of a logical error, see the table "FDBK Codes (Logical Errors)" in VSAM Reference.

The operation that uses but immediately releases a buffer and does not establish positioning is:

```
  GET  DIR,NUP,MVE
```

CREATING AN ALTERNATE INDEX (DEFINE ALTERNATEINDEX)

An alternate index can be defined over a key-sequenced cluster or an entry-sequenced cluster. An alternate index cannot be defined to support a reusable cluster, a relative record cluster, a catalog, another alternate index, or a non-VSAM data set.

To build an alternate index, define and load its base cluster (use the DEFINE CLUSTER command), and then define the alternate index and relate it to the base cluster. The base cluster and alternate index are described by entries in the same catalog. Issue the DEFINE ALTERNATEINDEX command to define an alternate index entry and allocate space for an alternate index. VSAM uses three catalog entries to describe the alternate index:

- An alternate index entry describes the alternate index as a key-sequenced cluster.
- A data entry describes the alternate index's data component.
- An index entry describes the alternate index's index component.

Attributes of the alternate index's components can be specified separately from the attributes of the alternate index. If attributes are specified for the alternate index as a whole and not for the components, these attributes (except for passwords and other protection attributes) apply to the components. If the attributes are specified for the components, they override attributes specified for the alternate index.

When you define the alternate index, specify an entry name. The entry name is the JCL DD statement's dsname and is processed with an access method services command. You can also name the alternate index's components so a user's program can open and process the alternate index's data or index component as a data set. For more details on this kind of processing, see "Chapter 6. Options for Advanced Applications" on page 81.

SPECIFYING ALTERNATE INDEX INFORMATION

When you define an alternate index, you specify descriptive information and performance, security, and data integrity options. The information can apply to the alternate index's data component, its index component, or the alternate index as a whole. Information for the alternate index as a whole is specified with the ALTERNATEINDEX parameter and its subparameters. Information for the data component or the index component is specified with parameters of DATA or INDEX.

Descriptive Information

Descriptive information includes:

- The base cluster related to the alternate index.
- Whether the alternate index entries are re-created, or defined for the first time.
- Whether the alternate index is reusable (rebuildable).
- Average and maximum lengths of alternate index records.
- Length and position of the key field in data records of the alternate index's base cluster.
- Identity of the catalog that contains the alternate index's entries and which must be the same catalog that contains the base cluster's entries.
- Identity of the volume(s) on which space is allocated for the alternate index.
- Amount of space to allocate for the alternate index.
- The least amount of I/O buffer space that a user's program is to provide when the program processes the alternate index's data.

You can use an alternate key value to define an alternate index with its records assigned to various volumes. Each volume can contain the alternate index records whose alternate key values are within a certain key range. For example, an alternate index might reside on three volumes. The records with alternate keys A through E are on the first volume, F through M on the second volume, and N through Z on the third volume.

The performance options and the protection and integrity information for the alternate index are the same as for the cluster information. See "Chapter 3. VSAM Performance Considerations" on page 44.

BUILDING AN ALTERNATE INDEX (BLDINDEX)

The BLDINDEX command is used to build an alternate index. Before you can build the alternate index, you must define and load its base cluster (see the DEFINE CLUSTER command), then define the alternate index and relate it to the base cluster (see the DEFINE ALTERNATEINDEX command). The base cluster and alternate index are described by entries in the same catalog.

The alternate index's volume and the base cluster's volume must be mounted during the build process. Any volumes identified with the WORKFILES parameter must also be mounted.

An alternate index can be built for a key-sequenced cluster or for an entry-sequenced cluster. The base cluster must be nonempty (that is, its high-used RBA value cannot be zero). Each record's alternate key value must be unique, unless the alternate index was defined with the NONUNIQUEKEY attribute.

HOW AN ALTERNATE INDEX IS BUILT

When an alternate index is built by BLDINDEX processing, access method services opens the base cluster to sequentially read the data records, sorts the information obtained from the data records, and builds the alternate index records:

1. The base cluster is opened for read-only processing. To prevent other users from updating the base cluster's records during BLDINDEX processing, include the DISP=OLD parameter in the base cluster's DD statement. If INDATASET is specified, access method services dynamically allocates the base cluster with DISP=OLD.

2. The base cluster's data records are read and information is extracted to form the key-pointer pair:

   - When the base cluster is entry-sequenced, the alternate key value and the data record's RBA form the key-pointer pair.

   - When the base cluster is key-sequenced, the alternate key value and the data record's prime key value form the key-pointer pair.

   If the base cluster's data records can span control intervals the alternate key must be in the record's first control interval.

3. The key-pointer pairs are sorted in ascending alternate key order. If your program provides enough virtual storage, access method services performs an internal sort. (The sorting of key-pointer pairs takes place entirely within virtual storage.)

   Use the following process to determine the amount of virtual storage required to sort the records internally:

   a. Sort record length = alternate key length + (prime key length (for a key-sequenced data set) or 4 (for an entry-sequenced data set)).

   b. Record sort area size = sort record length x number of records in the base cluster, rounded up to the next integer multiple of 2048, or a maximum of 32 768, whichever is greater.

   c. Sort table size = (record sort area size/sort record length) x 4.

   d. The sum of b + c = required amount of virtual storage for an internal sort. (This amount is in addition to the normal storage requirements for processing an access method services command.)

   If you do not provide enough virtual storage for an internal sort, or if you specify the EXTERNALSORT parameter, access method services defines and uses two sort workfiles and sorts the key-pointer pairs externally. Access method services uses the sort workfiles to contain most of the key-pointer pairs while it sorts some of them in virtual storage. An external sort workfile is a VSAM entry-sequenced cluster, marked reusable. The minimum amount of virtual storage you need for an external sort is:

       32 768 + ((32 768/sort record length) x 4)

The amount of space that access method services requests when defining each sort workfile is calculated as follows:

a. Sort records per block = 2041/sort record length

b. Primary space allocation in records = (number of records in base cluster/sort records per block) + 10

c. Secondary space allocation in records = (primary space allocation x 0.10) + 10

Both primary and secondary space allocation are requested in records with a fixed-length record size of 2041 bytes and a control interval size of 2048 bytes.

4. When the key-pointer pairs are sorted into ascending alternate key order, access method services builds an alternate index record for each key-pointer pair. If the NONUNIQUEKEY attribute is used and more than one key-pointer pair has the same alternate key values, the alternate index record contains the alternate key value, followed by the pointer values in ascending order. If the UNIQUEKEY attribute is used, each alternate key value must be unique.

   When the record is built, it is written into the alternate index as though it is a data record loaded into a key-sequenced cluster. Attributes and values to the load data records that can be specified when the alternate index is defined include:

   RECORDSIZE
   CONTROLINTERVALSIZE
   BUFFERSPACE
   FREESPACE
   WRITECHECK
   SPEED
   RECOVERY
   REPLICATE
   IMBED

5. When all alternate index records are built and loaded into the alternate index, the alternate index and its base cluster are closed. Steps 1 through 4 are repeated for each alternate index that is specified with the OUTFILE and OUTDATASET parameter. When all alternate indexes are built, any defined external sort workfiles are deleted. Access method services finishes processing and issues messages that indicate the results of the processing.

DD Statements That Describe the Sort Workfiles

VSAM data space available for the sort routine can be identified by specifying two dnames with the WORKFILES parameter and supplying two DD statements that describe the workfiles to be defined. Each workfile DD statement should be coded:

```
//ddname DD   DSNAME=dsname,VOL=SER=volser,
//            UNIT=devtype,DISP=OLD,AMP='AMORG'
```

ddname
: As specified in the WORKFILES parameter. If you do not specify the WORKFILES parameter and you intend to provide VSAM data space for external sort workfiles, identify the workfile DD statements with the names IDCUT1 and IDCUT2.

dsname
: A data set name. The scheduler generates a data set name for the workfile if none is provided.

VOL=SER=volser
: Required. Identifies the volume owned by the STEPCAT, JOBCAT, or master catalog where the workfile is cataloged.

The workfile's space is allocated from the volume's space. You can specify a maximum of five volumes for each workfile. See "How an Alternate Index is Built" for a description of how to calculate the amount of space to be allocated for each sort workfile. If your BLDINDEX job requires external sort workfiles, this space must be available on the volume(s) identified by volser or your job will fail.

UNIT=devtype
Type of direct access device on which the volume is mounted. You can specify a generic device type (for example, 3330) or a unit address (for example 121, 248). You cannot specify SYSDA.

DISP=OLD
Required.

AMP='AMORG'
Required.

If BLDINDEX is used interactively in a TSO environment, these sort workfile DD statements must be in the logon procedure.

DEFINING A PATH (DEFINE PATH)

The DEFINE PATH command is used to establish the relationship between an alternate index and its base cluster. The base cluster and its alternate index must be defined before defining the path that relates them.

When your program opens a path for processing, both the alternate index and its base cluster are opened. When data in a key-sequenced data set's base cluster is read or written using the path's alternate index, keyed processing is used. RBA processing is not allowed for key-sequenced data sets, only for an entry-sequenced data set's base cluster.

You can also use the DEFINE PATH command to establish a password-protected alias for a VSAM cluster. The cluster must be defined before defining a path as its alias. The path entry, as an alias, allows the cluster another set of protection attributes. Specify NOUPDATE access for the cluster, to open the cluster without also opening its upgrade set (all the cluster's alternate indexes that are to be updated whenever the cluster's data is changed). The NOUPDATE specification is overridden by opening the path, allowing sharing of a control block structure that permits update.

DEFINING A NON-VSAM DATA SET (DEFINE NONVSAM)

The DEFINE command can be used to catalog a non-VSAM data set in an ICF catalog. An already existing non-VSAM data set can be introduced into a master or user catalog through the DEFINE command. When a DEFINE command is used to define a non-VSAM entry, an entry is created in a master or user catalog. No space is allocated or reserved.

When you define, alter, or delete a non-VSAM data set in a password-protected catalog, the catalog's update- or higher-level password is required.

Specify the VOLUMES and DEVICETYPES parameters by using indirect volume serial identification and device type when the data sets are to reside on the system residence volume.

To change the device type, code DEVICETYPES(0000), and the field is resolved to the device type at SUPERLOCATE, LOCATE, and DELETE time. This allows you to use the non-VSAM data sets without recataloging them to point to the new volume.

Note: If you code DEVICETYPES(0000), you must also code VOLUMES('******'), or an error will result.

To change the system residence volume serial number, code VOLUMES('******') and the field is resolved in the system residence volume at SUPERLOCATE, LOCATE, and DELETE time. This allows you to use the non-VSAM data sets without recataloging them to point to the new volume.

For APF-authorized data sets, you must specify the volume serial number and device type explicitly.

DEFINING AN ALTERNATE NAME (DEFINE ALIAS)

The DEFINE command can be used to define an alternate name or an alias for a non-VSAM data set even if it is a generation data set. DEFINE ALIAS can also be used to define an alias for a user catalog connector in the master catalog. An alias cannot be defined for a generation data group.

DEFINING A GENERATION DATA GROUP (DEFINE GENERATIONDATAGROUP)

Use the DEFINE command to create a catalog entry for a generation data group. For further information on generation data groups, see Data Management Services.

Once a catalog entry for a generation data group has been created, existing data sets can be attached to it as generation data sets. This is accomplished through JCL, in the form DSNAME=name(n) (where n is the generation increment), or by defining each data set as a non-VSAM data set with a generation data set name, as follows:

```
  name.GnnnnVnn
```

If the name matches the name of a previously defined generation data group, the non-VSAM data set is associated with the generation data group as a generation data set. See "Defining a Non-VSAM Data Set (DEFINE NONVSAM)" on page 35 for a description of how to define a non-VSAM entry.

A generation data group can be cataloged in an ICF catalog. The model DSCB must exist on the generation data group's catalog volume.

DEFINING A PAGE SPACE (DEFINE PAGESPACE)

Use the DEFINE command to define an entry for a page space. A page space is a system data set that contains pages of virtual storage. The pages are stored into and retrieved from the page space by the auxiliary storage manager. A page space is a nonindexed data set (an entry-sequenced cluster) that is entirely preformatted before it is used. In a system with the Mass Storage System, a page space cannot be defined on a mass storage volume, but must reside on a single volume. A page space cannot be opened as a user data set.

A page space has a maximum usable size equal to 65 535 paging slots (records). See the description for space size declarations (CYLINDERS, RECORDS, and TRACKS for "DEFINE PAGESPACE") in Access Method Services Reference for further details.

You can define a page space in a user catalog, then move the catalog to a new system and establish it as the system's master catalog. When you define a page space in a user catalog, code a STEPCAT or JOBCAT statement to identify and allocate the catalog. A page space cannot be used if its entry is in a user catalog.

When you issue a DEFINE PAGESPACE command, the system creates an entry in the catalog for the page space, then preformats the page space. If an error occurs during the preformatting process (for example, an I/O error or an allocation error), the page space entry remains in the catalog even though no space for it exists. Issue a DELETE command to remove the page space entry before you redefine the page space.

Each page space is represented by two entries in the catalog: a cluster entry and a data entry. (A page space is conceptually an entry-sequenced cluster.) Both of these entries should be password protected if the page space is to be password protected.

Note: The passwords you specify with the DEFINE PAGESPACE command are put in both the page space's cluster entry and its data entry. When you define page spaces during system generation (sysgen), use the ALTER command to add passwords to each entry, because passwords cannot be specified during system generation.

Unless you ensure that the catalog containing the page space entry is password protected, a user can list the catalog's contents and find out each entry's passwords.

A page space is made known to the system as a system data set at sysgen or through members of a partitioned data set: SYS1.PARMLIB. To be used as a page space, it must be defined in a master catalog.

RESTORING END-OF-FILE VALUES (VERIFY)

When a data set is closed, its end-of-file (EOD) and end-of-key-range (EOKR) information is used to upgrade the data set's cataloged information. If a system failure occurs before the data set is closed (before the user's program issues CLOSE), the data set's cataloged information is not upgraded. The data set's real EOD and EOKR indicators are written in the data set, but are not shown in the data set's cataloged information.

When the data set is subsequently opened and the user's program attempts to process records EOD or EOKR, a read operation results in a "no record found" error, and a write operation might write records over previously written records.

The VERIFY command is used to compare the end-of-data-set and end-of-key range information in a catalog with the true end-of-file and end-of-keyrange. If the information in the catalog does not agree with the true end-of-file or end-of-key range, the catalog information is corrected. The VERIFY command can be used following a system failure that caused a component opened for update processing to be improperly closed. Clusters, alternate indexes, and catalogs can be verified. Paths over an alternate index cannot be verified. Paths defined directly over a base cluster can be verified. Although the data and index components of a key-sequenced cluster or alternate index can be verified, the timestamps of the two components are different following the separate verifies, possibly causing further OPEN errors. Therefore, use the cluster or alternate index name as the target of your VERIFY command.

To use the VERIFY command to verify a catalog, access method services must be authorized. For information about program authorization, see "Authorized Program Facility (APF)" in System Programming Library: Supervisor.

A user catalog is opened as a data set when restoring its end-of-file and end-of-keyrange information and a JOBCAT or STEPCAT DD statement must be supplied.

VERIFY cannot be used to correct catalog records for a key-sequenced data set or a relative record data set create (load) mode failure. An entry-sequenced data set defined with the RECOVERY attribute may be verified following a create (load) mode failure.

Note: Data sets must be verified unconditionally if they are shared between systems and can be extended.

COPYING A DATA SET (REPRO)

Use the REPRO command to do any of the following:

- Copy or merge a VSAM data set into another VSAM data set.
- Copy or merge a sequential data set into another sequential data set.
- Copy an alternate index as a key-sequenced VSAM data set.
- Copy a VSAM data set whose records are fixed length into an empty VSAM relative record data set.
- Convert a sequential or indexed-sequential data set into a VSAM data set.
- Convert a VSAM or indexed-sequential data set into a sequential data set.
- Copy a data set (other than a catalog) to reorganize it. Data set reorganization is an automatic feature.
- Merge two VSAM data sets.

For the remainder of the REPRO discussion, all these functions are referred to as copying.

VSAM data sets used as either input or output must be cataloged. Sequential and indexed-sequential data sets need not be cataloged.

If a sequential or indexed-sequential data set is not cataloged, include the appropriate volume and unit parameters on your DD statement. Also, supply a minimum set of DCB parameters when the input data set is sequential or indexed sequential, and/or the output data set is sequential. The following table shows the four key parameters.

Parameters	**Must be Supplied by User**	**Default if Not Supplied**
DSORG	IS	PS
RECFM	F, FB, V, VB	U
BLKSIZE	block size	none
LRECL	lrecl	BLKSIZE for F or FB BLKSIZE-4 for V or VB

The one parameter not supplied by default is BLKSIZE; you must supply this value. The DCB parameter DSORG must be supplied via the DD statement. The DCB parameters RECFM, BLKSIZE, and LRECL can be supplied via the DSCB or header label of a standard labeled tape, or by the DD statement.

For a variable record length VSAM data set, the 3-byte VSAM record definition field (RDF) is not included in the VSAM record length. When REPRO attempts to copy a VSAM record whose length is within 4 bytes of LRECL, a recoverable error occurs and the record is not copied.

Access method services does not support records greater than 32 760 bytes for non-VSAM data sets (LRECL=X is not supported). If the logical record length of a non-VSAM input data set is greater than 32 760 bytes, or if a VSAM data set defined with a record length greater than 32 760 is to be copied to a sequential data set, the REPRO command terminates with an error message.

Records in an indexed-sequential data set that have a fixed-length, unblocked format with a relative-key-position (RKP) of zero are preceded by the key string when used as input. The records in the output data set must have a record length defined that includes the extended length caused by the key string. To copy "dummy" indexed-sequential records (records with hexadecimal 'FF' in the first byte) specify the DUMMY option in the ENVIRONMENT parameter.

Because data is copied as single logical records in either key order or physical order, automatic reorganization takes place. The reorganization can cause any of the following:

- Physical relocation of logical records
- Alteration of a record's physical position within the data set
- Redistribution of free space throughout the data set
- Reconstruction of the VSAM indexes

Figure 10 on page 40 describes how the records from the input data set are added to the output data set when the output data set is an empty or nonempty entry-sequenced, key-sequenced, sequential, or relative record data set.

Note: The REPRO operation is terminated if:

- One physical I/O error is encountered while writing to the output data set
- A total of four errors is encountered in any combination of the following:
    - Logical error while writing to the output data set
    - Logical error while reading the input data set
    - Physical error while reading the input data set

When copying to a key-sequenced data set, the records to be copied must be in ascending order, with no duplicates in the input data set. With an entry-sequenced data set, the records to be copied can be in any order.

REPRO causes access method services to retrieve records from a sequential, indexed-sequential, or VSAM data set and store them in VSAM format in key sequence, record number sequence, entry sequence, or a sequential data set. When records are stored in key sequence, index entries are created and loaded into an index component as control intervals and control areas fill up. Free space is left as indicated in the data set definition in the catalog, and, if indicated in the definition, records are stored on particular volumes according to key ranges.

You can load all the records in one job or in several jobs. In subsequent jobs, VSAM stores records as before, extending the data set as required.

To use your own program to load a key-sequenced data set, first sort the records (or build them) in key sequence, then store them with sequential access (the PUT macro). For information on using macros to write your own program to load records into a data set, see "Chapter 8. User-Written Exit Routines" on page 129.

Type of Data Set	**Empty**	**Nonempty**
Entry-Sequenced/ Sequential	Creates new data set in sequential order.	Adds records in sequential order to the end of the data set. (DISP=MOD must be specified for SAM sequential.)
Key-Sequenced	Creates new data set in key sequence and builds an index.	Merges records by key and updates the index. Unless the REPLACE option is specified, records whose key duplicates a key in the output data set are lost.
Relative Record	Creates a new data set in relative record sequence, beginning with 1.	Records from another relative record data set are merged, keeping their old record numbers. Unless the REPLACE option is specified, a new record whose number duplicates an existing record number is lost. Records from any other type of organization cannot be copied into a nonempty relative record data set.

Figure 10. Adding Records to Various Types of Output Data Sets

PRINTING A DATA SET (PRINT)

Use the PRINT command to list part or all of a key-sequenced, relative record, or entry-sequenced VSAM data set, alternate index, ICF catalog, or non-VSAM data set. The components of a key-sequenced data set or an alternate index can be listed individually. To list a component of a key-sequenced data set or alternate index, specify the component name as the data set name. An alternate index is printed as though it were a key-sequenced cluster.

If a sequential or indexed-sequential data set is not cataloged, include the appropriate volume and unit parameters on your DD statements.

You must supply a minimum set of DCB parameters when the data set to be printed is sequential or indexed-sequential. The following table shows the four key parameters:

Paramaters	**Must be Supplied by User**	**Default if Not Supplied**
DSORG	IS	PS
RECFM	F, FB, V, VB	U
BLKSIZE	block size	none
LRECL	lrecl	BLKSIZE for F or FB BLKSIZE-4 for V or VB

The one parameter not supplied by default is BLKSIZE; you must supply this value. The DCB parameters can be supplied via the DSCB or header label of a standard labeled tape; otherwise, they must be supplied via the DD statement.

Access method services does not support records greater than 32 760 bytes for non-VSAM data sets (LRECL=X is not supported). If the logical record length of a non-VSAM input data set is greater than 32 760 bytes, the PRINT command terminates with an error message.

Sequential and entry-sequenced data sets are listed in physical sequential order. Indexed-sequential and key-sequenced data sets can be listed in key order or in physical sequential order. A base cluster can be listed in alternate key sequence by specifying a path name as the data set name for the cluster.

Only the data content of logical records is listed. System-defined control fields are not listed. Each record listed is identified by one of the following:

- The relative byte address (RBA) for entry-sequenced data sets
- The key for indexed-sequential and key-sequenced data sets, and for alternate indexes
- The sequential record number for sequential (non-VSAM) and relative record data sets

Note: If four logical and/or physical errors are encountered while trying to read the input, printing is terminated.

To use the PRINT command to print a catalog, access method services must be authorized. For information about program authorization, see "Authorized Program Facility (APF)" in _System Programming Library: Supervisor_.

LISTING TAPE VOLUMES MOUNTED AT CHECKPOINT (CHKLIST)

During processing, a program can issue the CHKPT macro to record information for use in restarting the processing if an error interrupts the program. Recording information by way of CHKPT is called taking a checkpoint. The records that contain the information make up a checkpoint entry in the checkpoint data set, which contains an entry for each checkpoint that is taken.

The checkpoint data set can be a sequential data set or a partitioned data set. In a partitioned data set, each checkpoint entry is a member of it. Further information on the CHKPT macro may be found in _Checkpoint/Restart_.

Checkpoint information includes the volume serial numbers of tape data sets opened at the checkpoint. The CHKLIST command enables you to list these volume serial numbers to identify the tape data sets that need to be mounted for restart.

For a checkpoint data set with DSORG=PS (sequential data set), you can select one or more specific checkpoint entries to list tape information by a single CHKLIST command. All checkpoint entries will be processed if no entry is selected.

You can use CHKLIST to process a checkpoint data set with DSORG=PO (partitioned data set) in the following manner:

- Specify DSNAME=dsname(member) on the JCL statement that defines the checkpoint data set.
- If you do not select a specific checkpoint entry, the single entry specified by member after dsname is processed.

The CHKLIST command lists the following information:

- The checkpoint identifier for the entry processed.
- The following items are listed for each tape data set open at the time of the checkpoint:
  - dsname
  - ddname
  - Type of unit on which the volume was mounted

- The sequence number of the mounted volume
- Volume serial numbers with an * by the volume serial number of the mounted volume

To process multiple members of a partitioned checkpoint data set, use the CHKLIST command once for each member.

Note: The CHKLIST command cannot be invoked as a TSO command.

ESTABLISHING BACKUP AND RECOVERY PROCEDURES

You should establish backup and recovery procedures for your data sets. All VSAM data sets must be cataloged. Because the physical and logical description of a data set is contained in its catalog entries, VSAM requires up-to-date catalog entries to access data sets.

Any recovery procedure must match both data set and catalog entry status. Recovery by way of reloading the data set automatically takes care of this problem: A new catalog entry is built when the data set is reloaded.

BACKING UP DATA

Access method services has two utility functions for creating backup copies:

- The EXPORT command is used to create an unloaded, portable copy of the data set. There are options for protection, and most catalog information is exported along with the data, easing the problem of redefinition. You can prevent the exported data set from being updated until the IMPORT command reestablishes its accessibility.
- The REPRO command is used to create a duplicate VSAM data set for backup. An advantage over EXPORT is the accessibility of the backup copy. A DEFINE command is required before reloading, but the original DEFINE statements can be used.

You can also write your own backup scheme, which should be integrated into the regular processing procedures whenever possible.

Any backup procedure that does not involve an image copy of the data set (such as EXPORT or REPRO) results in data reorganization and the recreation of the index for a key-sequenced data set. Any absolute references by way of RBA may become invalid.

Consider the following when using EXPORT or REPRO as a backup:

- A backup copy created by EXPORT can only be a sequential data set. A backup copy created by REPRO can be a VSAM data set or a sequential data set. The backup copy obtained by using REPRO can be a different type of VSAM data set than the original. For example, you could back up a VSAM key-sequenced data set by copying it to a VSAM entry-sequenced data set using REPRO. When using REPRO, the REUSE attribute allows repeated backups to the same VSAM reusable target data set.
- A backup copy created by EXPORT contains all the information necessary to redefine the VSAM cluster or alternate index when you IMPORT the copy. A backup copy created by REPRO does not contain this catalog information. To use the backup copy, REPRO requires you to delete the original data set, if it still exists, using the DELETE command. Next, you must redefine the data set using the DEFINE command, then restore it with the backup copy using the REPRO command. (The delete and define operations are not

necessary if the original data set still exists and was defined with the REUSE attribute.)

A key-sequenced data set is reorganized when it is transported or backed up using EXPORT and IMPORT or when it is copied using REPRO. The data records are rearranged physically in ascending key sequence (control interval and control area splits may place them physically out of order) and free-space quantities are balanced.

After replacing a damaged data set with its backup copy, rerun the jobs that updated the original between the time it was backed up and the time it became inaccessible. This updates the backup copy.

Backing up the data sets in a user catalog allows you to recover from damage to the catalog. You can import the backup copy of a data set whose entry is lost or redefine the entry and reload the backup copy. If the catalog is completely lost, you can redefine it, then import or redefine and reload all the data sets that were defined in the catalog.

Exporting and Importing a Data Set

Use the EXPORT command (with the TEMPORARY parameter) to copy a VSAM cluster and its catalog entries onto a movable volume (a demountable direct access volume or a magnetic tape volume). Then, use the IMPORT command to move the portable copy to another system that includes VSAM or to replace the original cluster and its catalog entries. The portable copy itself is inaccessible for processing.

Making a Copy of a Data Set

Use the REPRO command to copy a VSAM or a sequential data set into another newly defined (for protection) VSAM or sequential data set, in another catalog. The REPRO command does not copy the data set's catalog information. Use the REPRO command with the REPLACE parameter to merge the backup copy into the original data set, or delete and redefine the original data set and use REPRO to reload the backup copy into it. Because the backup copy is itself accessible for processing, you can replace the original with it. The catalog in which you defined the backup must be available for processing, and the name of the backup must be used.

If you periodically process a data set sequentially, you can easily create a backup copy as a by-product of normal processing. This backup copy can be used like one made by REPRO.

CORRECTIVE MEASURES

When part or all the online data is destroyed, you can replace the destroyed data set with its backup copy.

- Use the REPRO command with the REPLACE option to merge the backup copy with the destroyed data. When only part of the data set is damaged, specify REPRO parameters that allow the records in the damaged part of the data set to be replaced.

- Use the IMPORT command to totally replace a VSAM cluster whose backup copy was built using the EXPORT command. The IMPORT command uses the backup copy to replace the cluster's contents and catalog information.

CHAPTER 3. VSAM PERFORMANCE CONSIDERATIONS

This chapter describes many of the options and factors that either influence or, in some cases, determine VSAM's performance as well as the performance of the operating system. The main topics include control interval and control area size, buffer management, allocation units, distributed free space, and index options.

Most of the options are specified in the access method services DEFINE command when a data set is created. (The DEFINE command is described in Access Method Services Reference.) In some cases, options can be specified in the ACB and GENCB macro instructions and in the DD AMP parameter, all of which are described in this publication.

CONTROL INTERVAL SIZE

Control interval size, which can be specified for VSAM data sets, affects record processing speed and storage requirements in these ways:

- For data sets containing large data records, you might want larger control intervals, even though VSAM allows records to cross control interval boundaries.
- For data sets containing large control interval sizes, more buffer space is required in virtual storage for each control interval.
- For data sets containing large control interval sizes, fewer I/O operations (control interval accesses) are required to bring a given number of records into virtual storage; fewer index records must be read. This is usually significant only for sequential and skip-sequential access.
- Free space will probably be used more efficiently (fewer control interval splits and less wasted space) as control interval size increases relative to data record size, especially with variable-length records. (Free space in a control interval isn't used if there isn't enough for a complete data record.)

You can let the system select the size of a control interval for a data or index component or you can request a particular control interval size in the DEFINE command. The size you specify must, however, fall within acceptable limits determined by the system, or the DEFINE will fail. The limits depend on:

- The maximum size of the data records, which you specify by the required RECORDSIZE parameter of the DEFINE command
- The smallest amount of virtual storage space your processing programs will provide for I/O buffers, which you specify by the optional parameter BUFFERSPACE.

The size of a control interval may vary as follows:

- The size of a control interval must be a multiple of 512 bytes, because a control interval is a whole number of physical records and physical record size is 512, 1024, 2048, or 4096 bytes. (The physical record size of 2048 does not apply to the IBM 3340 Disk Storage.)
- The size of a control interval in the data component of a cluster can be any multiple of 512, up to 32 768, except that, if it is over 8192 bytes, it must be a multiple of 2048: 512, 1024, 1536, 2048, 2560, ..., 8192, 10 240, 12 288, or 32 768.

- A control interval in an index is the same size as a physical record, and its size is therefore restricted to 512, 1024, 2048, or 4096 bytes.

The information recorded on a track is divided into physical records that are limited by the capacity of a track. The physical record sizes that VSAM uses are 512, 1024, 2048, and 4096 bytes. (The physical record size of 2048 does not apply to the IBM 3340 Disk Storage.) Control interval size is limited by the requirements that it be a whole number of physical records (up to 64, or a maximum of 32 768 bytes) and that, if it is greater than 8192 bytes, it be a multiple of 2048. A data set whose control intervals correspond with the tracks of one device might have more or less than one control interval per track if it were stored on a different device. Figure 11 illustrates the independence of control intervals from physical records.

Control Interval	Control Interval	Control Interval
Physical Records		
Track 1	Track 2	Track 3

Figure 11 shows the physical block size selected by VSAM for all of the possible control interval sizes for the IBM 3330, 3340, and 3350 direct access devices, together with the resulting track utilization (all numbers are in K bytes):

	Physical Block Size			Track Space Used		
CI SIZE	**3340**	**3330**	**3350**	**3340**	**3330**	**3350**
.5	.5	.5	.5	6	10	13.5
1	1	1	1	7	11	15
1.5	.5	.5	.5	6	10	13.5
2	1	2	2	7	12	16
2.5	.5	.5	.5	6	10	13.5
3	1	1	1	7	11	15
3.5	.5	.5	.5	6	10	13.5
4	4	4	4	8	12	16
4.5	.5	.5	.5	6	10	13.5
5	1	1	1	7	11	15
5.5	.5	.5	.5	6	10	13.5
6	1	2	2	7	12	16
6.5	.5	.5	.5	6	10	13.5
7	1	1	1	7	11	15
7.5	.5	.5	.5	6	10	13.5
8	4	4	4	8	12	16
10	1	2	2	7	12	16
12	4	4	4	8	12	16
14	1	2	2	7	12	16
16	4	4	4	8	12	16
18	1	2	2	7	12	16
20	4	4	4	8	12	16
22	1	2	2	7	12	16
24	4	4	4	8	12	16
26	1	2	2	7	12	16
28	4	4	4	8	12	16
30	1	2	2	7	12	16
32	4	4	4	8	12	16

Figure 11. Physical Block Size for Control Intervals

If you specify a control interval size that is not a proper multiple, VSAM increases it to the next multiple. For example, 2050 is increased to 2560.

The size of a control interval in a data component must be large enough to hold a data record of the maximum size specified in the RECORDSIZE parameter unless the data set was defined with the SPANNED attribute. The minimum amount of control information in a control interval is 7 bytes. Therefore, a control interval is normally at least 7 bytes larger than the largest record in the component.

The use of the SPANNED attribute removes this constraint by allowing data records to be continued across control intervals. The maximum record size is then equal to the number of control intervals per control area multiplied by (control interval size minus 10). The use of the SPANNED attribute places certain restrictions on the processing options that can be used with a data set. For example, records of a data set with the SPANNED attribute cannot be read or written in locate mode.

Because VSAM transmits the contents of a control interval between direct access storage and virtual storage, the amount of space allowed for I/O buffers limits the size of a control interval. The BUFFERSPACE parameter of the DEFINE command indicates the smallest amount of virtual storage space a processing program provides for buffers.

BUFFERSPACE, if you specify it, limits control interval size to values such that the buffer space can hold at least two data control intervals and one index control interval. If you do not specify BUFFERSPACE, control interval sizes are set independently, and the buffer space value is then set equal to the size of two data control intervals and one index control interval.

If you don't specify a size for a data control interval, the system calculates a default value for the given average record size but at least large enough to accommodate the maximum record size. For a key-sequenced data set, after control interval size has been set, the system determines the number of bytes to be reserved for free space, if specified. For example, if control interval size is 4096, and the percentage of free space in a control interval has been defined as 20%, approximately 820 bytes are reserved.

With a key-sequenced data set, if you don't specify a size for index control intervals, the system uses 512, if possible. After the system determines the number of control intervals in a control area (see the next section), it estimates whether an index record is large enough to handle all the control intervals in a control area. If not, the size of an index control interval is increased, if possible. If it's not possible, the size of the control area is decreased by decreasing the number of control intervals.

To find out what values are actually set in a defined data set, you can issue the access method services LISTCAT command.

DATA CONTROL INTERVAL SIZE

Normally, a 4096-byte data control interval is reasonably good regardless of the DASD device used, processing patterns, or the processor. However, there are some special considerations that might affect this choice.

A given logical record size may fit some control interval sizes better than others. Generally, large control interval sizes provide the best fits. Also, some control interval sizes fit a track of a given device better than others. For example, on a 3340 track, a 2048-byte control interval yields a potential 7168 bytes of usable space per track, whereas a 4096-byte control interval yields 8192 bytes (2 control intervals) of data on a

3340 track. Assuming a 300-byte record, in one case there would be 21 records per track and in the other case there would be 24 records per track.

Random Processing

A small data control interval is preferable when random processing is predominant. In general, select the smallest data control interval that yields a reasonable space utilization. Normally, 1024- or 2048-byte control intervals are good.

Sequential Processing

If the processing is predominantly sequential, even larger data control intervals may be good choices. Given a 16K-byte data buffer space, it is better to read two 8K-byte control intervals with one I/O operation than four 4K-byte control intervals with two I/O operations. After insertions have occurred, very large data control intervals often result in fewer out-of-sequence control intervals than small control intervals.

Figure 12 summarizes the generally acceptable data control interval sizes.

Accessing Pattern	**Condition or Device**	**Data CI Size**
Random	3340	1024
	3330	1024 or 2048
Sequential	3340	4096 or 8192
	3330	4096 or 6144
Ordered Direct	If there are fewer than 2 records referenced per track.	Then choose the CI size as done for Random.
	Otherwise	4096
Random Batch or Sorted Batch	If number of records per group is less than the number of records in a 2048-byte CI.	Then choose the CI size as done for Random.
	Otherwise	4096

Figure 12. Data Control Interval Sizes

INDEX CONTROL INTERVAL SIZE

A 512-byte index control interval is usually the best choice. If the number of data control intervals per control area is small, the full key size is not too large, and if the key compresses well, then a 512-byte index control interval is possible. The best way to find out if 512 bytes is large enough is to run an experiment using the data control interval size chosen and 512 bytes for index control interval size. Allow (0,0) free space and load enough records to equal one control area. At the end of the run, list the catalog index entry. If there is one level of index, then the 512-byte index control interval was large enough.

Summary of Control Interval Size Strategy

For random processing, choose the smallest data control interval that provides for reasonable space utilization. Choose an index control area size that is compatible with the data control area size. When a choice between large data and index control interval sizes exists, choose the combination that yields the smallest buffer space value (data control interval size + index control interval size). This combination requires the least amount of active real storage and results in the least amount of data transfer time.

For other than random processing, choose the data control interval size that yields the smallest index control interval size. When a conflict exists, it is better to increase the data control interval size rather than the index control interval size.

SOME ADDITIONAL CONTROL INTERVAL CONSIDERATIONS

Pick the smallest index control interval size you can for a given data control interval size. If a 512-byte index control interval is too small, increase the data control interval size. If the 512-byte index control interval is still too small with a 4096-byte data control interval, try a 1024-byte index control interval.

Do not choose data control interval sizes that result in multiple, small physical blocks.

Specify control interval size at the data and index levels, not at the cluster level.

For variable-length records, a small data control interval results in poor DASD space utilization with more control information than fixed-length records and free space that cannot be used. If you select too small a data control interval size, the number of data control intervals in a control area may be large enough to cause the index control interval size to exceed the maximum, thus causing the DEFINE to fail.

You need real storage to support large control intervals. In an overcommitted system, excessive paging may result. The control interval sizes you specify when the data set is defined are not necessarily the ones you will have in the catalog. VSAM makes adjustments so that control interval size conforms to proper size limits, minimum buffer space, adequate index-to-data size, and record size. This is done when your data set is defined.

For example:

1. You specify data and index control interval size. After VSAM determines the number of control intervals in a control area, it estimates whether one index record is large enough to handle all control intervals in the control area. If not, the size of the index control interval is increased, if possible. If the size cannot be increased VSAM decreases the number of control intervals in the control area.

2. Assume no spanned records. You specify maximum record size as 2560 and data control interval size as 2560. VSAM adjusts the data control interval size to 3072 to allow space for control information in the data control interval.

3. You specify buffer space as 4K, index control interval size as 512, and data control interval size as 2K. VSAM decreases the data control interval to 1536. Buffer space must include space for two data control intervals and one index control interval at DEFINE time.

CONTROL AREA SIZE

A control area is never larger than one cylinder. If the original space allocation is in cylinders, then the control area size is one cylinder; if space allocations are in tracks or records, then the control area size is equal to the lesser value of the primary or the secondary allocation. The size of the control area depends on the device type. For a key-sequenced data set, the size of a control area is also determined on the basis of the space allocation request, user-specified or default data and index control interval size, and available buffer space.

Control area size has significant performance implications. When a whole number of control areas occupies a cylinder, performance is better than when a fractional number of control area occupies a cylinder (for example, when a control area is two-thirds of a cylinder). If you allocate space in a DEFINE command using the CYLINDERS parameter, VSAM sets the control area size to one cylinder. If the control area is smaller than a cylinder, its size will be an integral multiple of tracks, and it can span cylinders. However, a control area can never span an extent of a data set; that is, an extent of a data set is made up of a whole number of control areas.

Other than specifying space in terms of cylinders, you don't have a direct way of specifying that a whole number of control areas will occupy a cylinder. But you can provide values in the DEFINE command that will influence the control area size as computed by VSAM.

VSAM checks the smaller of the primary and secondary space values against the specified device's cylinder size. If the smaller space quantity is less than or equal to the device's cylinder size, the size of the control area is set equal to the smaller space quantity. If the smaller quantity is greater than the device's cylinder size, the control area size is set equal to cylinder size.

You specify space in number of tracks, cylinders, or records; the system preformats space in control areas. By calculating the size of a control area as it does, VSAM is able to meet the primary and secondary space requirements without overcommitting space for this data set.

An index record must be large enough to address all the control intervals in a control area. The more control intervals an index record addresses, the fewer reads for index records are required for sequential access. Generally, the greater the size of the control area, the better the performance and space utilization for sequential processing.

IMPACT OF SMALL CONTROL AREAS

Control areas may be from one track to one cylinder in size. The smaller the control area, of course, the more areas there will be. Because an index record can contain only so many entries, more index records and index levels are required if the control area is small.

The IMBED option requires one track per control area for sequence set information. If the control area is three tracks of 3340, and the IMBED option is taken, 1/3rd of the direct access storage space is required for sequence sets. If the control area on a 3340 is a cylinder, 1/12th the DASD space is required.

I/O BUFFER SPACE MANAGEMENT

I/O buffer space is important because VSAM transmits the contents of a control interval to a buffer in virtual storage. For keyed access with the ACB operand STRNO=1, VSAM requires a minimum of three buffers, two for data control intervals and one for an index control interval. You may specify a minimum buffer space in the DEFINE command; if you do not specify a minimum buffer space, the default is enough buffer space for the minimum of three buffers.

VSAM keeps in virtual storage as many index set records as the buffer space will allow. Ideally, the index would be small enough to allow the entire index set to remain in virtual storage. Because the characteristics of the data set may not allow a small index, you should be aware of how index I/O buffers are used to enable you to determine the number you want to provide.

The one buffer minimum assumes that requests requiring concurrent data set positioning are not being issued. If such requests are issued, each requires exclusive control of an index I/O buffer. Therefore, the value specified for the STRNO operand (ACB or GENCB macro or AMP parameter) is the minimum number of index I/O buffers required when requests that require concurrent positioning are used.

If the number of I/O buffers provided for index records is greater than the number of requests that require concurrent positioning, one buffer is used for the highest-level index record. Any additional buffers are used, as required, for other index set index records.

To improve performance when you have adequate real storage available, you can increase the I/O buffer space for index records in virtual storage by specifying I/O buffers for index records through the BUFNI and BUFSP operands of the ACB macro. With direct access, you should provide at least enough index buffers to be equal to the value of the STRNO operand of the ACB plus one to allow VSAM to keep the highest-level index record always resident. With sequential access, having only one index I/O buffer doesn't hinder performance, because VSAM uses the horizontal pointer in a sequence-set record, not vertical sequence sets in the index set, to get to the next control interval.

BUFFER SPACE

BUFFERSPACE, specified in the access method services DEFINE command, is the minimum amount of virtual storage that will ever be provided for I/O buffers when the data set is being processed. BUFSP, specified in the ACB or GENCB macro or in the DD AMP parameter, is the maximum amount of virtual storage to be used for the data set's I/O buffers. It is important that VSAM must always have sufficient space available to process the data set as the specified processing options direct it to.

Buffer Allocation for a Path

A path typically consists of a base cluster, an alternate index for the base cluster, and the alternate indexes that are included in the upgrade set. The base cluster can be key sequenced or entry sequenced. The upgrade set identifies each alternate index that VSAM is to update when the base cluster is updated (there may be no alternate indexes in the upgrade set). The alternate index provides the user with an alternate key sequence to access records in the base cluster.

The BUFSP, BUFND, BUFNI, and STRNO parameters apply only to the path's alternate index when the base cluster is opened for processing with its alternate index. The minimum number of buffers are allocated to the base cluster unless the cluster's BUFFERSPACE value (in the cluster's catalog record) or ACBBSTNO allows for more buffers (VSAM assumes direct processing and extra buffers are allocated between data and index components accordingly).

Two data buffers and one index buffer are always allocated to each alternate index in the upgrade set. If the path's alternate index is a member of the upgrade set, the minimum buffer increase for each allocation is by 1 for both the data and index buffers. Buffers are allocated to the alternate index as though it were a key-sequenced data set.

Things You Should Know about Buffer Allocation

When processing a VSAM data set sequentially (SEQ or SKP):

- For mixed processing situations (SEQ and DIR), start with two data buffers per string and increase BUFND to three per string if paging is not a problem. For straight sequential processing environments, start with four data buffers per string.
- Extra index buffers have little effect during sequential processing, because VSAM usually searches the sequence set and does not refer to the higher levels of the index.
- Large data control intervals or small data control intervals with many buffers can produce similar results. With proper buffering, the same amount of data can be accessed with one I/O operation.
- Allocate more data buffers, because the data buffers are used to support the read-ahead function. When SHAREOPTIONS 4 is specified for the data set, the read-ahead function can be ineffective because the buffers are refreshed when each control interval is read. Therefore, for SHAREOPTIONS 4, keeping data buffers at a minimum can actually improve performance.
- If your operation is I/O bound, you should specify more data buffers to improve your job's run time. However, an excessive number of buffers can cause performance problems; see note below.

When processing a VSAM data set directly (DIR):

- The read-ahead function is inactive for direct processing. The minimum number of data buffers are needed.
- For optimum operation, specify the number of index buffers equal to the number of high-level index set control intervals plus one per string to contain the entire high-level index set and one sequence set control interval per string in virtual storage. Note that additional index buffers will not be used for more than one sequence set buffer per string unless shared resource pools are used. For large data sets, specify the number of index buffers equal to the number of index levels. Unused index buffers do not degrade performance.
- If you specify more data buffers than the minimum requirement, this has little beneficial effect with direct processing.

Note: More buffers (either data or index) than necessary might cause excessive paging or excessive internal processing. There is an optimum point at which more buffers will not help. What you should attempt is to have data available just before it is to be used. If data is read into buffers too far ahead of its use in the program, it may be paged out.

Data and index buffers are acquired and allocated only when the data set is opened. Buffer space is released when the data set is closed.

VSAM dynamically allocates buffers based on parameters in effect when the program opens the data set. Parameters that influence the buffer allocation are in the program's ACB: MACRF=(IN|OUT, SEQ|SKP, DIR), STRNO=n, BUFSP=n, BUFND=n, and BUFNI=n. Other parameters that influence buffer allocation are in the DD statement's AMP specification for BUFSP, BUFND, and BUFNI, and the BUFFERSPACE value in the data set's catalog record.

If you open a data set whose ACB includes MACRF=(SEQ,DIR), buffers are allocated according to the rules for sequential processing. If the RPL is modified later in the program, the buffers allocated when the data set was opened do not change.

Data and index buffer allocation (BUFND and BUFNI) can only be specified by the user with access to modify the ACB parameters or via the AMP parameter of the DD statement.

Any program can be assigned additional buffer space by modifying the data set's BUFFERSPACE value, or by specifying a larger BUFSP value with the AMP parameter in the data set's DD statement.

When processing the data set sequentially, VSAM reads ahead and provides overlap as buffers become available. For output processing (PUT-add or PUT-update), VSAM does not immediately write the updated control interval from the buffer unless a control interval split is required. The POINT macro does not cause read-ahead processing, unless SEQ is specified, because its purpose is to position the data set for subsequent sequential retrieval.

When processing a data set directly, VSAM reads only one data control interval at a time. For output processing (PUT-add or PUT-update), VSAM immediately writes the updated control interval.

When a buffer's contents are written, the buffer's space is not released. The control interval remains in storage until overwritten with a new control interval, so that if your program refers to that control interval VSAM does not have to reread it. Because VSAM checks to see if the desired control interval is in storage, when your program processes records in a limited key range throughput might be increased if extra data buffers are provided.

VSAM does not read-ahead index buffers, but you can have your entire index set in storage. Index buffers are loaded when the index is referred to. When many index buffers are provided, index buffers are not reused until a requested index control interval is not in storage. If you provide, in addition to one index buffer per string, as many index buffers as there are index set control intervals, the data set's entire index set will be read into storage as needed.

UNITS OF ALLOCATION

The parameters you specify that determine how VSAM allocates space are in the DEFINE command. Allocation may be specified at many levels: cluster and/or alternate index, data, or data and index levels.

MULTIPLE CYLINDER DATA SETS

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. Make the secondary allocation equal to or greater than one cylinder but less than the primary allocation. If the IMBED option is used, when doing the calculation, deduct the one track per cylinder used for the replicated embedded sequence set records. For example, using a 3340, calculate based on 11 tracks per cylinder rather than 12. An allocation of 3 primary and 1 secondary track for the index set is a good choice when the REPLICATE option is used. When the REPLICATE option is not used, specify 1 primary and 1 secondary track for the index set.

SMALL DATA SETS

VSAM uses track allocation when you define a data set if you specify either track allocation or record allocation requiring less than one cylinder. For data sets less than 1 cylinder in size, it is more advantageous to specify the maximum number of tracks required in the primary allocation of the data component, 1 track for the non embedded sequence-set index, and no secondary for either data or index. The buffer allocations for this data set should be set so that only 1 index buffer is allocated.

CHOOSING ALLOCATION PARAMETERS

The following list suggests some items you should consider when allocation parameters are specified:

- A control area is never larger than one cylinder. Improved performance is obtained when an integral number of control areas occupy a cylinder.

- A control area can never span an extent boundary. A cluster extent consists of a whole number of control areas.

- VSAM checks the smaller of primary and secondary space values against the specified device's cylinder size. If the smaller quantity is greater than the device's cylinder size, the control area is set equal to the cylinder size. If the smaller quantity is less than or equal to the device's cylinder size, the size of the control area is set equal to the smaller space quantity.

  For example:

CYLINDERS(5,10)	Results in a 1-cylinder control area
TRACKS(100,3)	Results in a 3-track control area
RECORDS(2000,5)	Assuming 10 records would fit on a track, results in a 1-track control area (minimum control area is 1 track)
TRACK(3,100)	Results in a 3-track control area

  To cause VSAM to select cylinder control areas:

  - Define the data set using the CYLINDERS parameter or

  - Define the data set using the RECORDS or TRACKS parameter, with the smaller of primary or secondary allocation resulting in at least one allocated cylinder. Note that migration to a different device type may result in a case of less than a cylinder, unless the allocation parameter is adjusted accordingly.

  Note: This causes VSAM to start at a cylinder control area size. If, because of other parameters (for example, control interval size) a cylinder control area size is not optimum, VSAM determines the optimum area size in tracks and changes all space allocation parameters to be tracks (multiples of the control area size).

- If allocation is specified at the cluster or alternate index level only, the amount needed for the index is subtracted from the specified amount. The remainder of the specified amount is assigned to data.

- If allocation is specified at the data level only, the specified amount is assigned to data. The amount needed for the index is in addition to the specified amount.

- If allocation is specified at both the data and index levels, the specified data amount is assigned to data and the specified index amount is assigned to the index.

- If secondary allocation is specified at the data level, secondary allocation must be specified at the index level (when it is not specified at the cluster level).

- If IMBED is specified (to place the sequence set with the data), the data allocation includes the sequence set. More space must be given for data allocation when IMBED is specified.

- If secondary allocation is specified, space for a data set can be expanded to a maximum of 123 extents (provided there is sufficient data space). For a key-sequenced data set, the index component as well as the data component can have up to 123 extents. When the sequence set is embedded with the data, each data extent is also considered an index extent: The number of extents for the index component equals the number of data extents plus the number of high-level index set extents.

- A spanned record cannot be longer than a control area less the control information (10 bytes per control interval), so do not specify large spanned records and small primary or secondary allocation that is not large enough to contain the largest spanned record.

- VSAM acquires space in increments of control areas. For example, if the allocation amount is 20 tracks and the device is a 3330, the control area size is 1 cylinder, and 2 cylinders of space (2 control areas) are allocated.

- VSAM data sets cataloged in an ICF catalog are allocated with the CONTIG attribute if the allocation unit is TRACKS. Therefore, the primary and secondary allocation will be in contiguous tracks.

DISTRIBUTED FREE SPACE

You can specify in the DEFINE command the percentage of free space in a control interval and the percentage of free control intervals in a control area. This free space improves performance by reducing the likelihood of control interval and control area splits, which, in turn, reduce the likelihood of VSAM moving a set of records to a different cylinder away from other records in key sequence.

The amount of free space to be provided depends on the number and location of records to be inserted, lengthened, or deleted. Too much free space may increase the number of index levels, which affects run times for direct processing. It also uses more direct access storage to contain the data set, and it requires more I/O operations to sequentially process the same number of records. Too little free space may result in an excessive number of control interval and control area splits, which are time consuming at the time of the split. After the splits occur, additional time is required for sequential processing because the data set is not physically in sequence. Control area splits increase the seek time during processing. Consider using LISTCAT or the ACB JRNAD exit to monitor control area splits and reorganize the data set when they become prevalent.

VSAM uses available free space when there is a direct insert and when a mass sequential insert does not result in a split.

Control interval free space should be consistent with the expected insertion activity. Determine the free space based on the percentage of additions between reorganizations. If there are to be no additions and if record sizes are not changed, there is no need for free space.

Your free space specification can be altered after the data set is loaded. To take full advantage of mass insertion, use the ALTER command to change free space to (0,0) after the data set is loaded.

If additions will occur only in a specific part of the data set, load those parts where additions will not occur with a free space of (0,0). Then, alter the specification to (n,n) and load those parts of the data set that will receive additions. Remember that, if SPEED is specified, it will be in effect for loading the initial portion only. When subsequent portions are loaded, RECOVERY will be in effect, regardless of the DEFINE specification.

If additions will be unevenly distributed throughout the data set, specify a small amount of free space. Additional splits, after the first, in that part of the data set with the most growth will produce control intervals with only a small amount of unneeded free space.

If there will be few additions to the data set, consider a free space specification of (0,0). When records are added, new control areas will be created to provide room for additional insertions and unused free space will not be provided.

Records are loaded or mass inserted at the end of a control interval until the free space threshold would be passed. The threshold is the point at which free space would be less than the amount specified in the catalog.

VSAM ensures that at least one record or a portion of one spanned record is placed in a control interval. A control area free space percentage specification resulting in less than one free control interval in the control area, is ignored.

Because a control interval contains logical records, free space, and control information, a 4K-byte control interval cannot contain four 1K-byte logical records. A 4K-byte control interval with (25,0) free space specified contains at least 1K-byte free space. Only two 1K-byte fixed-length records could be loaded in the control interval, and only one more 1K-byte record could be added before a control interval split would be required.

If a control interval can contain four logical records and (25,0) free space is specified, the control interval would contain three logical records and 25% free space. If (20,0) is specified, the result is three logical records and 25% free space. If (33,0) is specified, the result is two logical records and 50% free space. If (80,0) is specified, the result is one logical record and 75% free space.

FREE SPACE COMPUTATION

Determine the growth of the data set between creation and reorganization. Apportion this amount of growth between free control intervals in a control area and free space within a control interval. Make sure that the computations yield full records and full control intervals with a minimum amount of unusable space.

INDEX OPTIONS

Five options influence performance through the use of the index of a key-sequenced data set. Each option improves performance, but some of them require that you provide additional virtual storage or auxiliary storage space. The options are:

- Index-set records in virtual storage
- Size of index control interval

- Index and data set on separate volumes
- Replication of index records (REPL option)
- Sequence-set records adjacent to control areas (IMBED option)

INDEX-SET RECORDS IN VIRTUAL STORAGE

To retrieve a record from a key-sequenced data set or store a record in it using keyed access, VSAM needs to examine the index of that data set. Before your processing program begins to process the data set, it must specify the amount of virtual storage it is providing for VSAM to buffer index records. Enough space for one I/O buffer for index records is the minimum, but a serious performance problem would occur if an index record were continually deleted from virtual storage to make room for another and then retrieved again later when it is required. Ample space to buffer index records can improve performance by preventing this situation.

You ensure that index-set records will be in virtual storage by specifying enough virtual storage for index I/O buffers when you begin to process a key-sequenced data set. VSAM keeps as many index-set records in virtual storage as the space will hold. Whenever an index record must be retrieved to locate a data record, VSAM makes room for it by deleting from the space the index record that VSAM judges to be least useful under the circumstances then prevailing. It is generally the index record that belongs to the lowest index level or that has been used the least. VSAM does not keep more than one sequence set index record per string at a time unless shared resource pools are used.

SIZE OF THE INDEX CONTROL INTERVAL

The second option you might consider is ensuring that the index-set control interval is large enough to cover a full control area. Thus, the index-set control intervals might be larger than actually required to contain the pointers to the sequence-set level. However, this option also keeps to a minimum the number of index levels required, thereby reducing search time and improving performance. This option increases rotational delay and transfer time.

INDEX AND DATA ON SEPARATE VOLUMES

When a key-sequenced data set is defined, the entire index or the high-level index set alone can be placed on a volume separate from the data, either on the same or on a different type of device.

Using different volumes enables VSAM to gain access to an index and to data at the same time. Additionally, the smaller amount of space required for an index makes it economical to use a faster storage device for it than for the data.

REPLICATION OF INDEX RECORDS

You can specify that each index record be replicated (written on a track of a direct access volume as many times as it will fit). Replication reduces the time lost waiting for the index record to come around to be read (rotational delay). Average rotational delay is half the time it takes for the volume to complete one revolution. Replication of a record reduces this time, for example, if 10 copies of an index record fit on a track, average rotational delay is only 1/20th of the time it takes for the volume to complete one revolution.

On a 3340, the time usually is reduced by 50%. On a 3330, the time is reduced to 1/n, where n is the number of times the index is replicated on the track.

Because there are usually few control intervals in the index set, the cost in terms of direct access storage space is small. If the entire index set is not being held in storage and there is significant random processing, then replication is a good choice. If not, replication does very little. Because its cost is small and it is an attribute that cannot be altered, it may be desirable to choose this option.

SEQUENCE-SET RECORDS ADJACENT TO CONTROL AREAS

When the data set is defined, you can specify that the sequence-set index record for each control area is to be embedded on the first track of the control area. This reduces disk-arm movement, because it is not necessary to do separate seeks to locate both the sequence-set index record and the data record. One arm movement enables VSAM to retrieve or store both the index record and the contents of the control interval in which the data record is stored.

When the IMBED option is chosen, sequence-set records are replicated, regardless of whether you also chose the REPL option. This means that one track of each control area is used for sequence set records. In some situations, this may be too much space for index in relation to the data. For example, the space required for the sequence-set is 1/12th of the data space on a 3340, but only 1/19th of the data space on a 3330. IMBED must be specified explicitly to get the performance benefits of a replicated, embedded sequence-set.

CHAPTER 4. SECURITY

The protection of data includes:

- Data security, or the safety of data from theft or intentional destruction
- Data integrity, or the safety of data from accidental loss or destruction

The following sections describe the data protection available:

- Authorized program facility (APF)
- Access method services options
- Resource Access Control Facility (RACF)
- User-security-verification routine (USVR)
- Access method services cryptographic option

AUTHORIZED PROGRAM FACILITY (APF)

The authorized program facility (APF) limits the use of sensitive system services and resources to authorized system and user programs. See "Authorized Program Facility (APF)" in System Programming Library: Supervisor Services and Macros for information about program authorization.

All access method services load modules are contained in SYS1.LINKLIB, and the root segment load module (IDCAMS) is link-edited with the SETCODE AC(1) attribute. These two characteristics ensure that access method services executes with APF authorization.

APF authorization is established at the job step level. If, during the execution of an APF-authorized job step, a load request is satisfied from an unauthorized library, the task is abnormally terminated. It is the installation's responsibility to ensure that a load request cannot be satisfied from an unauthorized library during access method services processing.

The following situations could cause the invalidation of APF authorization for access method services:

- An access method services module is loaded from an unauthorized library.
- A user-security-verification routine (USVR) is loaded from an unauthorized library during access method services processing.
- An exception exit routine is loaded from an unauthorized library during access method services processing.
- A user-supplied special graphics table is loaded from an unauthorized library during access method services processing.

Because APF authorization is established at the job step task level, access method services is not authorized if invoked by an unauthorized problem program or an unauthorized terminal monitor program (TMP).

Under TSO, if the system does not have the TSO Command Package Program Product, you can authorize your TMP by relink-editing it with the SETCODE AC(1) attribute. You must enter the names of those access method services commands requiring APF authorization to execute under TSO in the authorized command list.

The restricted functions performed by access method services that cannot be requested in an unauthorized state are:

CNVTCAT	when converting to an ICF catalog
DEFINE	when the RECATALOG parameter is specified
DELETE	when the RECOVERY parameter is specified
EXPORT	when the object to be exported is an ICF catalog
IMPORT	when the object to be imported is an ICF catalog
PRINT	when the object to be printed is a catalog
REPRO	when copying an ICF catalog, or the ICF catalog unload/reload is to be used
VERIFY	when a catalog is to be verified

If the above functions are required and access method services is invoked from a problem program or a TSO terminal monitor program, the invoking program must be authorized.

ACCESS METHOD SERVICES OPTIONS

Access method services provides options to protect data sets against unauthorized use and loss of data. To effectively use the protection features, you must understand the difference between the operations:

- Referring to a catalog entry when new entries are defined (DEFINE), or existing entries are altered (ALTER), deleted (DELETE), or listed (LISTCAT)
- Using the data set represented by a catalog entry when it is connected to a user's program (OPEN), or disconnected (CLOSE), or when it reaches its upper RBA boundary (End-of-Volume)

Different passwords may be needed for each type of operation. The data set and catalog security options are described in the sections that follow.

VSAM PASSWORD PROTECTION

PASSWORDS TO AUTHORIZE ACCESS: You may, optionally, define passwords for clusters, cluster components (data and index), page spaces, alternate indexes, alternate index components (data and index), paths, master and user catalogs, for access to them. Different passwords have various degrees of security, with higher levels providing greater protection than lower levels. The levels are:

- Full access. This is the master password, which allows you to perform all operations (retrieving, updating, inserting, and deleting) on an entire VSAM data set and any index and catalog record associated with it. The master password allows all operations and bypasses any additional verification checking by the user-security-verification routine.
- Control access. This password authorizes you to use control interval access. For further information, see "Chapter 6. Options for Advanced Applications" on page 81.
- Update access. This password authorizes you to retrieve, update, insert, or delete records in a data set. The update

password does not allow you to alter passwords or other security information.

- Read access. The read-only password allows you to examine data records and catalog records, but not to add, alter, or delete them, nor to see password information in a catalog record.

Operations on a catalog may be authorized by the catalog's password or, in some cases, by the password of the data set whose definition in the catalog is being operated on. For example:

- Defining a data set in a password-protected catalog requires the catalog's update (or higher) password.

- Listing or deleting a definition requires the appropriate password of either the catalog or the data set.

- However, if the catalog, but not the data set, is protected, no password to list the data set's catalog definition is needed.

To delete a user catalog, you must give the master password, whether the master catalog is password protected or not.

Each higher-level password allows all operations permitted by lower levels. Any level may be null (not specified), but if a low-level password is specified, the master level password must also exist. The DEFINE and ALTER commands give the higher passwords the value of the highest password specified. A read-level password becomes the update-, control-, and master-level password as well. If you specify a read password and a control password, the control password value will become the master-level password as well. However, the update-level password is null.

Some access method services operations may involve more than one password authorization. For example, importing a data set involves defining the data set and loading records into it. If the catalog into which the data set is being imported is password protected, its update-level (or higher-level) password is required for the definition; if the data set is password protected, its update-level (or higher-level) password is required for the load. The IMPORT command allows you to specify the password of the catalog; the password, if any, of the data set being imported is obtained by the commands from the exported data.

Every VSAM data set is represented in an ICF catalog by two or more components: a cluster component and a data component, or, if the data set is a key-sequenced data set, a cluster component, a data component, and an index component. Of the two or three components, the cluster component is the controlling component. Each of the two or three components can have its own set of four passwords; the passwords you assign have no relationship to each other. For example, password-protecting a cluster but not the cluster's data component, allows someone to issue LISTCAT to determine the name of your cluster's data component, open the data component, and access records in it, even though the cluster itself is password protected.

One reason for password-protecting the components of a cluster is to prevent access to the index of a key-sequenced data set, because the only way to gain access to an index is to open it independently of the cluster. (See "Chapter 6. Options for Advanced Applications" on page 81 for a description of access to an index.)

Catalogs are themselves VSAM data sets, and may have passwords. For some operations (for example, listing all the catalog's entries with their passwords or deleting catalog entries), the catalog's passwords may be used instead of the entry's passwords. If the master catalog is protected, the update- or higher-level password is required when defining a user catalog, because all user catalogs have an entry in the master catalog. When deleting a protected user catalog, the user catalog's master password must be specified.

If the ATTEMPTS parameter is coded with 0, no password prompting is done.

PROTECTION CONSIDERATIONS AND PRECAUTIONS

FOR THE CATALOG: Observe the following precautions when using protection commands for the catalog:

- Passwords are 1 to 8 alphameric or special characters (see "How to Code Subparameters" in Access Method Services Reference for more information).

- To create a catalog entry (with the DEFINE command), the update- or higher-level password of the catalog is required.

- To modify a catalog entry (with the ALTER command), the master password of the entry or the master password of the catalog which contains the entry is required. However, if the entry to be modified is a non-VSAM or GDG entry, the update-level password of the catalog is sufficient.

- To gain access to passwords in a catalog (for example, to list or change passwords), specify the master-level password of either the entry or the catalog. A master-level password must be specified with the DEFINE command to model an entry's passwords.

- To delete a protected data set entry from a catalog requires the master-level password of the entry or the master-level password of the catalog containing the entry. However, if the entry in a VSAM catalog describes a VSAM data space, the update-level password of the catalog is sufficient.

- To delete a non-VSAM, GDG, or alias entry, the update level password of the catalog is sufficient.

- To list catalog entries with the read-level passwords, specify the read password of the entry or the catalog's read level password. However, entries without passwords may be listed without specifying the catalog's read-level password. To list the passwords associated with a catalog entry, specify the master password of the entry or the catalog's master password.

  To avoid unnecessary prompts, specify the catalog's password, which allows access to all entries that the operation affects. A catalog's master-level password allows you to refer to all catalog entries. However, a protected cluster cannot be processed with the catalog's master password.

- Specification of a password where none is required is always ignored.

FOR A DATA SET: Observe the following precautions when using protection commands that cause access to a data set:

- Passwords are 1 to 8 alphameric or special characters (see "How to Code Subparameters" in Access Method Services Reference).

- To access a VSAM data set using its cluster name, instead of data or index names, you must specify the proper level password for the cluster even if the data or index passwords are null.

- To access a VSAM data set using its data or index name, instead of its cluster name, you must specify the proper data or index password. However, if cluster passwords are defined, the master password of the cluster may be specified instead of the data or index password.

- If a cluster has only null (not specified) passwords, you may access the data set using the cluster name without specifying passwords. This is true even if the data and index entries of the cluster have passwords defined. This allows unrestricted access to the VSAM data set as a whole but protects against unauthorized modification of the data or index as separate components.

RELATION OF DATA SET AND CATALOG PROTECTION: If you define passwords for any data sets in a catalog, you must also protect the catalog by defining passwords for the catalog or by defining the catalog to RACF. If you do not protect the catalog, no password checking takes place during operations on the data set's catalog entries or during open processing of data sets cataloged in that catalog.

The VVDS requires the master password of the master catalog whenever a delete recover or a GENDSP function is requested for a VVDS.

PASSWORD PROMPTING: Computer operators and TSO terminal users may supply a correct password if a processing program does not give the correct one when it tries to open a password-protected data set. When the data set is defined, you may specify a code instead of the data set name to prompt the operator or terminal user for a password. The prompting code keeps your data secure by not allowing the operator or terminal user to know both the name of the data set and its password.

A data set's code is used for prompting for any operation against a password-protected data set. The catalog code is used for prompting when the catalog is opened as a data set, when an attempt is made to locate catalog entries that describe the catalog, and when an entry is to be defined in the catalog.

If you do not specify a prompting code, VSAM identifies the job for which a password is needed with the JOBNAME and DSNAME for background jobs or with the DSNAME alone for foreground (TSO) jobs.

When you define a data set, you may specify the number of times the computer operator or terminal user is allowed to give the password when a processing program is trying to open a data set. If the allowed number of attempts is exceeded and you are using System Management Facilities, a record is written to the SMF data set to indicate a security violation.

Note: When logged onto TSO, VSAM tries the logon password before prompting at the user terminal. Using the TSO logon password counts as one attempt.

PASSWORDS FOR NON-VSAM DATA SETS

When you define a non-VSAM data set in an ICF catalog, the data set is not protected with passwords in its catalog entry. To password-protect a non-VSAM data set when it is created, specify LABEL=(PASSWORD|NOPWREAD) in the DD statement that describes the data set (for more details, see _JCL_). Use the PROTECT macro instruction to assign a password to the non-VSAM data set (for more details, see _Data Management Services Guide_ and _System Programming Library: Data Management_).

If the catalog is update protected, you must supply the catalog's update- or higher-level password to define, delete, or alter a non-VSAM data set. The password can be supplied as a subparameter of the command's CATALOG parameter, or as a response to the password-prompting message.

RESOURCE ACCESS CONTROL FACILITY (RACF)

The IBM Resource Access Control Facility (RACF) provides an optional software access control measure you may use in addition to or instead of passwords. Password protection and RACF protection can coexist for the same data set. When RACF protection is applied to a data set that is already password protected, password protection is bypassed and access is controlled solely through the RACF authorization mechanism. If a user-security-verification routine (USVR) exists, it is not invoked for RACF-defined data sets. For more information, see "Chapter 8. User-Written Exit Routines" on page 129.

To have password protection take effect for a data set, the catalog containing it must be either RACF protected or password protected and the data set itself not defined to RACF.

Although passwords are ignored for a RACF-indicated data set, they can still provide protection if the data set is moved to another system that does not have RACF protection.

In addition to protection of VSAM data sets by way of discrete profiles, RACF provides a generic profile checking facility. VSAM data sets which are generically protected are not RACF indicated in the catalog.

Therefore, RACF is invoked for any access to data sets cataloged in ICF catalogs whether or not the data set is RACF indicated or password protected. If the data set is not protected by either a discrete profile or a generic profile, password protection is in effect.

RACF authorization checking is generally compatible with the password authorization checking scheme. The compatibility includes the time the authorization check is made and the sources of authorization. The RACF authorization levels of alter, control, update, and read correspond to the password levels of master, control, update, and read in VSAM.

Deleting any type of RACF-protected entry from a RACF-protected catalog requires alter-level authorization to the catalog or the entry being deleted.

Altering the passwords in a RACF-protected catalog entry requires RACF alter authority to the entry being altered, or the operations attribute. Alter authority to the catalog itself is not sufficient for this operation.

For more information, see Resource Access Control Facility (RACF): General Information Manual.

USER-SECURITY-VERIFICATION ROUTINE (USVR)

In addition to password protection, VSAM allows you to protect data by specifying a program to verify a user's authorization. Specific requirements of the user-security-verification routine are described under "Chapter 8. User-Written Exit Routines" on page 129. To use this routine, specify the name of the authorization routine you have written in the AUTHORIZATION parameter of the DEFINE or ALTER command.

If a password exists for the type of operation being performed, the password must be given, either in the command or in response to prompting. The user-security-verification routine is called only after the password specified is verified; it is bypassed whenever a correct master password is specified, whether or not the master password is required for the requested operation.

ACCESS METHOD SERVICES CRYPTOGRAPHIC OPTION

You can provide security for your online data by using such facilities as VSAM password protection and the IBM Resource Access Control Facility (RACF) program product. These facilities, however, do not protect data when it is stored offline. Sensitive data stored offline is susceptible to misuse.

It is generally recognized that data cryptography is an effective means of protecting data, if the enciphering techniques are adequate. The function of protecting data stored offline in an enciphered form is available by using the ENCIPHER option of the access method services REPRO command. The REPRO command uses the services of the IBM Programmed Cryptographic Facility (5740-XY5) or the Cryptographic Unit Support (5740-XY6). The Programmed Cryptographic Facility conforms to the Data Encryption Standard (DES) of the United States National Bureau of Standards for enciphering data. The data remains protected until the REPRO DECIPHER option is used to decipher it with the correct key.

Note: The format and examples of the REPRO command are in Access Method Services Reference.

There are three types of offline environments in which the enciphering of sensitive data adds to its security:

- Data sets that are transported to another installation, where data security is required during transportation and while the data is stored at the other location
- Data sets that will be stored for long periods of time at a permanent storage location
- Data sets that are stored offline at the site at which they are normally used

You can use REPRO to copy a plaintext (not enciphered) data set to another data set in enciphered form. Enciphering converts data to an unintelligible form called a ciphertext. The enciphered data set can then be stored offline or sent to a remote location. When desired, the enciphered data set can be brought back online and you can use REPRO to recover the plaintext from the ciphertext by copying the enciphered data set to another data set in plaintext (deciphered) form.

Enciphering and deciphering are based on an 8-byte binary value called the key. Using the REPRO DECIPHER option, the data can be either deciphered on the system it was enciphered on, or deciphered on another system that has this functional capability and the required key to decipher the data. Given the same key, encipher and decipher are inverse operations. This option uses the services of the Programmed Cryptographic Facility or the Cryptographic Unit Support to encipher/decipher the data, and uses block chaining with ciphertext feedback during the encipher/decipher operation.

With the exception of catalogs, all data sets supported for copying by REPRO are supported as input (SAM, ISAM, VSAM) for enciphering and as output (SAM, VSAM) for deciphering.
The input data set for the decipher operation must be an enciphered copy of a data set produced by REPRO. The output data set for the encipher operation can only be a VSAM entry-sequenced or sequential (SAM) data set. The target (output) data set of both an encipher and a decipher operation must be empty. If the target data set is a VSAM data set that has been defined with the reusable attribute, you can use the REUSE parameter of REPRO to reset it to an empty status.

You use the REPRO ENCIPHER parameter to indicate that REPRO is to produce an enciphered copy of the data set, and to supply information needed for the encipherment. The INFILE or INDATASET parameter identifies and allocates the plaintext (not enciphered) source data set. The OUTFILE or OUTDATASET parameter identifies and allocates a target data set to contain the enciphered data.

You use the REPRO DECIPHER parameter to indicate that REPRO is to produce a deciphered copy of the data set, and to supply information needed for deciphering. The INFILE or INDATASET parameter identifies and allocates the enciphered source data set. The OUTFILE or OUTDATASET parameter identifies and allocates a target data set to contain the plaintext data.

See Figure 13 on page 66 for a graphic representation of the input and output data sets involved in REPRO ENCIPHER/DECIPHER operations.

You should not build an alternate index over a VSAM entry-sequenced data set that is the output of a REPRO ENCIPHER operation.

When a VSAM relative record data set is enciphered, the record size of the output data set must be at least 4 bytes greater than the record size of the relative record data set. (The extra 4 bytes are needed to prefix a relative record number to the output record.) You can specify the record size of an output VSAM entry-sequenced data set through the RECORDSIZE parameter of the DEFINE CLUSTER command. You can specify the record size of an output sequential (SAM) data set through the DCB LRECL parameter in the output data set's DD statement. When an enciphered VSAM relative record data set is subsequently deciphered with a relative record data set as the target, any empty slots in the original data set will be reestablished.

If you specify the REPLACE parameter of the REPRO command together with either the ENCIPHER or the DECIPHER parameter, access method services will ignore REPLACE, because the target VSAM data set must be empty and thus has no records to replace.

When you encipher a data set, you can specify any of the delimiter parameters available with the REPRO command (SKIP, COUNT, FROMADDRESS, FROMKEY, FROMNUMBER, TOADDRESS, TOKEY, TONUMBER) that are appropriate to the data set being enciphered. However, no delimiter parameter can be specified when a data set is deciphered. If DECIPHER is specified together with any REPRO delimiter parameter, your REPRO command terminates with a message.

When the REPRO command copies and enciphers a data set, it places one or more records of clear header data preceding the enciphered data records. This header data consists of information necessary for the deciphering of the enciphered data. Information a header may contain consists of:

- Number of header records
- Number of records to be ciphered as a unit
- Key verification data
- Enciphered data encrypting keys

KEY MANAGEMENT

This section is intended to give you an overview of key management used when enciphering or deciphering data via the REPRO command.

Data Encryption Keys

A "key" is defined as an 8-byte value. When you use the encipher/decipher function of the REPRO command, you may specify keys that the Programmed Cryptographic Facility or the Cryptographic Unit Support generates and manages for you (system-key management), or keys that you manually generate and privately manage (private-key management). In either case, REPRO invokes the appropriate Programmed Cryptographic Facility program product service.

For both private- and system-key management, REPRO allows you to supply an 8-byte value to be used as the plaintext data-encrypting key. If you do not supply the data encrypting key, REPRO provides an 8-byte value to be used as the plaintext data-encrypting key. The plaintext data encrypting key is used to encipher/decipher the data using the Data Encryption Standard.

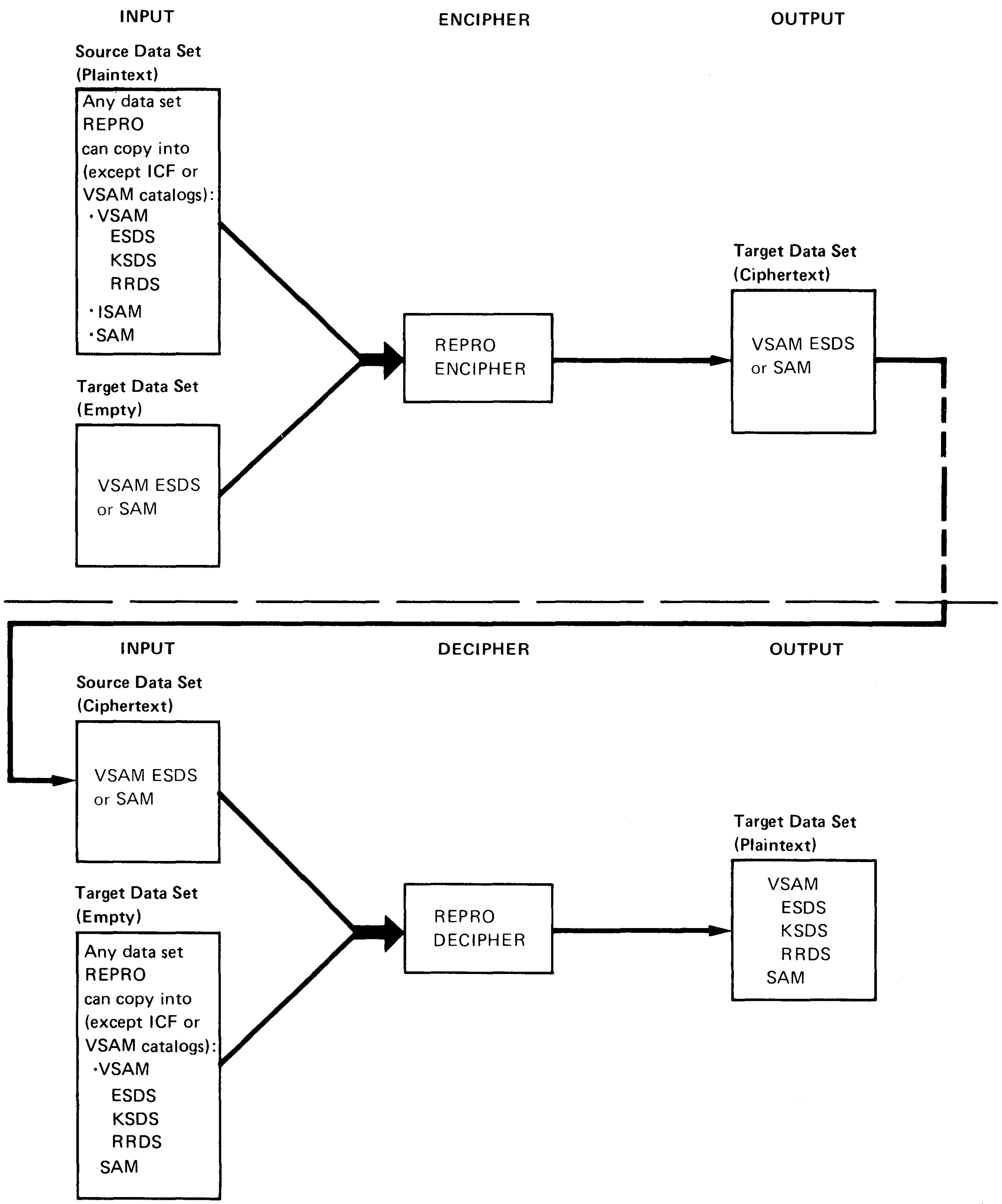

INPUT
ENCIPHER
OUTPUT
Source Data Set (Plaintext)
Any data set REPRO can copy into (except ICF or VSAM catalogs):
•VSAM
ESDS
KSDS
RRDS
•ISAM
•SAM
Target Data Set (Empty)
VSAM ESDS or SAM
REPRO ENCIPHER
Target Data Set (Ciphertext)
VSAM ESDS or SAM
INPUT
DECIPHER
OUTPUT
Source Data Set (Ciphertext)
VSAM ESDS or SAM
Target Data Set (Empty)
Any data set REPRO can copy into (except ICF or VSAM catalogs):
•VSAM
ESDS
KSDS
RRDS
SAM
REPRO DECIPHER
Target Data Set (Plaintext)
VSAM
ESDS
KSDS
RRDS
SAM

Figure 13. REPRO Encipher/Decipher Operations

If you want to supply your own plaintext data-encrypting key on ENCIPHER or DECIPHER through the REPRO command, there is a risk of exposing that key when the command is listed on SYSPRINT. To avoid this exposure, you may direct REPRO to a data-encrypting key data set to obtain the plaintext data encrypting key. You identify the DD statement for this data set through the DATAKEYFILE parameter of the REPRO command. REPRO obtains the data-encrypting key from this data set by locating the first nonblank column in the first record of the data set and processing 16 consecutive columns of data. These 16 columns contain the hexadecimal character representation for the 8-byte key. Each column of data must contain the EBCDIC character representation of a hexadecimal digit (that is, 0 through F). If a blank column is found before 16 nonblank columns have been processed, the data key is padded to the right with EBCDIC blanks. If the end of the first record is encountered before 16 columns have been processed, the data key is considered invalid. If a valid data-encrypting key cannot be obtained from the first record of the data-encrypting key data set, then REPRO either generates the data-encrypting key (ENCIPHER) or terminates with a message (DECIPHER).

If you allow (or force, by an invalid data-encrypting key data set record) REPRO to provide the data-encrypting key on ENCIPHER, there is a risk of exposing the data-encrypting key only if you have chosen to manage your keys privately, because REPRO lists the generated data-encrypting key only for privately managed keys.

Secondary File Keys

When you want to decipher the data, you must supply the data-encrypting key that was used to encipher the data. However, as a security precaution, you may want to supply the data-encrypting key in a disguised form. When enciphering the data set, the name of a system-managed key, called a secondary file key, may be supplied. REPRO uses the secondary file key indicated by the supplied name to disguise (encipher) the data-encrypting key. When deciphering the data set, the name of the file key and the disguised data-encrypting key may be supplied rather than the plaintext data-encrypting key. In this way, the actual plaintext data-encrypting key is not revealed. (Note that the secondary file key is used only in key communication, not when enciphering data.)

The Programmed Cryptographic Facility or the Cryptographic Unit Support offers the installation this secondary file key-management facility. To use this facility, your installation must first execute the Programmed Cryptographic Facility key generator utility. This utility is used to generate the secondary-file keys to be used by the installation.

The Programmed Cryptographic Facility key generator utility generates the secondary-file keys you request and stores the keys, in enciphered form, in the cryptographic key data set (CKDS). It lists the external name (key name) of each secondary key and the plaintext form of the secondary key. To access a particular secondary file key, you supply the keyname of the secondary file key. If the secondary key is to be used on a system other than the system on which the keys were generated, the utility must also be executed at the other system to define the same plaintext secondary file keys. The plaintext secondary file keys may be defined in the CKDS of the other system with different keynames.

If you choose to manage your own private keys, no secondary file keys are used to encipher the data-encrypting key; it is your responsibility to ensure the secure nature of your private data-encrypting key.

If you choose to have the Programmed Cryptographic Facility or the Cryptographic Unit Support manage your keys, REPRO uses secondary file keys to encipher the data-encrypting key. You specify a secondary file key to REPRO through the key name by which it is known in the CKDS. Two types of secondary file keys are recognized by REPRO:

- An internal file key is one that is defined in the CKDS by using a remote or cross key 2 statement as input to the Programmed Cryptographic Facility key generator utility. Use of a cross key 2 statement for DECIPHER requires APF (Authorized Program Facility) authorization and thus can provide an additional level of security.

- For ENCIPHER, an external file key is one that is defined in the CKDS by using a local or cross key 1 statement as input to the Programmed Cryptographic Facility key generator utility. An internal file key may be used for ENCIPHER but not for DECIPHER operations.

Although both internal and external secondary file keys may be used when doing a REPRO ENCIPHER, only internal secondary file keys may be used when doing a REPRO DECIPHER. Thus, external secondary file keys may not be used to do a REPRO ENCIPHER and REPRO DECIPHER on the same system. External secondary file keys are used when the enciphered data is to be deciphered on another system. The REPRO DECIPHER is done on the other system using the same secondary file key; however, in this other system's CKDS, the secondary file key must be defined as an internal secondary file key.

An internal secondary file key may be used to do a REPRO ENCIPHER and REPRO DECIPHER on the same system, and, if the same secondary file key is also defined as an internal secondary file key in another system's CKDS, then a REPRO ENCIPHER and REPRO DECIPHER may also be done on that system.

REQUIREMENTS

In planning to use the ENCIPHER or DECIPHER functions of the REPRO command, you should be aware of the following requirements:

- Either the Programmed Cryptographic Facility Program Product, Program Number 5740-XY5, or the Cryptographic Unit Support Facility Program Product, Program Number 5740-XY6, must be installed on your system.

- Prior to issuing the REPRO command (via a batch job or from a TSO terminal), the Programmed Cryptogaphic Facility program product must have already been started through a START command at the operator's console.

In planning to use REPRO DECIPHER with the SYSTEMKEYS parameter specifying a cross key 2 secondary file key, you should be aware of the following requirement:

- Access method services must be authorized. For information about program authorization, see "Authorized Program Facility (APF)" in _System Programming Library: Supervisor Services and Macros_.

CHAPTER 5. SHARING A VSAM DATA SET

A VSAM data set can be shared by different jobs in a single operating system and by different subtasks in an address space. The guidelines in this chapter discuss methods of accessing shared data sets as well as provisions for controlling data sharing to prevent the loss of data.

When you define a data set, you can select the level of sharing you intend to allow for the data set. Before you define the data set's level of sharing, evaluate the consequences of reading incorrect data (a loss of read integrity) and writing incorrect data (a loss of write integrity)—situations that may result when one or more of the data set's users do not adhere to guidelines recommended for accessing shared data sets.

When your program issues a GET request, VSAM reads an entire control interval into virtual storage (or obtains a copy of the data from a control interval already in virtual storage.) If your program modifies the control interval's data, VSAM ensures within a single control block structure, that you have exclusive control over the control interval until it is written back to the data set. If the data set is accessed by more than one program at a time, and more than one control block structure contains buffers for the data set's control intervals, VSAM can not ensure that your program has exclusive control over the data. You must obtain exclusive control yourself, using ENQ and DEQ.

The extent to which your data set can be shared is determined by:

1. The use of the DISP=SHR and DISP=OLD parameters in the DD statement that identifies the data set to be opened.

   Note: Scheduler "disposition" processing remains the same for VSAM and non-VSAM. This is the first level of shared protection.

2. The type of processing (specified with the ACB's MACRF field) for which the data set is opened.

3. The value for the SHAREOPTIONS parameter, specified when the VSAM data set is defined (using access method services DEFINE command), indicates the level at which the data set can be shared by users in the same MVS operating system (cross-region sharing) and by users in different MVS operating systems (cross-system sharing).

4. Your program's use of ENQ and DEQ to obtain exclusive control of the data set within your program's operating system.

5. The ability of the data set's direct access device to be accessed by more than one processor (and consequently, by more than one MVS operating system). If the data set's direct access device can be shared between two or more MVS operating systems, your program should use ENQ/DEQ in the UCB option to obtain exclusive control of the device.

If the VSAM data set cannot be shared, a scheduler failure will occur or a return code is set in the ACB ERROR field when the user's program issues the OPEN request.

If the data set's DD statement specifies DISP=OLD, only the dsname associated with the DD statement is exclusively controlled. Only the cluster name is reserved for the Open routine's exclusive use. You can include DD statements with DISP=OLD for each of the cluster's components to reserve them as well. Doing this ensures that all resources needed to open the data set will be exclusively reserved before your task is initiated.

Note: Protecting the cluster name with DISP processing and the components by VSAM OPEN SHAREOPTIONS is the minimum procedure.

When a shared data set is opened with DISP=OLD, or is opened for create or reset processing, your program has exclusive control of the data set within your operating system. If the data set can be shared between MVS operating systems, a user's program in another system may concurrently access the data set. Before you open the data set specifying DISP=OLD, it is your responsibility to protect across systems with ENQ/DEQ in the UCB option or equivalent functions.

When a data set defined with cross-region SHAREOPTIONS 3 or 4, and with cross-system SHAREOPTIONS 3, is opened with DISP=SHR, VSAM puts critical control block data in a common storage area and maintains the data there. The control block data in the common storage area is available to each program (each region) sharing the data set. The common storage area is available only to regions within your MVS operating system. Communicating this information to another MVS operating system is your responsibility.

Before your program opens a data set to be shared (with cross-region SHAREOPTIONS 3 or 4, and with cross-system SHAREOPTIONS 3), you should note that:

- In a shared environment, VSAM does not allow you to process the data set in an initial load or reset mode (create). VSAM forces your data set to be processed as though it were defined with SHAREOPTIONS(1 3).
- A user program cannot share a system data set (for example, the master catalog, page space data sets, SYS1. data sets, duplex data sets, and swap data sets).
- The user's program must serialize all VSAM requests against the data set, using ENQ/DEQ (or a similar function).
- The user's program must insure that all VSAM resources are acquired and released within ENQ/DEQ protocol to:
  - Force VSAM to write sequential update and insert requests.
  - Release VSAM's positioning within the data set.
- VSAM invalidates buffers used with SHAREOPTIONS 4 data sets, but does not invalidate buffers used with SHAREOPTIONS 3 data sets. When a buffer is marked invalid (it is invalidated), it is identified as a buffer that VSAM must refresh (read in a fresh copy of the control interval from DASD) before your program can use the buffer's contents.
- Programs that use GSR and LSR can invalidate and force writing of buffers using the MRKBFR and WRTBFR macros.

When the data set is shared under cross-system SHAREOPTIONS 4, regardless of cross-region requests, VSAM does not allow changes to high-used and high-key RBAs. Control area splits and the addition of a new high-key record for a new control interval that results from a control interval split are not allowed. VSAM returns a logical error control to the user's program that is used to detect this condition. Also, the data and sequence-set control interval buffers are marked invalid following I/O operation to a direct access storage device.

When your program examines the data set's statistics (in the AMDSB), it obtains information only about its own use of the data set. When all ACBs that refer to the data set are closed, VSAM updates the catalog with the combined activity of all users.

If your program shares a data set defined with SHAREOPTIONS(3 3) or SHAREOPTIONS (4 3), the following restrictions apply:

- Because programs in many regions can share the same data set, an error that occurs in one region may affect programs in other regions that share the data set.

- When a shared data set needs additional space, VSAM end-of-volume processing obtains the space and updates the data set's control block information in the common storage area. All programs (regions) sharing the data set can access the new space. If an error prevents the end-of-volume processing from completing, all regions are prevented from performing additional end-of-volume processing. To obtain additional space for the data set, you must first close that data set in all regions.

  Note: To correct the data set's catalog record, the ACBSWARN flag should be off in the ACB (the default) that is reopening the data set in order to allow implicit VERIFY processing to take place. If implicit verify is suppressed (ACBSWARN flag is on), access method services VERIFY should be issued to update the catalog record.

- Implicit VERIFY is invoked by the open-for-output indicator in the catalog. Normal CLOSE processing resets the indicator and suppresses the implicit VERIFY for the next open function.

SUBTASK SHARING

Subtask sharing should not be confused with, and is totally independent of the type of buffer management technique selected with NSR or LSR/GSR.

To successfully share within a task or between subtasks, you should assure that VSAM builds a single control block structure for the data set. This also includes blocks for control information in addition to input/output buffers. In this environment, VSAM record management will serialize updates to any single control internal and provide read and write integrity. When a control interval is not available for the type of user processing requested, VSAM record management returns a logical error feedback. When this occurs, you must decide whether to retry later or to free the resource that is causing the conflict. See Figure 14 on page 72 for a diagram on exclusive control conflict feedback and the results for different user requests. Also, see "Read Integrity during Cross-Region Sharing" on page 74 and"Write Integrity during Cross-Region Sharing" on page 75 about requests that can cause exclusive control conflicts.

The three methods of achieving a single control block structure for a VSAM data set while processing multiple concurrent requests include:

- Single access method control block (ACB) and a STRNO>1

- Mulitple ACBs pointing to a single DD statement (all with the same dsname)

- Multiple ACBs pointing to mulitple DD statements with different dsnames that are related with an ACB open specification (MACRF=DSN)

User B has

User A wants	Exclusive Control	Shared
Exclusive Control	User A gets logical error	OK User A gets second copy of buffer
Shared	OK User A gets second copy of buffer	OK User A gets second copy of buffer

Nonshared Resources (NSR)

User B has

User A wants	Exclusive Control	Shared
Exclusive Control	User A gets logical error	VSAM queues User A until buffer is released by User B[1]
Shared	User A gets logical error	OK User A shares same buffer with User B

Shared Resources (LSR/GSR)

1 Only a single request for each buffer call will be deferred at a time. Once a request is deferred, a logical error is returned for the second and succeeding requests until the first request is dequeued.

Figure 14. Exclusive Control Conflict Resolution

DATA SET NAME SHARING

Data set name sharing is established by the ACB option (MACRF=DSN). To understand DSN sharing, you must understand what a sphere and the base of the sphere are and how they function.

A sphere is a VSAM cluster and its associated data sets. The cluster is originally defined with the access method services command DEFINE CLUSTER. The most common use of the sphere is to open a single cluster. The base of the sphere is the cluster itself. When opening a path (which is the relationship between an alternate index and base cluster) the base of the sphere is again the base cluster. Opening the alternate index as a data set results in the alternate index becoming the base of the sphere. In Figure 15, DSN is specified for each ACB and output processing is specified.

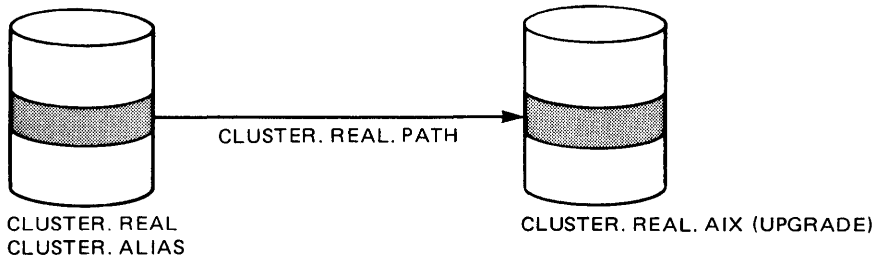


Figure 15. Relationship between the Base Cluster and the Alternate Index

Defining a Sphere

1. OPEN ACB=(CLUSTER.REAL)
   - Builds control block structure for CLUSTER.REAL
   - Builds control block structure for CLUSTER.REAL.AIX
2. OPEN ACB=(CLUSTER.REAL.PATH)
   - Adds to existing structure for CLUSTER.REAL
   - Adds to existing structure for CLUSTER.REAL.AIX
3. OPEN ACB=(CLUSTER.ALIAS)
   - Adds to existing structure for CLUSTER.REAL.
4. OPEN ACB=(CLUSTER.REAL.AIX)
   - Does not add to existing structure as the base of the sphere is not the same.
   - SHAREOPTIONS are enforced for CLUSTER.REAL.AIX since multiple control block structures exist.

When processing with a single structure across multiple subtasks, concurrent GET and PUT requests are allowed. However, a control interval is protected for write operations using an exclusive control facility provided in VSAM record management. Other PUT requests to the same control interval are not allowed and a logical error is returned to the user following the request macro. Depending on the selected buffer option, nonshared (NSR) or shared (LSR/GSR) resources, GET requests to the same control interval as that begin updated may or may not be allowed. Figure 14 on page 72 illustrates the exclusive control facility.

When a subtask issues OPEN to an ACB that will share a control block structure that may have been previously used, obtain the position for the data set issuing the POINT macro. It should not be assumed that positioning, in this case, is at the beginning of the data set.

CROSS-REGION SHARING

Independent job steps or subtasks in an MVS operating system or multiple systems using global resource serialization (GRS) can access a VSAM data set simultaneously. To share a data set, each user must specify DISP=SHR in the data set's DD statement. The level of cross-region sharing allowed by VSAM is established (when the data set is defined) with the SHAREOPTIONS value:

- Cross-region SHAREOPTIONS 1: The data set can be shared by any number of users for read processing, or the data set can be accessed by only one user for read and write processing. With this option, VSAM ensures complete data integrity for the data set.
- Cross-region SHAREOPTIONS 2: The data set can be accessed by any number of users for read processing and it can also be accessed by one user for write processing. With this option, VSAM ensures write integrity. If you want read integrity, it is your responsibility to provide the necessary procedures (see "Read Integrity during Cross-Region Sharing") to invalidate data, index buffers, and relate control block information. Only one control block structure can be built to do WRITE operations.
- Cross-region SHAREOPTIONS 3: The data set can be fully shared by any number of users. With this option you are responsible for maintaining both read and write integrity for the data the program accesses. User programs that ignore the write integrity guidelines can cause VSAM program checks, lost or inaccessible records, uncorrectable data set failures, and other unpredictable results. This option places heavy responsiblity on each user sharing the data set.
- Cross-region SHAREOPTIONS 4: The data set can be fully shared by any number of users, and buffers are refreshed for each request. This option requires your program to use ENQ/DEQ to maintain data integrity while sharing the data set. Improper use of ENQ can cause problems similar to those described under SHAREOPTIONS 3.

READ INTEGRITY DURING CROSS-REGION SHARING

You are responsible for ensuring read integrity when the data set is opened for sharing with cross-region SHAREOPTIONS 2, 3, and 4. When your program issues a GET request, VSAM obtains a copy of the control interval containing the requested data record. Another program sharing the data set may also obtain a copy of the same control interval, and may update the data and write the control interval back into the data set. When this occurs, your program has lost read integrity: The control interval copy in your program's buffer is no longer the current copy.

The following should be considered when you are providing read integrity:

- Establish ENQ/DEQ procedures for all requests, read as well as write.

- Decide how to determine and invalidate buffers (index and/or data) that are possibly down level.

- Do not allow secondary allocation. If you do allow secondary allocation you should provide a communication mechanism to the read-only tasks that the extents are increased, force a CLOSE, and then issue another OPEN.

- With an entry-sequenced data set, you must also have the READ use the VERIFY command to update possible down-level control blocks.

- Generally, the loss of read integrity results in down-level data records as an erroneous no-record-found condition.

When your program requires that no updating occur before it completes processing of the requested data record, your program can issue an ENQ to obtain exclusive control over the VSAM data set. (This discussion assumes your program only reads the data record and does not update it. When your program updates the data record, its primary concern is ensuring write integrity and additionally read integrity.) If your program completes processing, it can relinquish control of the data set with a DEQ. When your program is only reading data and not updating it, is probably a good practice to serialize the updates and have the readers wait while the update is occurring. After the update has completed the ENQ/DEQ bracket, the reader must determine the required operations for control block refresh and buffer invalidation based on a communication mechanism or assume that everything is down-level and refresh each request.

WRITE INTEGRITY DURING CROSS-REGION SHARING

You are responsible for ensuring write-integrity if a data set is opened with cross-region SHAREOPTIONS 3 or 4. When an application program issues a "direct" or "skip-sequential" PUT-for-update or no-update (RPL OPTCD=DIR|SKP), the updated control interval is written to direct access storage when you obtain control following a synchronous request (RPL OPTCD=SYN) or following the CHECK macro from an asynchronous request (RPL OPTCD=ASY). To force direct access I/O for a sequential PUT (RPL OPTCD=SEQ), the application program must issue an ENQREQ. Another program sharing the data set may also want to write its updated copy of the same control interval into the data set.

To maintain write integrity for the data set, your program must ensure that there is no conflicting activity against the data set until your program completes updating the control interval. Conflicting activity may be divided into two categories:

1. A data set that is totally preformatted and the only write activity is update-in-place.

   In this case, the sharing problem is simplified by the fact that data cannot change its position in the data set. The lock that must be held for any write operation (PUT RPL OPTCD=UPD) is the unit of transfer that is the control interval. It is your responsibility to associate a lock with this unit of transfer; the record key is not sufficient.

   The following is an example of the required procedures:

   a. Issue a GET for the RPL that has the parameters OPTCD=(SYN,KEY,NUP,DIR),ARG=MYKEY

   b. Determine the RBA of the control interval (RELCI) where the record resides. This is based on the RBA field supplied in the RPL(RPLDDDD).

      RELCI=CISIZE * integer-part-of(RPLDDDD / CISIZE)

   c. Enqueue MYDATA.DSNAME.RELCI (the calculated value)

d. Issue a GET for the RPL that has the parameters OPTCD=(SYN,KEY,UPD,DIR),ARG=MYKEY

e. Issue a PUT for the RPL that has the parameters OPTCD=(SYN,KEY,UPD,DIR)

f. Dequeue MYDATA.DSNAME.RELCI

2. A data set in which record additions and updates with length changes are permitted.

   In this case, the minimum locking unit is a control area to accommodate control interval splits. A higher level lock must be held during operations involving a control area split. The split activity must be serialized at a data set level. To implement a multi-level locking procedure, you must be prepared to provide additional programming using the information provided during VSAM JRNAD processing. This exit is responsible for determining the level of data movement and obtaining the appropriate lock(s).

   The following is a procedure to provide the necessary protection while incurring the penalty of locking all updates at the data set level:

   Enqueue MYDATA.DSNAME

   Issue VSAM request macros

   .
   .
   .

   Dequeue MYDATA.DSNAME

   In any sharing situation, it is a general rule that all resources be obtained and released between the locking protocol. All positioning must be released by using all direct requests or by issuing the ENQREQ macro prior to ending the procedure with the DEQ.

With cross-region SHAREOPTIONS 3, you have the added responsibility of invalidating buffers, data, and/or index. This may be done by the use of an informational control record as the low-key or first record in the data set. The following information is required to accomplish the necessary index record invalidation:

1. Number of data control interval splits and index updates for sequence set invalidation

2. Number of data control area splits for index set invalidation

All data buffers should always be invalidated. In order to perform selective buffer invalidation, an internal knowledge of the VSAM control blocks is required.

Your program must serialize the following types of requests (precede the request with an ENQ and, when the request completes, issue a DEQ):

- All PUT requests.
- POINT, GET-direct-NSP, GET-skip, and GET-for-update requests that are followed by a PUT-insert or PUT-update request.
- VERIFY requests. When VERIFY is executed by VSAM, your program must have exclusive control of the data set.
- Sequential GET and PUT requests.

CROSS-SYSTEM SHARING

The following sharing options, that you may specify when you define a data set, apply in a multiple system environment:

- Cross-system SHAREOPTION 3: The data set may be fully shared. With this option, the access method does nothing to assure integrity. You must assume full responsibility for read and write integrity. Incorrect write-integrity processing can cause access method program checks, lost or inaccessible records, uncorrectable data set failures, and other unpredictable results. This option places very heavy responsibility upon you and it should not be treated lightly.

- Cross-system SHAREOPTION 4: The data set may be fully shared, and buffers used for direct processing are refreshed for each request. The DEQ and ENQ macros are required with this option to maintain data set integrity. Output processing is limited to update and/or add processing that does not change the high-used RBA if DISP=SHR is specified. Data set integrity cannot be maintained unless all jobs having access to the data set in a cross-system environment specify DISP=SHR. Improper use of RESERVE will cause failures similar to those described under SHAREOPTION 3.

Job steps of two or more systems may gain access to the same data set regardless of the disposition specified in each step's JCL. To get exclusive control of a volume, a task in one system must issue a RESERVE macro.

Note: In a shared-DASD environment, integrity cannot be guaranteed by the system when users share a data set for output processing. VSAM does, however, provide assistance in protecting the integrity of the catalog.

In a shared-DASD environment, VERIFY should run before opening the shared data set.

CONTROL BLOCK UPDATE FACILITY (CBUF)

Whenever a data set is opened with cross-region SHAREOPTION 3 or 4, and cross-system SHAREOPTION 3 and DISP=SHR, record management maintains critical control block data in common storage. Associated with this data is a level number that can be checked to ensure the current status of a given data set. It is possible for the cross-system user to use the control block update facility (CBUF) to perform cross-system and cross-region sharing under SHAREOPTION 3.

The cross-system sharing is performed when you send VSAM shared information (VSI) blocks that have changed to the other host at the conclusion of each output request. Generally, the VSIs will not have changed and only a check occurs. The receiving host should scan its ACBs looking for the same open data set with a matching VSI. When found, the old VSI is replaced with the new VSI shipped from the other host. The first VSAM request for the data set results in the invocation of a VSI upgrade routine and the VSAM control blocks are updated to current status.

It should be noted that the SHAREOPTION 3 user must continue to provide read/write integrity. Although VSAM ensures that SHAREOPTION 3 and 4 users will have correct control block information, providing serialization is done correctly, the SHAREOPTION 3 user will not get the buffer invalidation that will occur with SHAREOPTION 4. For LSR and GSR, buffer invalidation and refreshment are available through the use of MRKBFR MARK=DINVALID/XINVALID and WRTBFR TYPE=DRBA.

The restriction to prohibit control area splits under cross-region SHAREOPTION 4 has been eliminated. Therefore, you do not need to restrict code to prevent control area splits, or allow for the control area split error condition. The restriction to prohibit control area splits for cross-systems SHAREOPTION 4 still exists.

CBUF update does not include statistics. (These statistics are number of index levels, number of extents in the data set, number of records, number of deletes, number of bytes of free space, number of control interval splits, number of control area splits, and number of EXCPs.) For those user programs that interrogate statistics, only the activity resulting from the usage of the particular control block structure will be reflected. After all ACBs using a data set have been closed, the catalog will reflect the combined activity of all control block structures.

When improved control interval processing is specified with SHAREOPTION 3 or 4, the data set can be opened; however, if another control block structure extends the data set, the control block structure using improved control interval processing will not be updated unless it is closed and reopened.

To maintain data integrity, the user must use ENQ/DEQ or some other similar function to serialize VSAM requests if ACB DSN sharing was not specified and multiple subtasks are doing output processing or if multiple regions are doing output processing. In either case, VSAM open builds multiple control block structures (even though they might reside within the same storage). The type of serialization required depends on whether read/write integrity or only write integrity is required.

To maintain write integrity, the user for whom multiple control block structures have been built by OPEN must serialize around the following requests:

- PUT
- POINT, GET-DIR-NSP, GET-SKP, or GET-UPD, followed by PUT insert or update request
- VERIFY
- OPEN/CLOSE

Note: If sequential accessing is used, ENDREQ must be used to force write sequential updates and inserts and to release positioning within the data set before releasing control of the data set. For LSR and GSR, buffer ownership must be released by MRKBFR TYPE=RLS and a WRTBFR TYPE=TRN must be issued.

If asynchronous processing is being used, the CHECK must be issued before control of the data set can be released.

For those users for whom multiple control block structures have been built by OPEN and who are using SHAREOPTION 4 data sets, read integrity is provided if retrieval requests are serialized with respect to those requests that modify the data set. Refer to the above discussion of write integrity to see what requests modify the data set and therefore must be serialized to ensure write integrity.

Figure 16 on page 79 illustrates how the SHAREOPTIONS specified in the catalog and the disposition specified on the DD statement interact to affect the type of processing done.

(CR,CS)	DISP=SHR[1]
(3,3)[1]	CBUF
(3,4)[1]	Buffer invalidated. No CA Split.
(4,3)[1]	Buffer invalidated. CBUF
(4,4)[1]	Buffer invalidated. No CA Split.

Legend:

CA = Control Area
CR = Cross-Region
CS = Cross-System
CBUF = Control Block Update Facility
Buffer invalidated = Invalidation of buffers is automatic

[1] When DISP=OLD is specified or the data set is in create or reset mode (regardless of the disposition specified), the share options specified in the catalog are ignored. The data set is processed under the rules for SHAREOPTIONS(1,3). OPEN ensures that the user has exclusive control of the data set within a single system. If the data set can be shared between systems, VSAM does nothing to ensure that another system is not accessing the data set concurrently. With cross-system sharing, the user must ensure that another system is not accessing the data set before specifying DISP=OLD.

Figure 16. Specifying SHAREOPTIONs

If the data set has cross-system SHAREOPTION 4, but does not reside on shared DASD when it is opened, the data set is still processed as a cross-system SHAREOPTION 4 data set on shared DASD; that is, CBUF processing is not provided. When a key-sequenced data set has cross-system SHAREOPTION 4, control area splits are prevented; also, split of the control interval containing the high key of a key range (or data set) is prevented. With control interval access, adding a new control interval is prevented.

INVOCATION

Invocation occurs automatically when the proper share options and DISP=SHR are specified. Programs accessing the VSI must be authorized.

USER INTERACTIONS

If you issue a checkpoint or if a restart occurs, then none of the VSAM data sets open in your region at that time may be using SHAREOPTION 4 processing. If you issue checkpoints, you should open the VSAM data sets that are eligible for SHAREOPTION 4 processing, with a disposition of OLD, or CLOSE them prior to the checkpoint. Note that, if an alternate index was using SHAREOPTION 4 processing, the associated base cluster and any other paths OPEN over that base cluster must also be closed, even if they are not using SHAREOPTION 4 processing.

Code the following instructions to get the length and address of the data to be sent to another processor:

- Get ACB address into register RY:
- To locate the VSI for a data component:

```
L    RX,04(,RY)     Put AMBL address into register RX
L    1,52(,RX)      Get data AMB address
L    1,68(,1)       Get VSI address
LH   0,24(,1)       Load data length
LA   1,24(,1)       Point to data to be communicated
```

- To locate the VSI information for an index component of a key-sequenced data set:

```
L    RX,04(,RY)     Put AMBL address into register RX
L    1,56(,RX)      Get index AMB address
L    1,68(,1)       Get VSI address
LH   0,24(,1)       Load data length
LA   1,24(,1)       Point to data to be communicated
```

ERRORS

Because multiple regions can be sharing the same data set, it is possible for errors occurring in one region to affect any other regions sharing the same data set. If a logical error (Register 15=8) or physical error (Register 15=12) is detected, any control block changes made before the error was detected will be propagated to the shared information in common storage.

When a VSAM data set requires additional space, EOV calls the VSAM catalog to acquire new extents for the VSAM data set, updates the VSAM control block structure for the data set with the new extent information, and updates the critical control block data in common storage so that this new space is accessible by all regions using this VSAM data set. If an abend or unexpected error occurs, which prevents this space allocation from being completed, all regions will be prevented from further extending the data set. To obtain additional space, you must close the VSAM data set in all regions, then reopen it.

You should issue a VERIFY command when sharing, because the VSAM catalog entry for the data set may be inaccurate if the first ACB used to reopen the data set has implicit verify suppressed (ACBSWARN=ON).

CHAPTER 6. OPTIONS FOR ADVANCED APPLICATIONS

PROCESSING CONTROL INTERVALS

Control interval access gives you access to the contents of a control interval; keyed and addressed access give you access to individual data records.

Control interval access is provided for programmers of utilities and system control programs who may need more flexibility than access to data records allows them. With control interval access, you have the option of letting VSAM manage I/O buffers or managing them yourself. With keyed and addressed access, VSAM always manages I/O buffers. If you manage I/O buffers yourself, you have the further option of using improved control interval access for faster processing than with normal control interval access.

This chapter gives the format of a control interval and explains how to use control interval access and manage your own I/O buffers.

THE FORMAT OF A CONTROL INTERVAL

Figure 17 shows the relative positions of data, unused space, and control information in a control interval.

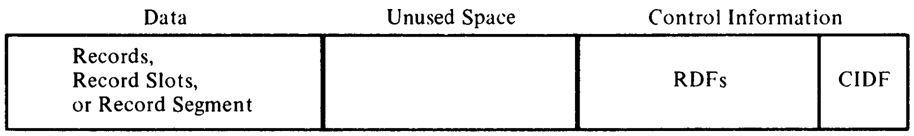


Figure 17. General Format of a Control Interval

Control information consists of a CIDF (control interval definition field) and, for a control interval containing at least one record, record slot, or record segment, one or more RDFs (record definition fields). The CIDF and RDFs are ordered from right to left.

CIDF—Control Interval Definition Field

The CIDF is a 4-byte field that contains two 2-byte binary numbers:

Offset	Length	Description
0(0)	2	The displacement from the beginning of the control interval to the beginning of the unused space, or, if there is no unused space, to the beginning of the control information. This number is equal to the length of the data (records, record slots, or record segment). In a control interval without data, the number is 0.
2(2)	2	The length of the unused space. This number is equal to the length of the control interval, minus the length of the control information, minus the 2-byte value at CIDF+0. In a control interval without data (records, record slots, or record segment), the number is the length of the control interval, minus 4 (the length of the CIDF; there are no RDFs). In a control interval without unused space, the number is 0.
2(2)	1...	Busy flag; set when the control interval is being split; reset when the split is complete.

In an entry-sequenced data set, when there are unused control intervals beyond the last one that contains data, the first of the unused control intervals contains a CIDF filled with 0's. In a key-sequenced or relative record data set or a key-range portion of a key-sequenced data set, the first control interval in the first unused control area (if any) contains a CIDF filled with 0's. A control interval with such a CIDF contains no data or unused space, and is used to represent the software end-of-file (SEOF).

RDF—Record Definition Field

The RBAs of records or relative record numbers of slots in a control interval ascend from left to right. RDFs from right to left describe these records or slots or a segment of a spanned record. RDFs describe records one way for key-sequenced and entry-sequenced data sets and another way for relative record data sets.

In a key-sequenced or entry-sequenced data set, records may vary in length and may span control intervals. An unspanned record with no other records of the same length next to it is described by a single RDF that gives the length of the record. Two or more unspanned records of the same length together are described by a pair of RDFs: The RDF on the right gives the length of each record, and the RDF on the left gives the number of consecutive records of the same length. Each segment of a spanned record (one segment per control interval) is described by a pair of RDFs: The RDF on the right gives the length of the segment, and the RDF on the left gives its update number. (The update number in each segment is incremented by one each time a spanned record is updated. A difference among update numbers within a spanned record indicates a possible error in the record.)

In a relative record data set, records do not vary in length or span control intervals. Each record slot is described by a single RDF that gives its length and indicates whether it contains a record.

An RDF is a 3-byte field that contains a 1-byte control field and a 2-byte binary number:

Offset	Length and Bit Pattern	Description
0(0)	1	Control Field:
	x... ..xx	Reserved
	.x..	Indicates whether there is (1) or is not (0) a paired RDF to the left of this RDF.
	..xx	Indicates whether the record spans control intervals: 00 No. 01 Yes; this is the first segment. 10 Yes; this is the last segment. 11 Yes; this is an intermediate segment.
	.... x...	Indicates what the 2-byte binary number gives: 0 The length of the record, segment, or slot described by this RDF. 1 The number of consecutive unspanned records of the same length, or the update number of the segment of a spanned record.
	.... .x..	For a relative record data set, indicates whether the slot described by this RDF does (0) or does not (1) contain a record.
1(1)	2	Binary number: When bit 4 of byte 0 is 0, gives the length of the record, segment, or slot described by this RDF. When bit 4 of byte 0 is 1 and bits 2 and 3 of byte 0 are 0, gives the number of consecutive records of the same length. When bit 4 of byte 0 is 1 and bits 2 and 3 of byte 0 are not 0, gives the update number of the segment described by this RDF.

KEY-SEQUENCED AND ENTRY-SEQUENCED DATA SETS: In a key-sequenced or entry-sequenced data set with unspanned records, the possible hexadecimal values in the control field of an RDF are:

Left RDF	**Right RDF**	**Description**
	X'00'	This RDF gives the length of a single unspanned record.
X'08'	X'40'	The right RDF gives the length of each of two or more consecutive unspanned records of the same length. The left RDF gives the number of consecutive unspanned records of the same length.

Figure 18 shows the contents of the CIDF and RDFs of a 512-byte control interval containing variable-length unspanned records.

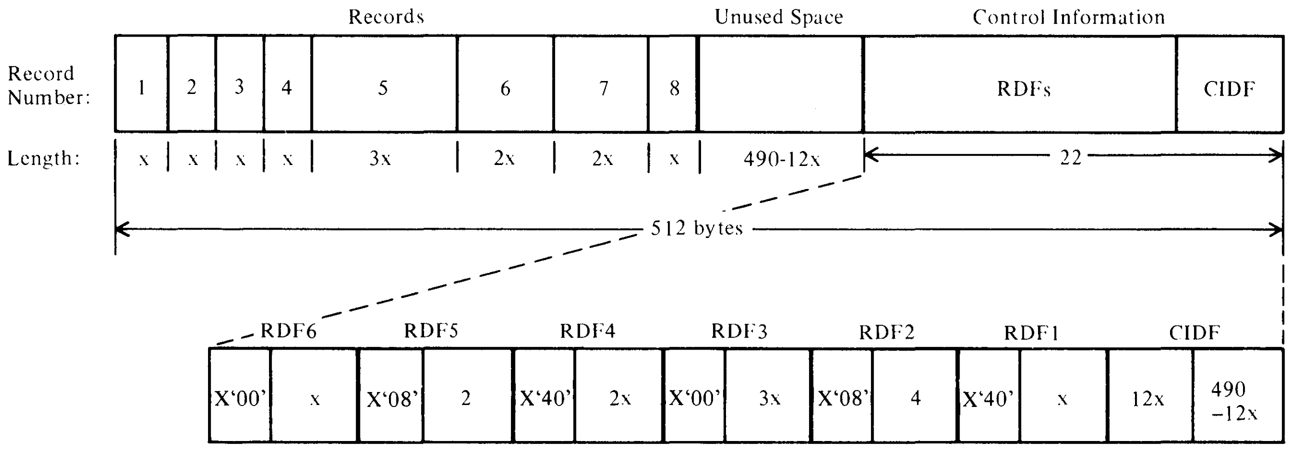


Figure 18. Format of a Control Interval with Unspanned Records

The CIDF and the six RDFs make up 22 bytes of control information. The first 2-byte field in the CIDF gives the total length of the eight records—12x, which is the displacement from the beginning of the control interval to the unused space. The second 2-byte field gives the length of the unused space, which is the length of the control interval minus the total length of the records and the control information—512 minus 12x minus 22, or 490 minus 12x. RDF1 and RDF2 describe the first four records; RDF3 describes the fifth record; RDF4 and RDF5 describe the sixth and seventh records; RDF6 describes the eighth record.

A control interval that contains the record segment of a spanned record contains no other data; it always has two RDFs. The possible hexadecimal values in their control fields are:

Left RDF	Right RDF	Description
X'18'	X'50'	The right RDF gives the length of the first segment of a spanned record. The left RDF gives the update number of the segment.
X'28'	X'60'	The right RDF gives the length of the last segment of a spanned record. The left RDF gives the update number of the segment.
X'38'	X'70'	The right RDF gives the length of an intermediate segment of a spanned record. The left RDF gives the update number of the segment.

Figure 19 shows the contents of the CIDF and RDFs of three 512-byte control intervals that contain the segments of a spanned record. The number n in RDF2 is the update number.

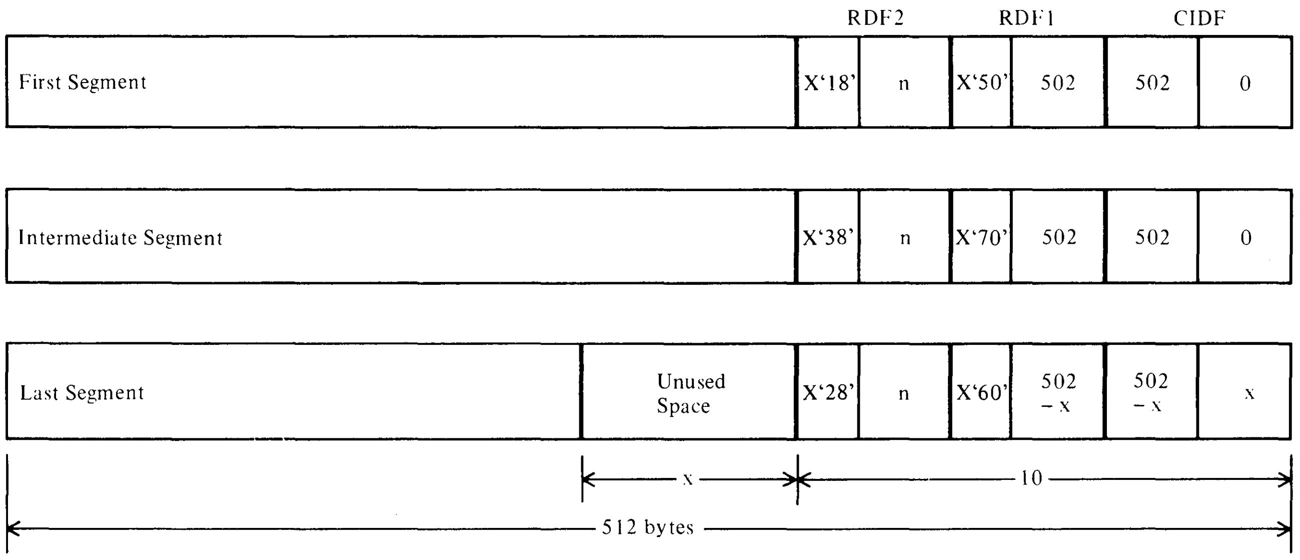


Figure 19. Format of Control Intervals with Spanned Records

Only the control interval that contains the last segment of a spanned record can have unused space. Each of the other segments uses all but the last 10 bytes of a control interval.

In a key-sequenced data set, the control intervals might not be contiguous or in the same order as the segments (that is, for example, the RBA of the second segment can be lower than the RBA of the first segment).

All the segments of a spanned record must be in the same control area. When enough control intervals in a control area are not available for a spanned record, it is stored in a new control area.

RELATIVE RECORD DATA SETS: In a relative record data set, the possible hexadecimal values in the control field of an RDF are:

X'04'	This RDF gives the length of an empty slot.
X'00'	This RDF gives the length of a slot that contains a record.

Every control interval in a relative record data set contains the same number of slots and the same number of RDFs, one for each slot. The first slot is described by the rightmost RDF; the second slot is described by the next RDF to the left, and so on.

HOW TO GAIN ACCESS TO A CONTROL INTERVAL

Control interval access is specified entirely by the ACB MACRF operand and the RPL (or GENCB) OPTCD operand. To prepare for opening a data set for control interval access with VSAM managing I/O buffers, specify:

```
  ACB       MACRF=(CNV,...),...
```

NUB (no user buffering) and NCI (normal control interval access) may be specified or taken as defaults. (This section does not consider the MACRF operands UBF (user buffering) and ICI (improved control interval access); their use is described under "Managing Your Own I/O Buffers" on page 87.)

With NUB and NCI, you may specify in the MACRF operand that the data set is to be opened for keyed and addressed access as well as for control interval access. For example, MACRF=(CNV, KEY, SKP, DIR, SEQ, NUB, NCI, OUT) is valid.

You define a particular request for control interval access by coding:

```
  RPL       OPTCD=(CNV,...),...
```

In general, control interval access with no user buffering has the same freedoms and limitations as keyed and addressed access have. Control interval access may be synchronous or asynchronous, may have the contents of a control interval moved to your work area (OPTCD=MVE) or left in VSAM's I/O buffer (OPTCD=LOC), and may be defined by a chain of request parameter lists (except with OPTCD=LOC specified).

Both direct and sequential access may be used with control interval access, but skip sequential access may not. That is, you may specify OPTCD=(CNV,DIR) or (CNV,SEQ), but not OPTCD=(CNV,SKP).

With sequential access, VSAM takes an EODAD exit when you try to retrieve the control interval whose CIDF is filled with 0's or, if there is no such control interval, when you try to retrieve a control interval beyond the last one. However, VSAM does not prevent you from using a direct GET or a POINT and a sequential GET to retrieve the control interval whose CIDF is filled with 0's or any of the following control intervals.

The RPL (or GENCB) operands AREA and AREALEN have the same use for control interval access in relation to OPTCD=MVE or LOC as they do for keyed and addressed access: With OPTCD=MVE, AREA gives the address of the area into which VSAM moves the contents of a control interval; with OPTCD=LOC, AREA gives the address of the area into which VSAM puts the address of the I/O buffer containing the contents of the control interval.

The search argument for a direct request with control interval access is the RBA of the control interval whose contents are desired.

You can define a password especially for control interval access. Its authority is second only to that of the master password; it authorizes control interval access as well as update and read-only access. Because control interval access gives you access to control information, you should consider protecting your data sets with control interval passwords.

All the request macros except ERASE can be used for control interval access: GET, PUT, POINT, CHECK, and ENDREQ. To update the contents of a control interval, you must (with no user buffering) previously have retrieved them for update. You cannot alter the contents of a control interval with OPTCD=LOC specified.

With keyed access and addressed access, VSAM maintains the control information in a control interval. With control interval access, you are responsible for that information.

You may load an entry-sequenced data set with control interval macro access. If you open an empty entry-sequenced data set, VSAM allows you to use only sequential storage. That is, you may issue only PUTs, with OPTCD=(CNV,SEQ,NUP). PUT with OPTCD=NUP stores information in the next available control interval (at the end of the data set).

You may not load or extend a key-sequenced data set with control interval access, because VSAM could not build index records for the data set. VSAM also prohibits you from extending a relative record data set by way of control interval access.

To update the contents of a control interval, you can either:

- Retrieve them with OPTCD=UPD and store them back, or,
- Without retrieving them, store new contents in the control interval with OPTCD=UPD (user buffering only).

PUT with OPTCD=UPD stores information in the control interval specified. If a GET for update precedes the PUT, specification is automatic. If no GET (or a GET with OPTCD=NUP) precedes the PUT, you have to specify the RBA of the control interval in the ARG operand.

MANAGING YOUR OWN I/O BUFFERS

With ACB MACRF=(CNV,UBF) specified (control interval access with user buffering), the work area specified by the RPL (or GENCB) AREA operand is, in effect, the I/O buffer: VSAM transmits the contents of a control interval directly between the work area and direct access storage.

If you specify user buffering, you cannot specify KEY or ADR in the MACRF operand; you can specify only CNV. That is, you cannot intermix keyed and addressed requests with requests for control interval access.

OPTCD=LOC is inconsistent with user buffering and is not allowed.

Improved Control Interval Access

With user buffering you may, if you follow the restrictions below, specify improved control interval access:

```
ACB      MACRF=(CNV,UBF,ICI,...),...
```

Processing is faster than that of normal control interval access. A processing program can achieve the best performance with improved control interval access by combining it with SRB dispatching. (SRB dispatching is described in System Programming Library: Supervisor.)

To open an object for improved control interval access, the named object must:

- Not be empty.

- Be an entry-sequenced or relative record cluster; the data component of a key-sequenced, entry-sequenced, or relative record cluster; or the index component of a key-sequenced cluster (index records must not be replicated).

- Have control intervals the same size as physical records. When you use the access method services DEFINE command to define the object, you can specify control interval size equal to a physical record size used for the device on which the object is stored. VSAM uses physical record sizes of 512, 1024, 2048, and 4096 bytes.

The following table identifies the direct access devices for which the physical record size equal to the control interval size is selected for a data component. The physical record size is always equal to the control interval size for an index component.

Device	**Control**	**Interval**	**Size**	**(Bytes)**
	512	**1024**	**2048**	**4096**
2305-1	X	X	X	X
2305-2	X	X	X	X
2314	X	X	X	
2319	X	X	X	
3330	X	X	X	X
3330-1	X	X	X	X
3340	X	X		X
3344	X	X		X
3350	X	X	X	X
3375	X	X	X	X
3380	X	X	X	X

If these restrictions are not met, the object cannot be opened.

To process a data set with improved control interval access, a request must be:

- Defined by a single RPL (VSAM ignores the NXTRPL operand).

- A direct GET, GET for update, or PUT for update (no POINT, no processing empty data sets; a relative record data set with slots formatted is considered not to be empty, even if no slot contains a record).

- Synchronous (no CHECK, no ENDREQ).

With improved control interval access, VSAM assumes (without checking) that an RPL whose ACB has MACFR=ICI has OPTCD=(CNV, DIR, SYN) that a PUT is for update (OPTCD=UPD), and that your buffer length (specified in RPL AREALEN=number) is correct. Because VSAM does not check these operands, you should debug your program with ACB MACRF=NCI, then change to ICI.

With improved control interval access, VSAM does not take JRNAD exits and does not keep statistics (which are normally available by way of SHOWCB).

With improved control interval access, you may specify that control blocks are to be fixed in real storage (ACB MACRF=(CFX,...)). If you so specify, your I/O buffers must also be fixed in real storage. Having your control blocks fixed in real storage, but not your I/O buffers, may cause physical errors or unpredictable results. If you specify MACRF=CFX without ICI, VSAM ignores CFX. NFX is the default; it indicates that buffers are not fixed in real storage, except for an I/O operation. A program must be authorized to fix pages in real storage, either in supervisor state with protection key 0 to 7, or link-edited with authorization. (The authorized program facility is described in System Programming Library: Supervisor.) An unauthorized request is ignored.

CBIC (Control Blocks in Common) Option

The CBIC option places the VSAM control blocks associated with a VSAM data set into the common service area (CSA). The control block structure and VSAM I/O operations are essentially the same whether or not the CBIC option is invoked, except for the location of the control block structure. The user-related control blocks are generated in the protect key (0 through 7); the system-related control blocks are generated in protect key 0. The VSAM control block structure generated when the CBIC option is invoked retains normal interfaces to the region that opened the VSAM data set (for example, the DEB is chained to the region's TCB).

The CBIC option is invoked when a VSAM data set is opened. To invoke the CBIC option, you set the CBIC flag (located at offset X'33' (ACBINFL2) in the ACB, bit 2 (ACBCBIC)) to one. When your program opens the ACB with the CBIC option set, your program must be in supervisor state with a protect key from 0 to 7; otherwise, VSAM will not open the data set.

The CBIC option must be used only when the ICI option is also specified. The following restrictions apply when using the CBIC option:

- You cannot also specify LSR or GSR.
- The following types of data sets cannot be used with the CBIC option: catalogs, user or system CRAs, VVICs (virtual volume inventory control), swap data sets, or system data sets.
- If CBIC data sets are being used in a region, your program cannot take a VSAM checkpoint.

If another region accesses the data set's control block structure in the CSA via VSAM record management, the following conditions should be observed:

- An OPEN macro should not be issued against the data set.
- The ACB of the user who opened the data set with the CBIC option must be used.
- CLOSE and temporary CLOSE cannot be issued for the data set (only the user who opened the data set with the CBIC option can close the data set).
- A region must have the same storage protect key as the user who opened the data set with the CBIC option.
- User exit routines should be accessible from all regions accessing the data set with the CBIC option.

SRB OR CROSS-MEMORY INVOCATION

In order to function in cross-memory or SRB mode, record management does not issue any SVCs or take any user exits as they were entered. In lieu of issuing SVCs, RPL return codes are set to indicate that an SVC (such as an end of volume) is required, in order to complete the request. You must switch to TCB mode and reissue the request in noncross-memory mode in the storage from which the data set was opened.

Cross-memory or SRB mode invocation must only occur for synchronous, supervisor state requests. An attempt to invoke VSAM asynchronously in either mode results in a logical error. Also, VSAM makes no attempt to synchronize cross-memory mode requests. This means that the RPL must specify WAITX, and a UPAD exit must be provided to handle cross-memory request synchronization. Failure to provide a UPAD routine that ensures the ECB is posted before returning to VSAM will cause a logical error.

INVOCATION

Invocation of this function occurs whenever VSAM action macros are issued by a program in SRB or cross-memory mode (home address space ID, HASID, unequal to primary address space ID or HASID unequal to secondary HASID and secondary addressing bit on the current PSW). No locks may be held at VSAM entry, and VSAM assumes this is the case.

The caller must be in supervisor state; a UPAD routine must be provided for cross-memory mode invocation; the ACB must specify MACRF=LSR, MACRF=GSR, or MACRF=ICI; and the RPL must specify WAITX and SYN. At the time a VSAM request is issued, proper addressability must exist to the RPL, ECB, and user data areas. Therefore, these must be either in CSA or in the same address space that opened the data set.

If any requests to a VSAM data set or to a shared resource pool are in supervisor state, all should be, otherwise it would sometimes be necessary for a nonsupervisor state request to resume a deferred supervisor state request. Alternatively, you may wish to design your UPAD POST routine to handle this situation, thereby avoiding the restriction.

USER INTERACTION

Whenever VSAM cannot avoid the SVC, it sets an RPL return code to indicate that you must switch to TCB, noncross-memory mode and reissue the VSAM request. For performance reasons, these instances are kept to a minimum. Areas identified as requiring TCB, noncross-memory mode are EXCEPTIONEXIT, loaded exits, EOV, dynamic string addition, AIX processing, and MSS-related macros.

ERRORS

VSAM's I/O is done under the functional recovery routine (FRR) to ensure related actions (such as page fixes or obtaining the local lock) are cleaned up in the event of failure.

The remainder of VSAM processing does not run under a VSAM FRR or ESTAE routine. Therefore, if processing is to continue after a VSAM error, the user must provide an ESTAE routine that issues a TERMRPL macro instruction. In an attempt to allow processing to continue, TERMRPL frees VSAM resources associated with the RPL.

To help debug errors that may arise during cross-memory processing, VSAM saves the primary, secondary, and current ASIDs in an internal control block, IOMB.

If VSAM processing is to be done under a user-established FRR, a UPAD exit routine must be provided to allow for deleting the FRR when the UPAD is taken for a POST. VSAM makes no attempt to preserve any FRR status that may have been established by its caller.

If a logical error or an end-of-data condition occurs during cross-memory or SRB processing, VSAM attempts to enter the LERAD or EODAD routine. If the routine must be loaded, it cannot be taken; VSAM sets the RPL feedback to indicate "invalid TCB." If an I/O error occurs during cross-memory or SRB processing and an

EXCEPTIONEXIT or loaded SYNAD routine is specified, these routines cannot be taken; the RPL feedback indicates an I/O error condition.

MESSAGES AND CODES

If VSAM cannot process the request because an SVC is required, it issues a logical error with the RPL feedback set to 120(78), indicating an invalid TCB mode request.

If the caller is in SRB or cross-memory mode but the RPL specifies ASY or the ACB specifies NSR, or if the cross-memory mode caller is not in supervisor state, a logical error with a feedback code of 228(E4) is set in the RPL. If the cross-memory mode caller does not have WAITX specified in the RPL or lacks an active UPAD, a logical error code of 232(E8) is indicated in the RPL.

If processing fails at a point at which TERMRPL cannot release the resources needed to continue, the RPL contains a logical error with a feedback code of 240(F0) to indicate that the data set or the shared resource pool is in an indeterminate state. The ESTAE routine should cause termination in that case.

SHARING RESOURCES AMONG DATA SETS

VSAM has a set of macros that enables you to share I/O buffers, I/O-related control blocks, and channel programs among many VSAM data sets and to manage I/O buffers.

Sharing these resources is not the same as sharing a data set itself (that is, sharing among different tasks that independently open it). Data set sharing can be done with or without sharing I/O buffers, I/O-related control blocks, and channel programs. For a discussion of data set sharing, see "Chapter 5. Sharing a VSAM Data Set" on page 69.

In VSAM, an I/O buffer is a virtual storage area from which the contents of a control interval are read and written. The I/O-related control blocks that are shared are: PLH, IOMB, IOSB, IQE, IRB, and SRB.

Sharing these resources optimizes their use to reduce the requirement for virtual storage and therefore to reduce paging of virtual storage. Managing I/O buffers includes:

- Deferring writes for direct PUT requests, which reduces the number of I/O operations.
- Writing buffers that have been modified by related requests.
- Locating buffers that contain the contents of specified control intervals.
- Marking a buffer to be written without issuing a PUT.
- When your program accesses an invalid buffer, VSAM refreshes the buffer (that is, reads in a fresh copy of the control interval) before making its contents available to your program.

Managing I/O buffers should enable you to speed up direct processing of VSAM data sets whose activity is unpredictable. You probably will not be able to speed up sequential processing or processing of a data set whose activity is consistently heavy.

When you share resources for sequential access, positioning at the beginning of a data set has to be explicit: with the POINT macro or the direct GET macro with RPL OPTCD=NSP. With shared resources, VSAM does not automatically position itself at the beginning of a data set opened for sequential access, because placeholders belong to the resource pool, not to individual data sets. You may not use a resource pool to load records into an empty data set. (Sharing would not improve sequential processing, and it would take up buffers and could hinder other processing.)

The macros you use to share resources and manage I/O buffers are:

- BLDVRP (build VSAM resource pool)
- DLVRP (delete VSAM resource pool)
- WRTBFR (write buffer)
- SCHBFR (search buffer)
- MRKBFR (mark buffer)

Certain operands in the ACB and RPL macros are used with these macros. The use of these macros and operands and of SHOWCB, and TESTCB macros is described in this chapter. VSAM Reference explains operand notations for coding macros.

PROVIDING A RESOURCE POOL

To share resources, you use the BLDVRP macro to build a resource pool. You code a MACRF operand in the ACB and use OPEN to connect your data sets to the resource pool. After you have closed all the data sets, you use the DLVRP macro to delete the resource pool.

You may share resources locally or globally:

- LSR (local shared resources). Each partition or address space may have one resource pool independently of other partitions or address spaces.
- GSR (global shared resources). All address spaces for a given protection key in the system share one resource pool. A resource pool can be built for each of the protection keys 0 through 7. With GSR, an access method control block and all related request parameter lists, exit lists, data areas, and extent control blocks must be in the common area of virtual storage with protection key the same as that of the resource pool. To get storage in the common area with that protection key, issue the GETMAIN macro while in that key, for storage in subpool 241. Generate ACBs, RPLs, and EXLSTs with the GENCB macro—code the WAREA and LENGTH operands. The program that issues macros related to that global resource pool must be in supervisor state with the same key. (The macros are: BLDVRP, CHECK, CLOSE, DLVRP, ENDREQ, ERASE, GENCB, GET, GETIX, MODCB, MRKBFR, OPEN, POINT, PUT, PUTIX, SCHBFR, SHOWCB, TESTCB, and WRTBFR. The SHOWCAT macro is not related to a resource pool, because it is issued independently of an open data set.)

You may have both a global resource pool and local resource pools. Tasks in an address space that has a local resource pool may use either the global resource pool, under the restrictions described above, or the local resource pool.

Deciding How Big a Resource Pool to Provide

You have to provide a resource pool before any clusters or alternate indexes are opened to use it. Specifying the BUFFERS, KEYLEN, and STRNO operands of the BLDVRP macro requires knowledge of the size of the control intervals, data records (if spanned), and key fields in the components that will use the resource pool and knowledge about the way the components are processed. (See "BLDVRP: Building a Resource Pool" on page 93.)

DISPLAYING INFORMATION ABOUT AN UNOPENED DATA SET: The SHOWCAT macro enables you to get information about a component before its cluster or alternate index is opened. The program that is to issue BLDVRP can issue SHOWCAT on all the components to find out the sizes of control intervals, records, and keys. This information enables the program to calculate values for the BUFFERS and KEYLEN operands of BLDVRP.

A program need not be in supervisor state with protection key 0 to 7 to issue SHOWCAT, even though it must be in supervisor state and in protection key 0 to 7 to issue BLDVRP TYPE=GSR.

The SHOWCAT macro is described in Catalog Users Guide.

DISPLAYING STATISTICS ABOUT A BUFFER POOL: You can get statistics about the usage of buffer pools to determine how you could improve a previous definition of a resource pool and the mix of data sets that use it. The SHOWCB macro enables you to get these statistics. The statistics are available via an ACB that describes an open data set that is using the buffer pool. They reflect the usage of the buffer pool from the time it was built to the time SHOWCB is issued. All the statistics except one are for a single buffer pool. To get statistics for the whole resource pool, issue SHOWCB for each buffer pool in it.

The statistics cannot be used to redefine the resource pool while it is in use. You have to make adjustments the next time you build it.

The use of SHOWCB to display an ACB is described in VSAM Reference. If the ACB has MACRF=GSR, the program that issues SHOWCB must be in supervisor state with protection key 0 to 7. For buffer pool statistics, the following keywords are specified in the FIELDS operand:

```
SHOWCB   FIELDS=([BFRFND][,BUFRDS][,NUIW][,STRMAX]
         [,UIW],...),...
```

These fields may be displayed only after the data set described by the ACB is opened. Each field requires one fullword in the display work area:

Field	Description
BFRFND	The number of requests for retrieval that could be satisfied without an I/O operation (the data was found in a buffer)
BUFRDS	The number of reads to bring data into a buffer
NUIW	The number of nonuser-initiated writes (writes that VSAM was forced to do because no buffers were available for reading the contents of a control interval)
STRMAD	The maximum number of placeholders currently active for the resource pool (for all of the buffer pools in it)
UIW	The number of user-initiated writes (PUTs not deferred or WRTBFRs—see "Deferring Write Requests" on page 94.)

Return Codes from SHOWCB: The return codes from VSAM for SHOWCB with these keywords are the same as for other keywords and are given in VSAM Reference. With MACRF=GSR specified in the ACB, a program check can occur if SHOWCB is issued by a program that is not in supervisor state with the same protection key as that of the resource pool.

BLDVRP: Building a Resource Pool

To share resources locally, a task in the partition or address space issues the BLDVRP (build VSAM resource pool) macro, TYPE=LSR. To share resources globally, a system task issues BLDVRP, TYPE=GSR. Issuing BLDVRP causes VSAM to share the I/O buffers, I/O-related control blocks, and channel programs of data sets whose ACBs indicate the corresponding option for shared resources. (See "OPEN: Connecting a Data Set to a Resource Pool" on page 94.) Control blocks and channel programs are shared automatically. But you may control the sharing of buffers.

When you issue BLDVRP, you specify the number of buffer pools in the resource pool and the size and number of buffers in each buffer pool. A data set uses the buffer pool whose buffers are the appropriate size—either the exact size needed, if there are any, or the next larger size available. The data set uses only the one buffer pool.

TYPE=({<u>LSR</u>|GSR},...),...

This specifies whether a local (LSR) or a global (GSR) resource pool is to be built. Only one BLDVRP TYPE=LSR may be issued per partition or address space; only one BLDVRP TYPE=GSR may be issued for the system for each of the protection keys 0 through 7. The program that issues BLDVRP TYPE=GSR must be in supervisor state with key 0 to 7.

DLVRP: Deleting a Resource Pool

After all data sets using a resource pool are closed, you delete the resource pool by issuing the DLVRP (delete VSAM resource pool) macro. Failure to delete a local resource pool causes virtual storage to be lost until the end of the job step. This loss is protected with a global resource pool: If the address space that issued BLDVRP terminates without having issued DLVRP, the system deletes the global resource pool when its use count is 0.

OPEN: Connecting a Data Set to a Resource Pool

You cause a data set to use a resource pool that was built by BLDVRP by specifying LSR or GSR in the MACRF operand of the data set's ACB before you open the data set.

ACB   MACRF=({<u>NSR</u>|LSR|GSR},...),...

NSR, the default, indicates the data set does not use shared resources. LSR indicates it uses the local resource pool. GSR indicates it uses the global resource pool.

When an ACB indicates LSR or GSR, VSAM ignores its BSTRNO, BUFNI, BUFND, BUFSP, and STRNO operands because VSAM will use the existing resource pool for the resources associated with these parameters.

For a data set described by an ACB with MACRF=GSR, the ACB and all related RPLs, EXLSTs, ECBs, and data areas must be in the common area of virtual storage with the same protection key as that of the resource pool.

Deferring Write Requests

VSAM automatically defers writes for sequential PUT requests. It normally writes out the contents of a buffer immediately for direct PUT requests. With shared resources, you can cause writes for direct PUT requests to be deferred. Buffers are finally written out:

- When you issue the WRTBFR macro
- When VSAM needs a buffer to satisfy a GET request
- Or when a data set using a buffer pool is closed (temporary CLOSE is ineffective against a data set that is sharing

buffers; nor does ENDREQ cause buffers in a resource pool to be written)

Deferring writes saves I/O operations when subsequent requests can be satisfied by the data in the buffer pool. Data processing performance with VSAM will be improved if control intervals are updated more than once.

You indicate that writes are to be deferred by coding the MACRF DFR option in the ACB, along with MACRF=LSR or GSR.

```
  ACB        MACRF=({LSR|GSR},{DFR|NDF},...),...
```

NDF, the default, indicates that writes are not to be deferred for direct PUTs.

The DFR option is incompatible with SHAREOPTIONS 4. (SHAREOPTIONS is a parameter of the DEFINE command of access method services. It is described in Access Method Services Reference.) A request to open a data set with SHAREOPTIONS 4 for deferred writes is rejected. (See "OPEN: Connecting a Data Set to a Resource Pool" on page 94.)

Relating Deferred Requests by Transaction ID

You can relate action requests (GET, PUT, and so forth) according to transaction by specifying the same ID in the RPLs that define the requests.

The purpose of relating the requests that belong to a transaction is to enable WRTBFR to cause all the modified buffers used for a transaction to be written. When the WRTBFR request is complete, the transaction is physically complete.

```
  RPL        TRANSID=number,...
```

TRANSID specifies a number from 0 to 31. The number 0, which is the default, indicates that requests defined by the RPL are not associated with other requests. A number from 1 to 31 relates the request(s) defined by this RPL to the request(s) defined by other RPLs with the same transaction ID.

You can find out what transaction ID an RPL has by issuing SHOWCB or TESTCB. Their use with an RPL is described in VSAM Reference.

```
  SHOWCB     FIELDS=([TRANSID],...),...
```

TRANSID requires one fullword in the display work area.

```
  TESTCB     TRANSID=number,...
```

If the ACB to which the RPL is related has MACRF=GSR, the program that issues SHOWCB or TESTCB must be in supervisor state with the same protection key as that of the resource pool.

Return Codes from SHOWCB and TESTCB: The return codes from VSAM for SHOWCB and TESTCB with the TRANSID keyword are the same as with other keywords and are given in VSAM Reference. With MACRF=GSR specified in the ACB to which the RPL is related, a program check can occur if SHOWCB or TESTCB is issued by a program that is not in supervisor state with protection key 0 to 7.

WRTBFR: Writing Buffers Whose Writing Is Deferred

If DFR is specified in the ACB of any data set that is using a resource pool, you can use the WRTBFR (write buffer) macro to write:

- All modified buffers for a given data set

- All modified buffers in the resource pool
- The least recently used modified buffers in each buffer pool in the resources pool
- All buffers that were modified by requests with the same transaction ID (see "Relating Deferred Requests by Transaction ID" on page 95)
- A buffer, identified by an RBA value, that has been modified and has a use count of zero

You can specify the DFR option in an ACB without using the WRTBFR to write buffers—a buffer is written when VSAM needs one to satisfy a GET request, or all modified buffers are written when the last of the data sets that uses them is closed.

Besides using WRTBFR to write buffers whose writing is deferred, you can use it to write buffers that are marked for output with the MRKBFR macro, which is described in "MRKBFR: Marking a Buffer of Output" on page 102.

Using WRTBFR can improve performance, if you schedule WRTBFR to overlap other processing.

VSAM notifies the processing program when there are no more unmodified buffers into which to read the contents of a control interval. (VSAM would be forced to write buffers when another GET request required an I/O operation.) VSAM sets register 15 to 0 and puts 12 (X'0C') in the feedback field of the RPL that defines the PUT request that detects the condition.

VSAM also notifies the processing program when there are no buffers available to which to assign a placeholder for a request. This is a logical error (register 15 contains 8 unless an exit is taken to a LERAD routine); the feedback field in the RPL contains 152 (X'98'). You may retry the request; it gets a buffer if one is freed.

When sharing the data set with a user in another region, your program might want to write the contents of a specified buffer without writing all other modified buffers. Your program issues the WRTBFR macro to search your buffer pool for a buffer containing the specified RBA. If found, the buffer is examined to verify that it is modified and has a use count of zero. If so, VSAM writes the contents of the buffer into the data set.

Note: Before using WRTBFR TYPE=CHK|TRN|DRBA, be sure to release all buffers ("Multi-String Processing" on page 29 for details about releasing buffers). If one of the buffers is not released, VSAM defers processing until the buffer is released.

Handling Exits to Physical Error Analysis Routines

With deferred writes of buffers, a processing program continues after its PUT request has been completed, even though the buffer has not been written. The processing program is not synchronized with a physical error that occurs when the buffer is finally written. A processing program that uses MRKBFR MARK=OUT (see "MRKBFR: Marking a Buffer of Output" on page 102) is also not synchronized with a physical error. An EXCEPTION or a SYNAD exit routine must be supplied to analyze the error.

The ddname field of the physical error message identifies the data set that was using the buffer, but, because the buffer might have been released, its contents might be unavailable. You can provide a JRNAD exit routine to record the contents of buffers for I/O errors (see "Using the JRNAD Exit with Shared Resources" on page 97). It can be coordinated with a physical error analysis routine to handle I/O errors for buffers whose writing has been deferred.

If a JRNAD exit routine is used to cancel I/O errors during a transaction, the physical error analysis routine will get only the last error return code.

Using the JRNAD Exit with Shared Resources

In addition to the reasons for having a JRNAD exit routine that are described in "Chapter 8. User-Written Exit Routines" on page 129, VSAM takes the JRNAD exit for the following reasons when the exit is associated with a data set whose ACB has MACRF=LSR or GSR:

- A data or index control interval buffer has been modified and is about to be written.
- A physical error occurred: VSAM takes the JRNAD exit first—your routine can direct VSAM to bypass the error and continue processing or to terminate the request that occasioned the error and proceed with error processing.
- A control area is about to be split.

Note: For the ICF catalog, the following reasons also apply:

- A data or index control interval is being read for shared or exclusive use.
- Exclusive control has been released on a control interval.
- A READ or WRITE operation has completed.
- The contents of a buffer have been made invalid.

"JRNAD Exit Routine to Journalize Transactions" on page 133 gives the contents of the registers when VSAM exits to the JRNAD routine. Some of the fields in the parameter list that register 1 points to have special meanings for buffer modification, I/O errors, and control area splits:

Note: For the ICF catalog, fields at locations 8, 12, 16, and 20 have meaning for the values listed below:

Offset	**Values**
8(8)	X'30'-X'4C'
12(C)	X'30',X'34',X'38',X'3C',X'48'-X'4C'
16(10)	X'30'-X'44'
20(14)	X'3C',X'40',X'44',X'4C'

Offset	**Length**	**Description**
0(0)	4	No special meaning. (Address of the RPL that defines the request that occasioned VSAM's exit to the routine.)
4(4)	4	No special meaning. (Address of a 5-byte field that identifies the data set being processed.)

Offset	Length	Description	
8(8)	4	Variable, depends on the reason indicator at offset 20:	
		X'20'	The RBA of the beginning of the control area about to be split.
		X'24'	The address of the I/O buffer into which data was going to be read.
		X'28'	The address of the I/O buffer from which data was going to be written.
		X'2C'	The address of the I/O buffer that contains the control interval contents that are about to be written.
		X'30'	A data or index control interval is about to be read in exclusive control.
		X'34'	A data or index control interval is about to be read in shared status.
		X'38'	Acquire exclusive control of a control interval already in the buffer pool.
		X'3C'	Build a new control interval for the data set and hold it in exclusive control.
		X'40'	Exclusive control of the indicated control interval has been released.
		X'44'	The contents of the indicated control interval have been made invalid.
		X'48'	READ completed.
		X'4C'	WRITE completed.
12(C)	4	Variable, depends on the reason indicator at offset 20:	
		X'20'	Unpredictable.
		X'24'	Unpredictable.
		X'28'	Bits 0 through 31 correspond with transaction IDs 0 through 31. Bits set to 1 indicate that the buffer that was being written when the error occurred was modified by the corresponding transactions. You can set additional bits to 1 to tell VSAM to keep the contents of the buffer until the corre-sponding transactions have modified the buffer.
		X'2C'	The size of the control interval whose contents are about to be written.

Offset	Length	Description	
		X'30'	The size of the control interval is about to be read in exclusive control.
		X'34'	A data or index control interval is about to be read in shared status.
		X'38'	Acquire exclusive control of a control interval already in the buffer pool.
		X'3C'	Build a new control interval for the data set and hold it in exclusive control.
		X'48'	READ completed.
		X'4C'	WRITE completed.
16(10)	4	Variable, depends on the reason indicator at offset 20:	
		X'20'	The RBA of the last byte in the control area about to be split.
		X'24'	The fourth byte contains the physical error code from the RPL FDBK field. You use this fullword to communicate with VSAM. Setting it to 0 indicates that VSAM is to ignore the error, bypass error processing, and let the processing program continue. Leaving it nonzero indicates that VSAM is to continue as usual: terminate the request that occasioned the error and proceed with error processing, including exiting to a physical error analysis routine.
		X'28'	Same as for X'24'.
		X'2C'	The RBA of the control interval whose contents are about to be written.
		X'30'	Not applicable.
		X'34'	Not applicable.
		X'38'	Not applicable.
		X'3C'	Not applicable.
		X'40'	Not applicable.
		X'44'	Not applicable.
		X'48'	READ completed.
		X'4C'	WRITE completed.

Offset	Length	Description	
20(14)	1	Indication of the reason VSAM exited to the JRNAD routine:	
		X'20'	A control area is about to be split. During the split, VSAM exits to the JRNAD routine as each control interval that is modified is about to be written (see '2C' below).
		X'24'	A physical input error occurred.
		X'28'	A physical output error occurred.
		X'2C'	A modified data or index control interval is about to be written.
		X'30'	A data or index control interval is about to be read in exclusive control.
		X'34'	A data or index control interval is about to be read in shared status.
		X'38'	Acquire exclusive control of a control interval already in the buffer pool.
		X'3C'	Build a new control interval for the data set and hold it in exclusive control.
		X'40'	Exclusive control of the indicated control interval has been released.
		X'44'	The contents of the indicated control interval have been made invalid.
		X'48'	READ completed.
		X'4C'	WRITE completed.
21(15)	1	Reserved.	

Preventing Deadlock in Exclusive Control

Contention for VSAM data (the contents of a control interval) can lead to deadlocks, in which a processing program is prevented from continuing because its request for data cannot be satisfied. A and B can engage as contenders in four distinct ways:

1. A wants exclusive control, but B has exclusive control. VSAM refuses A's request: A must either do without the data or retry its request.

2. A wants exclusive control, but B is only willing to share. VSAM queues A's request (without notifying A that it must wait) and gives A use of the data when B releases it.

3. A wants to share, but B has exclusive control. VSAM refuses A's request: A must either do without the data or retry its request.

4. A wants to share, and B is willing to share. VSAM gives A use of the data, along with B.

VSAM's action in a contention for data rests on the assumptions that, if a processing program has exclusive control of the data, it will (or at least might) update or delete it and that, if a processing program is updating or deleting the data, it has exclusive control of it. (The use of MRKBFR, MARK=OUT provides an exception to the second assumption: A processing program can update the contents of a control interval without exclusive control of them; see "MRKBFR: Marking a Buffer of Output" on page 102.)

In 1 and 3, above, B is responsible for giving up exclusive control of a control interval by way of an ENDREQ, a MRKBFR, a MARK=RLS, or a request for access to a different control interval. (The RPL that defines the ENDREQ, MRKBFR, or request is the one that was used to acquire exclusive control in the first place.)

Using Control Interval Access with Shared Resources

When you share resources, you can use two macros (SCHBFR and MRKBFR) to locate a buffer in a buffer pool and to mark it for output. You can update its contents (which are the contents of a control interval) independently of much of VSAM record management. Doing this might improve your data processing performance; it allows you to bypass some of the restrictions VSAM imposes on the way you treat VSAM data sets.

However, bypassing normal restrictions might prevent you from processing the data sets normally. For instance, key-sequenced data sets depend on the accuracy of their indexes and the RDFs and the CIDF in each control interval. You should not update key-sequenced data sets with control interval access.

Control interval access with SCHBFR and MRKBFR is intended primarily for processing entry-sequenced data sets. The definition of an entry-sequenced data set includes the unchangeability of RBAs and the permanence of records. With control interval access, you can change RBAs in a control interval and delete records by modifying the RDFs and the CIDF. However, if the data set has alternate indexes, you are responsible for upgrading them. If you have specified that you are doing keyed or addressed access (ACB MACRF={KEY|ADR},...) and control interval access, then those requests for keyed or addressed access (RPL OPTCD={ KEY|ADR},...) will cause VSAM to upgrade the alternate indexes. Those requests specifying control interval access will not cause upgrading of the alternate indexes. You are responsible for upgrading them. Upgrading an alternate index is described in "Alternate Index Maintenance" on page 14.

SCHBFR (search buffer) enables you to locate buffers that contain the contents of adjacent control intervals.

MRKBFR (mark buffer) enables you to mark a buffer for output. You can modify the contents of the buffer with or without having exclusive control of it, and, with WRTBFR, you can cause the modified buffer to be written without issuing a PUT for update.

MRKBFR also enables you to release a buffer from exclusive control or shared status if it was under exclusive control or in shared status. If the buffer was marked to be written, it is still marked to be written. The placeholder associated with the RPL that defines MRKBFR is kept for further use by the RPL, but it no longer holds a place (positioning is invalidated).

For the ICF catalog, MRKBFR allows you to invalidate a buffer. When a buffer is marked "invalid," it is identified as a buffer that VSAM must refresh before your program can use the buffer's contents.

Improved control interval access ACB MACRF=(ICI,...) and user buffering ACB MACRF=(UBF,...) are not allowed with shared resources (that is, where ACB MACRF= ({LSR|GSR},...).

SCHBFR: Locating an RBA in a Buffer Pool

When a resource pool is built, the buffers in each buffer pool are numbered from 1 through the number of buffers in each buffer pool. At a given time, several buffers in a buffer pool may hold the contents of control intervals for a particular data set. These buffers may or may not contain RBAs of interest to your processing program. The SHCBFR macro enables you to find out. You specify in the ARG operand of the RPL that defines SCHBFR the address of an 8-byte field that contains the first and last control interval RBAs of the range you are interested in.

The buffer pool to be searched is the one used by the data component defined by the ACB to which your RPL is related. If the ACB names a path, VSAM searches the buffer pool used by the data component of the alternate index. (If the path is defined over a base cluster alone, VSAM searches the buffer pool used by the data component of the base cluster.) VSAM begins its search at the buffer you specify and continues until it finds a buffer that contains an RBA in the range or until the highest numbered buffer is searched.

You should begin a search at the first buffer in the buffer pool. If a buffer satisfies the search, VSAM returns its address (OPTCD=LOC) or its contents (OPTCD=MVE) in the work area whose address is specified in the AREA operand of the RPL and returns its number in register 0. To find the next buffer that contains an RBA in the range, issue SCHBFR again and specify the number of the next buffer after the first one that satisfied the search. You continue until VSAM indicates that it found no buffer that contains an RBA in the range or until you reach the end of the pool.

Finding a buffer that contains a desired RBA does not get you exclusive control of the buffer. You may get exclusive control only by issuing GET for update. SCHBFR does not return the location or the contents of a buffer that is already under the exclusive control of another request.

MRKBFR: Marking a Buffer of Output

You locate a buffer that contains the RBA you are interested in by issuing a SCHBFR macro, a read-only GET, or a GET for update. When you issue GET for update, you get exclusive control of the buffer. Whether you have exclusive control or not, you can mark the buffer for output by issuing the MRKBFR (mark buffer) macro with MARK=OUT and then change the buffer's contents. Without exclusive control, you should not change the control information in the CIDF or RDFs (do not change the record lengths).

MRKBFR, MARK=OUT, indicates that the buffer's contents are modified. You must modify the contents of the buffer itself—not a copy. Consequently, when you issue SCHBFR or GET to locate the buffer, you must specify RPL OPTCD=LOC. (If you use OPTCD=MVE, you get a copy of the buffer but do not learn its location.) The buffer is written when a WRTBFR is issued or when VSAM is forced to write a buffer to satisfy a GET request.

If you are sharing a buffer or have exclusive control of it, you can release it from shared status or exclusive control with MRKBFR, MARK=RLS. If the buffer was marked for output, MRKBFR with MARK=RLS does not nullify it; the buffer is eventually written. Sequential positioning is lost. MRKBFR with MARK=RLS is similar to the ENDREQ macro.

SUMMARY OF RESTRICTIONS FOR SHARED RESOURCES

Restrictions for using the LSR and GSR options are:

- Empty data sets cannot be processed (that is, loaded).

- Improved control interval access cannot be used (ACB MACRF=ICI).
- Control blocks in common (CBIC) cannot be used.
- User buffering is not allowed (ACB MACRF=UBF).
- Writes for data sets with SHAREOPTIONS 4 cannot be deferred (ACB MACRF=DFR).
- Request parameter lists for MRKBFR, SCHBFR, and WRTBFR cannot be chained (the NXTRPL operand of the RPL macro is ignored).
- For sequential access, positioning at the beginning of a data set must be explicit: with a POINT macro or a direct GET macro with RPL OPTCD=NSP.
- Temporary CLOSE and ENDREQ do not cause buffers to be written if MACRF=DFR was specified in the associated ACB.
- With GSR, an ACB and all related RPLs, EXLSTs, data areas, and ECBs must be stored in the common area of virtual storage with protection key 0 to 7; all VSAM requests related to the global resource pool may be issued only by a program in supervisor state with protection key 0 to 7 (the same as that of the resource pool).
- Checkpoints cannot be taken for data sets whose resources are shared in a global resource pool. When a program in an address space that opened a data set whose ACB has MACRF=GSR issues the CHKPT macro, 8 is returned in register 15. If a program in another address space issues the CHKPT macro, the checkpoint is taken, but only for data sets that are not using the global resource pool.

  Checkpoint/restart can be used with data sets whose resources are shared in a local resource pool, but the restart program does not reposition for processing at the point where the checkpoint occurred—processing is restarted at a data set's highest used RBA. For information about restarting the processing of VSAM data sets, see _Checkpoint/Restart_.
- Buffer Retention because of Physical Error

  If a physical I/O error is encountered while writing a control interval to the direct access device, the buffer remains in the resource pool. The write-required flag (BUFCMW) and associated mod bits (BUFCMOBT) are turned off, and the BUFC is flagged in error (BUFCER2=ON). The buffer is not replaced in the pool, and buffer writing is not attempted. To release this buffer for reuse, a WRTBFR macro with TYPE=DS can be issued or the data set can be closed (CLOSE issues the WRTBUFR macro).

PROCESSING THE INDEX OF A KEY-SEQUENCED DATA SET

You can gain access to the index of a key-sequenced data set in one of two ways:

- By opening the cluster and using the GETIX and PUTIX macros
- By opening the index component alone and using the macros for normal data processing (GET, PUT, and so forth)

Processing the index component alone is identical to processing an entry-sequenced data set; an index itself has no index and thus cannot be processed by keyed access. "Processing VSAM Data Sets" on page 15 tells tells how to gain access to data records.

Processing an alternate index is identical to processing a key-sequenced data set; it too is described in "Processing VSAM Data Sets" on page 15.

This section gives the format of VSAM index records, explains the meaning and use of the information in them, and tells how to gain access to them by way of GETIX and PUTIX. You should not attempt to duplicate or substitute for the index processing that VSAM does during normal access to data records.

THE FORMAT OF AN INDEX RECORD

Index records are stored in control intervals in the same way as data records, except that only one index record is stored in a control interval, and there is no free space between the record and the control information. Consequently, there is only one RDF, that contains the flag X'00' and the length of the record (a number equal to the length of the control interval minus 7). The CIDF also contains the length of the record (the displacement from the beginning of the control interval to the control information); its second number is 0 (no free space). The contents of the RDF and CIDF are the same for every used control interval in an index. The control interval after the last-used control interval has a CIDF filled with 0's, and is used to represent the software end-of-file (SEOF).

Index control intervals are not grouped into control areas as are data control intervals. When a new index record is required, it is stored in a new control interval at the end of the index data set. As a result, the records of one index level are not segregated from the records of another level, except when the sequence set is separate from the index set. Each index record is identified with respect to the level to which it belongs by a field in the index header (see "The Header Portion" on page 105).

When an index record is replicated on a track, each copy of the record is identical to the other copies. Replication has no effect on the contents of records.

Figure 20 sketches the parts of an index record.

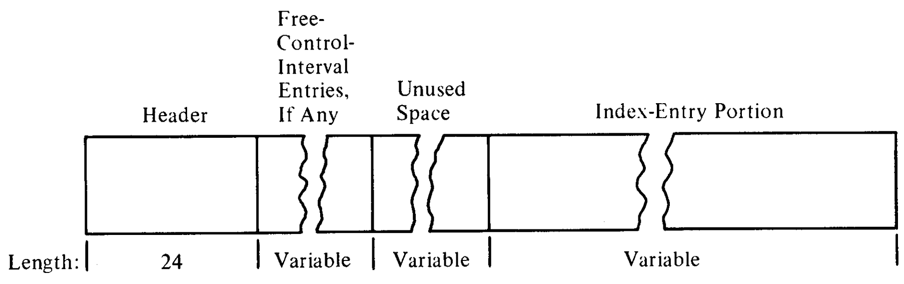


Figure 20. General Format of an Index Record

The first part is a header that contains control information about the record.

Next, a sequence-set index record (a record in the lowest level of the index) may contain a set of free control interval entries used to locate free control intervals in the control area governed by the index record. An index-set index record (a record in a higher level of the index) has no entries of this type.

Next, an index record may have unused space.

The last part is a set of index entries used to locate, for an index-set record, control intervals in the next lower level of the index, or, for a sequence-set record, used control intervals in the control area governed by the index record.

The Header Portion

The first 24 bytes of an index record is the header, which gives control information about the index record. Figure 21 shows its format. All lengths and displacements are in bytes. The discussions in the following two sections amplify the meaning and use of some of the fields in the header.

Field	Offset	Length	Description
IXHLL	0(0)	2	Record length. The length of the indexed record is equal to the length of the control interval minus 7.
IXHFLPLN	2(2)	1	Index-entry control-information length. This is the length of the last three of the four fields in an index entry. (The length of the first field is variable.) The length of the control information is 3, 4, or 5 bytes.
IXHPTLS	3(3)	1	Vertical-pointer-length indicator. The fourth field in an index entry is a vertical pointer to a control interval. In an index-set record, the pointer is a binary number that designates a control interval in the index. The number is calculated by dividing the RBA of the control interval by the length of the control interval. To allow for a possibly large index, the pointer is always 3 bytes. In a sequence-set record, the pointer is a binary number, beginning at 0, that designates a control interval in the control area governed by the sequence-set record. A free-control-interval entry is nothing more than a vertical pointer. There are as many index entries and free control interval entries in a sequence-set record as there are control intervals in a control area. Depending on the number of control intervals in a control area, the pointer is 1, 2, or 3 bytes. An IXHPTLS value of X'01' indicates a 1-byte pointer; X'03' indicates a 2-byte pointer; X'07' indicates a 3-byte pointer.
IXHBRBA	4(4)	4	Base RBA. In an index-set record, this is the beginning RBA of the index. Its value is 0. The RBA of a control interval in the index is calculated by multiplying index control interval length times the vertical pointer and adding the result to the base RBA. In a sequence-set record, this is the RBA of the control area governed by the record. The RBA of a control interval in the control area is calculated by multiplying data control interval length times the vertical pointer and adding the result to the base RBA. Thus, the first control interval in a control area has the same RBA as the control area (length times 0, plus base RBA, equals base RBA).

Figure 21 (Part 1 of 2). Format of the Header of an Index Record

Field	Offset	Length	Description
IXHHP	8(8)	4	Horizontal-pointer RBA. This is the RBA of the next index record in the same level as this record. The next index record contains keys next in ascending sequence after the keys in this record.
	12(C)	4	(Reserved.)
IXHLV	16(10)	1	Level number. The sequence-set is the first level of an index, and each of its records has an IXHLV of 1. Records in the next higher level have a 2, and so on.
	17(11)	1	(Reserved.)
IXHFSO	18(12)	2	Displacement to the unused space in the record. In an index-set record, this is the length of the header (24)—there are no free control interval entries. In a sequence-set record, the displacement is equal to 24, plus the length of free control interval entries, if any.
IXHLEO	20(14)	2	Displacement to the control information in the last index entry.The last (leftmost) index entry contains the highest key in the index record. In a search, if the search-argument key is greater than the highest key in the preceding index record but less than or equal to the highest key in this index record, then this index record governs either the index records in the next lower level that have the range of the search-argument key or the control area in which a data record having the search-argument key is stored.
IXHSEO	22(16)	2	Displacement to the control information in the last (leftmost) index entry in the first (rightmost) section. Index entries are divided into sections to facilitate a quick search. Individual entries are not examined until the right section is located.

Figure 21 (Part 2 of 2). Format of the Header of an Index Record

The Free Control Interval Entry Portion

If the control area governed by a sequence-set record has free control intervals, the sequence-set record has entries pointing to those free control intervals. Each entry is 1, 2, or 3 bytes long (indicated by IXHPTLS in the header: the same length as the pointers in the index entries).

The entries come immediately after the header. They are used from right to left. The rightmost entry is immediately before the unused space (whose displacement is given in IXHFSO in the header). When a free control interval gets used, its free entry is converted to zero, the space becomes part of the unused space, and a new index entry is created in the position determined by ascending key sequence.

Thus, the free control interval entry portion contracts to the left, and the index-entry portion expands to the left. When all the free control intervals in a control area have been used, the sequence-set record governing the control area no longer has free control interval entries, and the number of index entries equals the number of control intervals in the control area. Note that if the index control interval size was specified with too small a value, it is possible for the unused space to be used up for index entries before all the free control intervals have been used, resulting in control intervals within a data control area that cannot be utilized.

The Index-Entry Portion

The index-entry portion of an index record takes up all of the record that is left over after the header, the free control interval entries, if any, and the unused space.

Figure 22 shows the format of the index-entry portion of an index record. To improve search speed, index entries are grouped into sections, of which there are approximately as many as the square root of the number of entries. For example, if there are 100 index entries in an index record, they are grouped into 10 sections of 10 entries each. (The number of sections does not change, even though the number of index entries increases as free control intervals get used.)

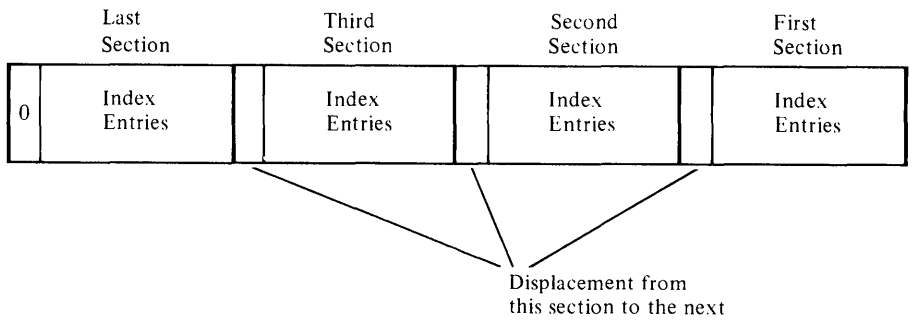


Figure 22. Format of the Index-Entry Portion of an Index Record

The sections, and the entries within a section, are arranged from right to left. IXHLEO in the header gives the displacement from the beginning of the index record to the control information in the leftmost index entry. IXHSEO gives the displacement to the control information in the leftmost index entry in the rightmost section. You calculate the displacement of the control information of the rightmost index entry in the index record (the entry with the lowest key) by subtracting IXHFLPLN from IXHLL in the header (the length of the control information in an index entry from the length of the record).

Each section is preceded by a 2-byte field that gives the displacement from the control information in the leftmost index entry in the section to the control information in the leftmost index entry in the next section (to the left). The last (leftmost) section's 2-byte field contains 0's.

Figure 23 on page 108 gives the format of an index entry. Index entries are variable in length within an index record, because VSAM compresses keys. That is, it eliminates redundant or unnecessary characters from the front and back of a key to save space. The number of characters that can be eliminated from a key depends on the relationship between that key and the preceding and following keys.

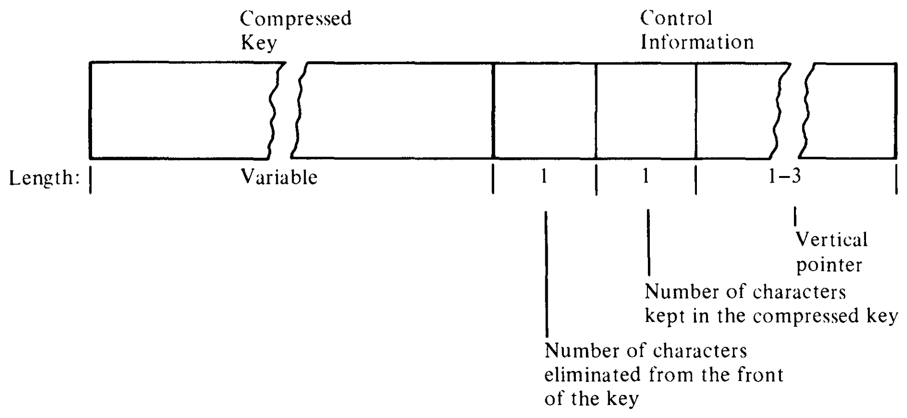

Compressed Key
Control Information
Length:
Variable
1
1
1-3
Vertical pointer
Number of characters kept in the compressed key
Number of characters eliminated from the front of the key

Figure 23. Format of an Index Entry

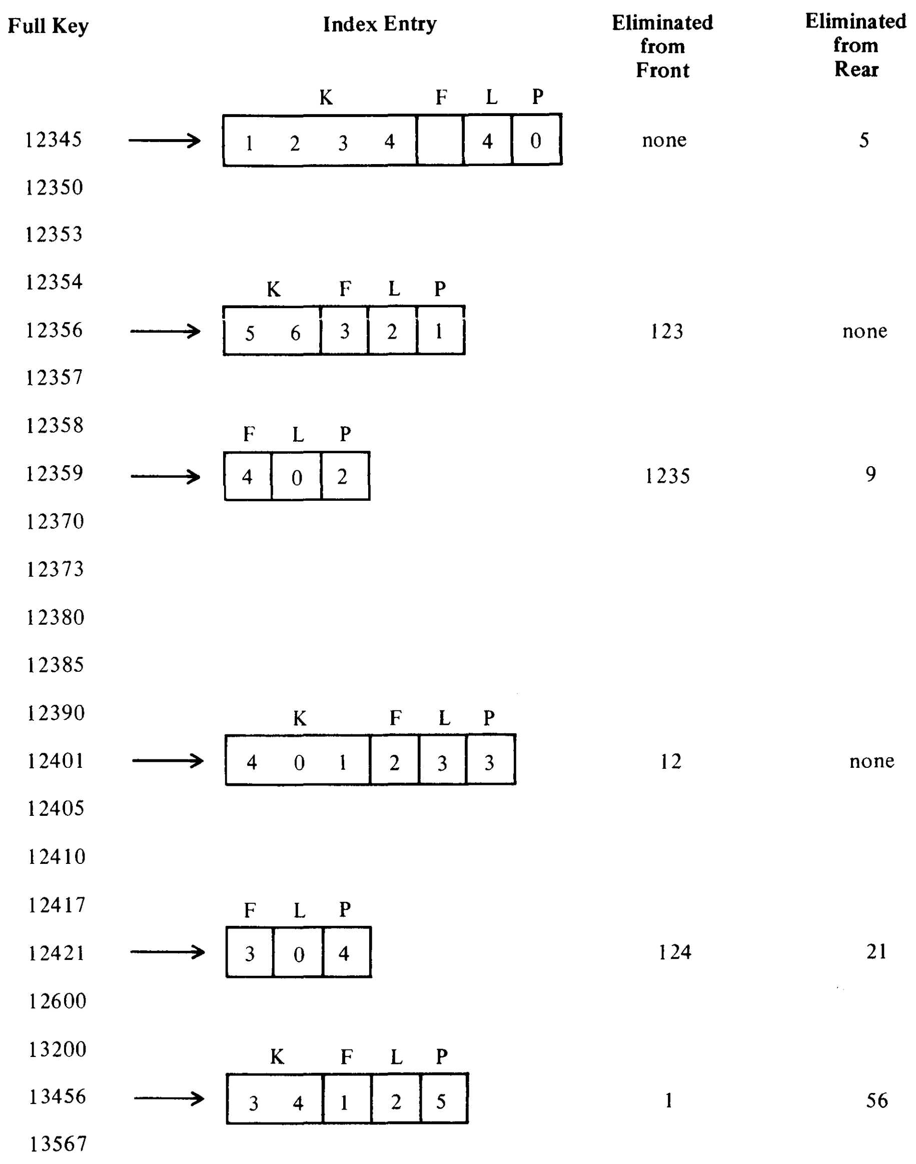
For front compression, VSAM compares a key in the index with the preceding key in the index and eliminates from the key those leading characters that are the same as the leading characters in the preceding key. For example, if key 12 356 follows key 12 345, the characters 123 are eliminated from 12 356 because they are equal to the first three characters in the preceding key. The lowest key in an index record has no front compression; there is no preceding key in the index record.

There is an exception for the highest key in a section. For front compression, it is compared with the highest key in the preceding section, rather than with the preceding key. The highest key in the rightmost section of an index record has no front compression; there is no preceding section in the index record.

For rear compression, VSAM compares a key in the index with the following key in the data and eliminates from the key those characters to the right of the first character that is unequal to the corresponding character in the following key. For example, if the key 12 345 (in the index) precedes key 12 356 (in the data), the character 5 is eliminated from 12 345 because the fourth character in the two keys is the first unequal pair.

The first of the control information fields gives the number of characters eliminated from the front of the key, and the second field gives the number of characters that remain. When the sum of these two numbers is subtracted from the full key length (available from the catalog when the index is opened), the result is the number of characters eliminated from the rear. The third field indicates the control interval that contains a record with the key.

The example in Figure 24 on page 109 gives a list of full keys and shows the contents of the index entries corresponding to the keys that get into the index (the highest key in each data control interval). A sequence-set record is assumed, with vertical pointers 1 byte long. The index entries shown in the figure from top to bottom are arranged from right to left in the assumed index record.



Legend:

⟶ = the key goes into the index

K = characters left in key after compression

F = number of characters eliminated from the front

L = number of characters left in key after compression

P = vertical pointer

Figure 24. Example of Key Compression

Key 12 345 has no front compression because it is the first key in the index record. Key 12 356 has no rear compression because, in the comparison between 12 356 and 12 357, there are no characters following 6, which is the first character that is unequal to the corresponding character in the following key.

You can always figure out what characters have been eliminated from the front of a key; you cannot figure out the ones eliminated from the rear. Rear compression, in effect, establishes the key in the entry as a boundary value instead of an exact high key. That is, an entry does not give the exact value of the highest key in a control interval, but gives only enough of the key to distinguish it from the lowest key in the next control interval. In Figure 24 on page 109, for example, the last three index keys, after rear compression, are 12 401, 124, and 134. Data records with key fields between 12 402 and 124FF are associated with index key 124; data records with key fields between 12 500 and 134FF are associated with index key 134.

If the data record with the highest key in a control interval is deleted, the index entry's key value does not change—does not change, in fact, even if all the data records are deleted.

The last index entry in an index level indicates the highest possible key value. The convention for expressing this value is to give none of its characters and indicate that no characters have been eliminated from the front. The last index entry in the last record in the sequence set looks like this:

F	L	P
0	0	x

where x is a binary number from 0 to 255, assuming a 1-byte pointer.

In a search, the two 0's signify the highest possible key value in this way: The fact that 0 characters have been eliminated from the front implies that the first character in the key is greater than the first character in the preceding key. A length of 0 indicates that no characters need be compared to determine whether the search is successful; that is, when a search encounters the last index entry, a hit has been made.

INDEX ENTRIES FOR A SPANNED RECORD: In a key-sequenced data set, there is an index entry for each control interval that contains a segment of a spanned record. All the index entries for a spanned record are grouped together in the same section. They are ordered from right to left according to the sequence of segments (first, second, third, and so on).

Only the last (leftmost) index entry for a spanned record contains the key of the record. The key is compressed according to the rules described above. All the other index entries for the record look like this:

F	L	P
y	0	x

where y is a binary number equal to the length of the key (y indicates that the entire key has been "eliminated from the front"; L indicates that 0 characters remain) and x identifies the control interval that contains the segment.

When a spanned record is deleted, all the index entries except the last (the one that contains the key) are deleted and free control interval entries are created to identify the freed control intervals.

HOW TO GAIN ACCESS TO A KEY-SEQUENCED DATA SET'S INDEX

You can gain access to the index of a key-sequenced data set in one of two ways:

- By opening the cluster and using the GETIX and PUTIX macros
- By opening the index component alone and using the macros for normal data processing (GET, PUT, and so forth)

Opening the Cluster

To process the index of a key-sequenced data set with GETIX and PUTIX, you must open the cluster with ACB MACRF=(CNV,...) specified. CNV provides for control interval access, which you use to gain access to the index component.

GETIX and PUTIX are coded the same way as the other request macros: Their only operand is RPL=address.

Access by way of GETIX and PUTIX is direct by control interval: VSAM requires RPL OPTCD=(CNV,DIR). The search argument for GETIX is the RBA of a control interval. The increment from the RBA of one control interval to that of the next is control interval size for the index.

GETIX can be issued either for update or not for update. VSAM recognizes OPTCD=NUP or UPD but interprets OPTCD=NSP as NUP.

The contents of a control interval cannot be inserted by way of PUTIX: VSAM requires OPTCD=UPD. The contents must previously have been retrieved for update by way of GETIX.

RPL OPTCD=MVE or LOC may be specified for GETIX, but only OPTCD=MVE is valid for PUTIX. If you retrieve with OPTCD=LOC, you must change OPTCD to MVE to store. With OPTCD=MVE, AREALEN must be at least index control interval size.

Beyond these restrictions, access to an index by way of GETIX and PUTIX follows the rules under "Processing Control Intervals" on page 81.

ERROR RETURN CODES: The error return codes for GETIX and PUTIX are the same as for GET and PUT, which are given in VSAM Reference.

Opening the Index Component of the Cluster

You can gain addressed or control interval access to the index component of a key-sequenced cluster by opening the index component alone and using the request macros for normal data processing. To open the index component alone, specify:

DSNAME=indexcomponentname

in the DD statement identified in the ACB (or GENCB) macro.

You can gain access to index records with addressed access and to index control intervals with control interval access. The use of these two types of access for processing an index is identical in every respect with their use for processing a data component.

BUILDING PARAMETER LISTS FOR GENCB, MODCB, SHOWCB, AND TESTCB

The standard forms of GENCB, MODCB, SHOWCB, and TESTCB build a parameter list, put its address in register 1, and pass control to a VSAM routine to generate, modify, display, or test an access method control block, exit list, or request parameter list. Other forms of the macros only build the parameter list (list forms) or only pass control to VSAM (execute forms). All these forms are described in VSAM Reference.

You can avoid using a macro to build the parameter list by building it yourself and issuing the execute form of the macro to pass control to VSAM. This chapter explains how to build the parameter lists for GENCB, MODCB, SHOWCB, and TESTCB. The rules for combinations of codes in a parameter list are the same as the rules for combinations of operands in a macro, which are given in VSAM Reference.

You can avoid issuing the execute form of the macro by coding the linkage instructions that pass control directly to the VSAM control block manipulation routine. Before passing control, you must build the parameter list yourself.

THE FORMAT OF THE PARAMETER LISTS

A parameter list for GENCB, MODCB, SHOWCB, or TESTCB is a list of fullword addresses. The first address points to a header entry that identifies the type of request and type of control block and gives other general information about the request. Each of the rest of the addresses in the parameter list points to an element entry that identifies the information you want to generate, modify, display, or test.

The fullwords in the parameter list must be contiguous, and the last one must have a 1 in its first bit. The header entry and each element entry may be separate from each other.

Figure 25 on page 113 gives the formats of the header and element entries for the four request types.

BUILDING HEADER AND ELEMENT ENTRIES

Five assembler macros are provided for building entries. IDAGENC, IDAMODC, IDASHOW, and IDATEST help you build a header entry for generation, modification, display, or test. IDAELEM helps you build an element entry.

[label]	IDAGENC	[DSECT={YES|NO}]
[label]	IDAMODC	[DSECT={YES|NO}]
[label]	IDASHOW	[DSECT={YES|NO}]
[label]	IDATEST	[DSECT={YES|NO}]
[label]	IDAELEM	[DSECT={YES|NO}]

DSECT={YES|NO}
Indicates whether a DSECT statement is to be generated. If you intend to build entries in a continuous area, you could have only the first of the macros generate a DSECT statement and use a single register to address the whole area.

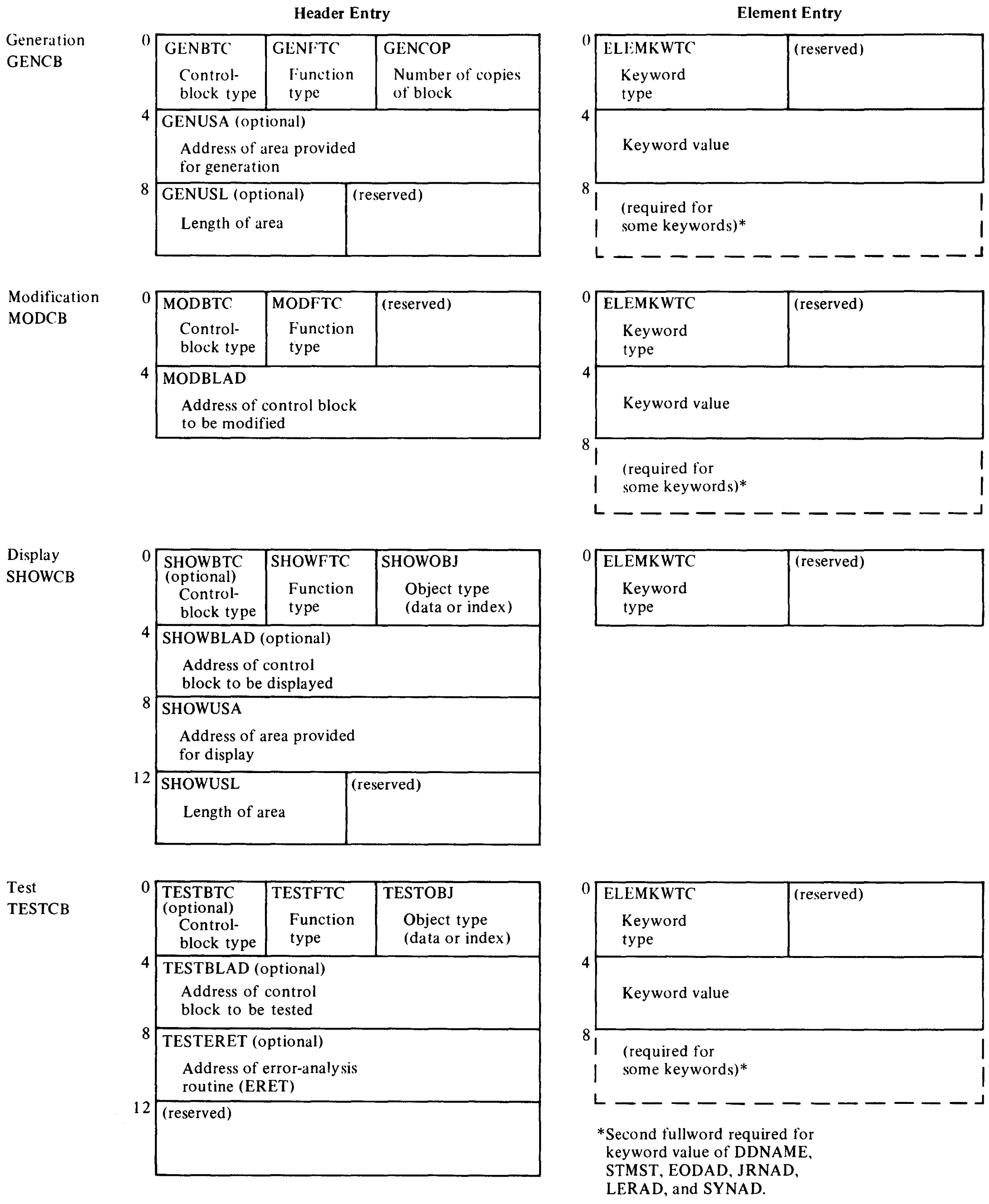

Header Entry
Element Entry
Generation GENCB
0 GENBTC Control-block type
GENFTC Function type
GENCOP Number of copies of block
4 GENUSA (optional) Address of area provided for generation
8 GENUSL (optional) Length of area
(reserved)
0 ELEMKWTC Keyword type
(reserved)
4 Keyword value
8 (required for some keywords)*
Modification MODCB
0 MODBTC Control-block type
MODFTC Function type
(reserved)
4 MODBLAD Address of control block to be modified
0 ELEMKWTC Keyword type
(reserved)
4 Keyword value
8 (required for some keywords)*
Display SHOWCB
0 SHOWBTC (optional) Control-block type
SHOWFTC Function type
SHOWOBJ Object type (data or index)
4 SHOWBLAD (optional) Address of control block to be displayed
8 SHOWUSA Address of area provided for display
12 SHOWUSL Length of area
(reserved)
0 ELEMKWTC Keyword type
(reserved)
Test TESTCB
0 TESTBTC (optional) Control-block type
TESTFTC Function type
TESTOBJ Object type (data or index)
4 TESTBLAD (optional) Address of control block to be tested
8 TESTERET (optional) Address of error-analysis routine (ERET)
12 (reserved)
0 ELEMKWTC Keyword type
(reserved)
4 Keyword value
8 (required for some keywords)*
*Second fullword required for keyword value of DDNAME, STMST, EODAD, JRNAD, LERAD, and SYNAD.

Figure 25. Format of Header and Element Entries for GENCB, MODCB, SHOWCB, and TESTCB Parameter Lists

These macros generate labeled DS statements that give the layout of an entry and EQU statements that pair a label with a numeric code. You can symbolically encode an entry with a series of move instructions. The macros are self-documenting; inspect a listing of their expansions and you can see what labels to code in your move instructions. (You can list the macros as they appear in the macro library.)

To generate an exit list with LERAD and SYNAD exits, you could code a GENCB of the standard form:

```
GENCB   BLK=EXLST,LERAD=(LOGERR,L),SYNAD=PHYSERR
```

The following example shows how to achieve the same effect by building the parameter list and entries yourself and issuing a GENCB of the execute form.

```
          LA      5,NTRYAREA        Set up base register for the
                                    entries.

          USING   5,GENC            GENC is the first label in the
                                    work area.
```

Build the list of addresses that point to the entries.

```
          ST      5,PLIST           Address of the header entry.

          LA      6,GENLEN(,5)      Address of the first element
                                    entry. GENLEN is equated to the
                                    length of a header element for
                                    generation.

          ST      6,PLIST+4

          LA      6,ELEMLLEN(,6)    Address of the second element
                                    entry. ELEMLLEN is equated to
                                    the length of an element entry
                                    for an exit list.

          ST     6,PLIST+8

          OI     PLIST+8,X'80'      End-of-list indicator.
```

Build the header entry.

```
          MVI    GENBTC,GENXLST     Indicate the block type—exit
                                    list.

          MVI    GENFTC,GENFTYP     Indicate the function
                                    type—generation.

          MVI    GENCOP+1,X'01'     Indicate the number of copies
                                    of the exit list to be
                                    generated.

          MVI    ELEMKWTC+1,        Indicate the keyword
                 ELEMLEAD           type—LERAD.


          LA     6,LOGERR           Address of the name of the
                                    logical error analysis module.

          ST     6,ELEMPTR

          MVI    ELEMXFLG,          Indicate the presence of an
                 ELEMXL+ELEMXADR    address ELEMPTR and that the
                                    exit routine is to be loaded.
```

Build the second element entry.

```
          LA      5,ELEMLLEN(,5)    Align the DSECT with the
                                    second element entry. ELEMLLEN
                                    is equated to the length of an
                                    element entry for an exit list.

          MVI     ELEMKWTC+1,       Indicate the keyword
                  ELEMSYAD          type——SYNAD.

          LA      6,PHYSERR         Address of the entry point of
                                    the physical error analysis
                                    routine.

          ST      6,ELEMPTR

          MVI     ELEMXFLG,         Indicate the presence of an
                  ELEMXADR          address in ELEMPTR.
```

Pass control to VSAM.

```
          GENCB   MF=(E,PLIST)

          LTR     15,115            Generation successful

          BNZ     CHECKO            No.
          .
          .
          .
CHECKO    ABEND   1,DUMP            Register 0 indicates the error.
```

Physical error analysis exit routine.

```
PHYSERR . . .
```

Work areas and constants.

```
LOGERR    DC      CL8'LEMOD'        Name of the
                                    logical error analysis
                                    module to be loaded.

PLIST     DC      3F'0'             List of entry addresses.
                                    3 addresses are required:
                                    1 for the header and 2 for
                                    the elements (1 for LERAD and
                                    1 for SYNAD).

NTRYAREA  DC      9F'0'             Work area for header and
                                    element entries. The header
                                    for GENCB is 3 fullwords,
                                    and so are the LERAD
                                    and SYNAD elements.
```

DSECT with labels for the header and element entries.

```
          IDAGENC                   Header entry. A DSECT
                                    statement is generated, and
                                    register 5 is used to
                                    address NTRYAREA with these
                                    labels.

          IDAELEM DSECT=NO          Element entry. Element labels
                                    are part of the same DSECT as
                                    the header labels.
```

PASSING CONTROL DIRECTLY TO VSAM

You can avoid using the execute form of GENCB, MODCB, SHOWCB, and TESTCB by building your own linkage instructions. You first build a parameter list, as described in the previous section, and put its address in register 1. Then you pass control to VSAM using the following instructions:

```
L        15,16          Put the address of the CVT into register
                        15.

L        15,256(,15)    Put the address of the AMCBS control block
                        into register 15.

L        15,12(,15)     Put the address of the control block
                        manipulation routine into register 15.

BALR     14,15
           or           Branch to the routine
BAL      14,xx(,15)
```

The BALR 14,15 instruction is used when the specific function (GENCB, MODCB, SHOWCB, or TESTCB) is not known, or when the control block type (ACB, EXLST, or RPL) is not known. The user-built parameter list contains the function code and control block type code.

The BAL 14,xx(,15) instruction is used to increase your program's performance. The "xx" is a decimal value that identifies a function (GENCB, MODCB, SHOWCB, or TESTCB) and a control block type (ACB, EXLST, or RPL).

Decimal Value of xx	Function	Control Block
8	GENCB	ACB
12	GENCB	RPL
16	GENCB	EXLST
20	Reserved	
24	MODCB	ACB
28	MODCB	RPL
32	MODCB	EXLST
36	Reserved	
40	SHOWCB	ACB
44	SHOWCB	RPL
48	SHOWCB	EXLST
52	Reserved	
56	TESTCB	ACB
60	TESTCB	RPL
64	TESTCB	EXLST
68	Reserved	
72	SHOWCB or TESTCB	Block length keywords only

Decimal Value of xx	Function	Control Block
76[1]	SHOWCB	RECLEN field of an RPL
80[1]	MODCB	RECLEN field of an RPL

[1] Register 1 points to an RPL when xx is 76 or 80. See the following section for details.

When VSAM returns to your program, register 15 contains a completion code. Register 15 contains a zero value if the task completed successfully. Otherwise, register 15 and register 0 contain codes that identify the reason VSAM could not complete the task. For error code information, see VSAM Reference.

Modifying and Displaying the RECLEN Field of an RPL Directly

You can modify or display the RECLEN field (that is, the record length) of an RPL without issuing a SHOWCB or MODCB macro, and without building a parameter list.

To modify a RPL's RECLEN field, you first put the address of the RPL in register 1, and the value to be set in the RECLEN field in register 0. Next, you code the instructions that put the address of the VSAM control block manipulation routine into register 15, then branch to the routine:

L	15,16	Put the address of the CVT into register 15.
L	15,256(,15)	Put the address of the AMCBS control block into register 15.
L	15,12(,15)	Put the address of the control block manipulation routine into register 15.
BAL	14,80(,15)	Branch to the routine.

When VSAM returns to your program, register 15 contains a completion code. Register 15 contains a zero value if the field was modified correctly. Otherwise, register 15 and register 0 contain codes that identify the reason VSAM could not complete the task. For error code information, see VSAM Reference.

To display the contents of a RPL's RECLEN field, you first put the address of the RPL in register 1. Next, you code the instructions that put the address of the VSAM control block manipulation routine into register 15, and then branch to the routine:

L	15,16	Put the address of the CVT into register 15.
L	15,256(,15)	Put the address of the AMCBS control block into register 15.
L	15,12(,15)	Put the address of the control block manipulation routine into register 15.
BAL	14,76(,15)	Branch to the routine.

When VSAM returns to your program, register 15 contains a completion code. Register 15 contains a zero value if the field is displayed correctly, and register 0 contains the value of the RPL's RECLEN field. When register 15 is not zero, register 15 and register 0 contain codes that identify the reason VSAM could not complete the task. For error code information, see VSAM Reference.

STAGING VSAM DATA SETS ON A KEY OR KEYRANGE BASIS FOR MASS STORAGE SYSTEM (MSS)

Certain MSS applications are characterized by very large data bases for which staging or binding of all the data at OPEN time is not practical. Also, for the purpose of retrieving the data, these applications characteristically require a large number of queries. Thus, cylinder fault mode is equally impractical. Applications of this nature require a capability for selective staging. In other words, what you want is a mechanism to stage necessary data from multiple adjacent or nonadjacent cylinders with a single MSS cartridge load operation. You would thus avoid an inordinate amount of time spent in loading and unloading MSS cartridges for each cylinder required by the processing program.

VSAM provides support for the above MSS environment by means of the stage-by-keyrange function. This function involves two interfaces with three macros. One interface supports prestaging of discretely identified records through the use of two macros: CNVTAD (convert address) and MNTACQ (mount acquire). The other interface supports prestaging of a specified range of records through the use of the ACQRANGE (acquire range) macro.

The use of the above VSAM macros enables you to prestage your data and thus to optimize your use of MSS in the environments discussed above. With prestaging, data extents can be acquired in advance of their use, thus reducing the number of cylinder faults incurred during processing.

The CNVTAD/MNTACQ combination would typically be used when your MSS application needs to refer to a small, noncontiguous subset of data contained within a large data set. The following is a description of a general application using the CNVTAD/MNTACQ interface:

STEP 1
Your application program queues a number of transactions for data stored on the MSS. On the basis of either time or the number of elements on the queue, STEP 2 is initiated.

STEP 2
The transaction for a given data set are processed to extract the keys to be used to access the data set. You supply these keys as input to the CNVTAD macro, which determines the volume and RBA on which each record resides in MSS. The entries for a given volume may then be used by your application as input to the MNTACQ macro. This results in acquiring the data from the corresponding MSS cartridges with a minimum number of cartridge loads.

STEP 3
Your application program will now process transactions for a given data set and volume, and, in general, would not encounter cylinder faults for the data acquired in STEP 2.

The second interface, ACQRANGE, gives you the capability of prestaging a continuous subset of a data set rather than individual cylinders or the entire data set. The continuous portion can be defined by relative record numbers, keys, or RBAs, depending upon the type of VSAM data set you have. The data to be staged may cross volume boundaries, but the volumes must be mounted. To minimize the subset of volumes that must be mounted, the keyrange function of KSDSs may be useful.

The three VSAM macros, CNVTAD, MNTACQ, and ACQRANGE, which support the stage by keyrange function, are described below.

CNVTAD: CONVERT AN ARGUMENT TO AN ADDRESS

The CNVTAD macro is intended to be used in conjunction with the MNTACQ macro in order to achieve prestaging of needed records before they are actually required by your application program. You supply an argument for CNVTAD. The argument depends on the type of your VSAM data set:

KSDS	Full key of the record
ESDS	RBA of the record
RRDS	Relative record number (RRN) of the record

MNTACQ: MOUNT A VOLUME AND ACQUIRE CYLINDERS

The MNTACQ macro is intended to be used in conjunction with the CNVTAD macro to prestage needed records from a single volume in advance of their actual use. When you invoke MNTACQ, you give it a list of RBAs and a volume serial number. MNTACQ causes the volume to be mounted (if it is not already mounted) and stages the cylinders corresponding to the RBAs provided.

ACQRANGE: ACQUIRE A CONTINUOUS RANGE OF RECORDS

ACQRANGE is intended for use in applications slightly different from those of CNVTAD and MNTACQ. It provides a mechanism to stage a continuous portion of your data set rather than individual cylinders or the entire data set. You would typically use ACQRANGE when you are doing processing within a range of keys and when you know the volume serial number of the virtual volume on which your range of keys resides. Input to ACQRANGE is a starting and ending pair of arguments (keys, RBAs, or RRNs) which delineates the subset of data you wish to be staged.

All required volumes must be mounted (via JCL or dynamic allocation) prior to execution of ACQRANGE. If not, your request will be rejected. Mounting must be in parallel, which means you must have at least as many units as virtual volumes.

NON-MSS SUPPORT

As an aid to MSS migration, installation, and testing, VSAM allows you to issue CNVTAD, MNTACQ, and ACQRANGE macros against non-MSS data sets. Such use against nonvirtual DASD will result in register 15 being set to zero; however, the RPLERRCD field in the RPL will be nonzero for ACQRANGE and MNTACQ. For error code information, see _VSAM Reference_.

RESTRICTIONS AND LIMITATIONS

The MSS staging VSAM data sets on a key or keyrange basis function has the following restrictions and limitations you need to be aware of:

- For key processing, key equal and key equal or greater than are supported, but the generic key facility is not supported. If you wish to use generic keys, you must pad your generic key to the full key length.

- The RPL option, OPTCD=WAITX, is not supported.

- Chained RPLs are not supported.

- The data set must be opened in non-ICI (improved control interval) mode.

- User buffers are not supported for LSR and GSR.
- You cannot use these macros against a data set in create mode.
- When used in path processing, alternate indexes are not supported. However, alternate indexes are supported when opened as end-use VSAM objects.
- You cannot use these macros against a key-sequenced data set with an embedded single-level index.
- Data staged by MNTACQ and ACQRANGE is not "bound." That is, it may be destaged prior to use of the data because of the requirements for MSS facilities either by your own program or by another MSS user program.
- VSAM, through its SHAREOPTIONS facilities, allows for concurrent updating of a data set by two or more tasks or regions with varying degrees of support. With these facilities, operations against a key-sequenced data set by one task cause copies of the index in use by other tasks to remain valid and useful. Keeping the index current allows data records to be retrieved and manipulated with data integrity.

  With prestaging, however, updating using SHAREOPTIONS may cause problems. Because the index is used well in advance of the actual reference to the data, concurrent insertions by other users can render the target addresses derived by ACQRANGE or CNVTAD obsolete and erroneous before or after the data records have been prestaged.

  This exposure increases when DISP=SHR is specified, because this may allow alterations concurrent with prestaging.

  You should remember that DISP and SHAREOPTIONS interact to determine the nature of data set sharing for VSAM. However, independent of data set sharing, prestaging may occur. Prestaging as described in this chapter has no facility for detecting or enforcing various sharing situations. If your installation allows prestaging concurrent with data set alteration, you can expect unpredictable errors as well as fallacious staging and perhaps a degree of "MSS thrashing," an effect in direct opposition to the reason for prestaging.
- The MNTACQ and ACQRANGE macros obtain virtual storage dynamically via the GETMAIN macro. If the request for storage fails, a logical error code is set in the RPL.

ALTERNATE INDEXES

Although these macros do not specifically support alternate indexes and paths, support for alternate indexes can be accomplished in the following manner: The alternate index is a key-sequenced data set and can be processed as such. CNVTAD, MNTACQ, and ACQRANGE macros can be used to stage the data portion of the alternate index. After you have staged the alternate index, the RBA and key pointers can be extracted from the data component of the alternate index. These pointers can be used as input to CNVTAD, MNTACQ, and ACQRANGE for the base data.

CHAPTER 7. JOB CONTROL LANGUAGE

This chapter describes the job control language, an optional method for connecting a data set and the program that is to use it, how to code the VSAM JCL parameter (AMP), and how to identify user catalogs for jobs or job steps (JOBCAT or STEPCAT).

A necessary link between a processing program and the data set to be processed is the data set name. When JCL is used, the access method control block gives the name of the DD statement so that the OPEN macro can make the connection between the program and the data set named in the DD statement, connecting the program and data. Figure 26 on page 122 shows the relationship between JCL and the VSAM macro. JCL is used to catalog, uncatalog, and delete non-VSAM data sets in a catalog. You can invoke dynamic allocation of auxiliary storage. Although this publication does not describe the dynamic allocation function, you can dynamically allocate VSAM data sets and user catalogs. When you define a VSAM data set or catalog, no DD statement is required if access method services can dynamically allocate the volume. (For an explanation of dynamic allocation, see the JCL manual.)

The catalog contains most of the information required by VSAM to process a data set, so VSAM requires minimal information from JCL. Data set name and disposition describe the data set. A key-sequenced data set is described with a single DD statement.

To allow only one job step to access the data set, specify DISP=OLD. If the data set's share options allow the type of sharing your program anticipates, you can specify DISP=SHR in the DD statements of separate jobs to enable two or more job steps to share a data set. For more details on sharing data sets, see the chapter, "Chapter 5. Sharing a VSAM Data Set" on page 69.

HOW TO CODE JCL

VSAM data sets are created using access method services and are cataloged in a catalog. To identify a VSAM data set through JCL, specify a DD statement of the form:

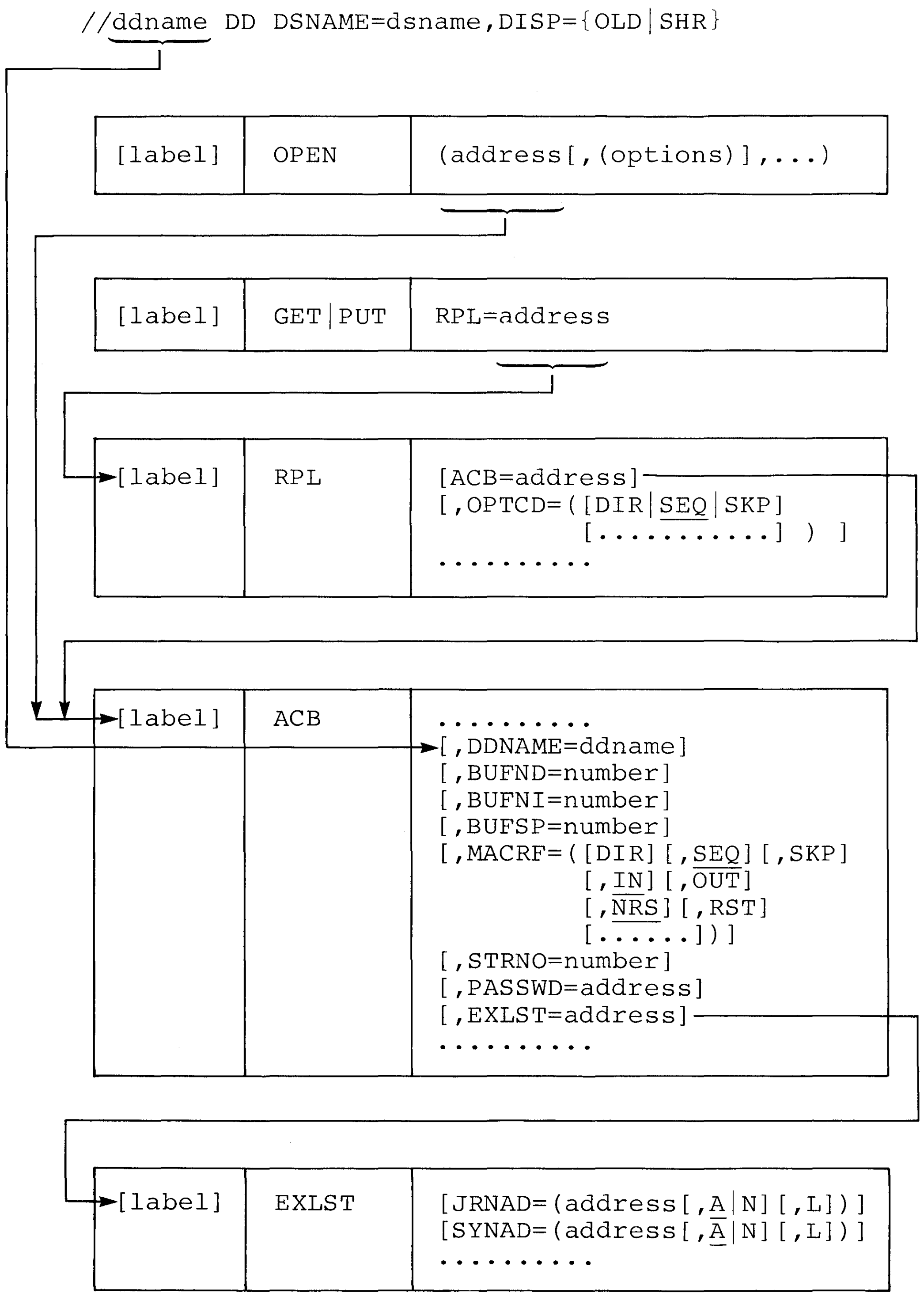
```
  //ddname DD DSNAME=dsname,DISP={OLD|SHR}
```

Optionally, the AMP parameter may be specified to modify how the program operates.

The DSNAME parameter specifies the name of the data set you are processing. Each VSAM data set is defined as a cluster of one or two components: A key-sequenced data set is made up of a data component and an index component; and an entry-sequenced and a relative record data set are made up of only a data component. If you need to process a component separately, you may specify the component's name in the DSNAME parameter.

If a data set has been defined in a user catalog, it is also necessary to identify the user catalog by means of either a JOBCAT or a STEPCAT DD statement.

When separate DD statements are used and one or more subtasks will perform output processing, the DD statements must specify DISP=SHR. With separate DD statements, several subtasks can share a data set under the same rules as for cross-region sharing.



```
//ddname DD DSNAME=dsname,DISP={OLD|SHR}

[label]   OPEN      (address[,(options)],...)

[label]   GET|PUT   RPL=address

[label]   RPL       [ACB=address]
                    [,OPTCD=([DIR|SEQ|SKP]
                              [...........] ) ]
                    ..........

[label]   ACB       ..........
                    [,DDNAME=ddname]
                    [,BUFND=number]
                    [,BUFNI=number]
                    [,BUFSP=number]
                    [,MACRF=([DIR][,SEQ][,SKP]
                              [,IN][,OUT]
                              [,NRS][,RST]
                              [......])]
                    [,STRNO=number]
                    [,PASSWD=address]
                    [,EXLST=address]
                    ..........

[label]   EXLST     [JRNAD=(address[,A|N][,L])]
                    [SYNAD=(address[,A|N][,L])]
                    ..........
```

Figure 26. JCL to VSAM Macro Relationship

Because the operating system allows DD parameters and subparameters that do not apply to a VSAM data set, you should use the DD parameters and subparameters that have a clear meaning when used with VSAM. Figure 27 shows the DD parameters and subparameters that can be used with VSAM and indicates their meaning for a VSAM data set. DD parameters and subparameters not shown in Figure 27 should be avoided.

With a data set defined in a VSAM catalog, you may mount some, but not all, of the volumes on which a data set is stored (called subset mount), you specify the DD parameters VOLUME and UNIT.

Note: This may cause excessive processing time because of mount and demount activities directed to those volumes.Specifying those parameters to open a DCB (to be processed through the ISAM interface program) prevents a reference to the VSAM catalog and requires that you use the AMP subparameter AMP='AMORG' to identify the data set as a VSAM data set. If you specify VOLUME and UNIT to open a VSAM ACB, AMORG is not required.

JCL PARAMETERS USED WITH VSAM

Parameter	Subparameter	Comment
AMP		See "Coding the AMP Parameter" on page 126.
DDNAME	ddname	Specifies name of DD statement
DISP	SHR	Indicates that you are willing to share the data set with other jobs. This subparameter alone, however, does not guarantee that sharing will take place. See "Chapter 5. Sharing a VSAM Data Set" on page 69for a description of data-set sharing.
	OLD	Defaults to SHR, if specified for an ICF or VSAM catalog
	PASS	For VSAM, KEEP is assumed for PASS.
DSNAME	dsname	Specifies VSAM data set or non-VSAM object.
DUMMY		An attempt to read results in an end-of-data condition, and an attempt to write results in a return code that indicates the write was successful. If specified, AMP='AMORG' must also be specified (see "Coding the AMP Parameter" on page 126 later in this chapter). No I/O activity is performed for a dummy data set.
UNIT	address	Must be the address of a valid device for VSAM. If not, OPEN will fail.
	type	Must be a type supported by VSAM. If not, OPEN will fail.
	group p	Must be a group supported by VSAM. If not, OPEN will fail. There must be enough units to mount all of the volumes specified. If sufficient units are available, UNIT=p can improve performance by avoiding the mounting and demounting of volumes.

Figure 27 (Part 1 of 2). JCL DD Parameters

Parameter	Subparameter	Comment
	unitcount	If the number of devices requested is greater than the number of volumes on which the data set resides, the extra devices are allocated anyway. If data and index components reside on unlike devices, the extra devices are allocated evenly between the unlike device types. If the number of devices requested is less than the number of volumes on which the data set resides but greater than the minimum number required to gain access to the data set, the devices over the minimum are allocated evenly between unlike device types. If devices beyond the count specified are in use by another task but are sharable and have mounted on them volumes containing parts of the data set to be processed, they will also be allocated to this data set. **Note:** Volume serial number and unit information should not be specified for data sets cataloged in an ICF catalog.
	DEFER	The volume to be used does not have to be mounted until the data set is opened. This causes all the units associated with demountable volumes to be flagged as nonshared.
VOLUME	**PRIVATE**	Specifies volume is demounted unless RETAIN is coded.
	RETAIN	Specifies that the volume is to retained at the system if it was demounted during the job.
	SER	The volume serial number(s) used in the access method services DEFINE command for the data set must match the volume serial numbers in the VOLUME=SER specification in the job in which the data set is defined. After a VSAM data set is defined, the volume serial (number(s) need not be specified on a DD statement to retrieve or process the data set. For data sets defined in VSAM catalogs if VOLUME=SER and UNIT=type are specified, only those volumes named are initially mounted. Other volumes may be mounted when they are needed, if at least one of the units allocated to the data set is not sharable and the number of OPENs issued against the volume is less than or equal to 1, or the unit count is greater than the total number of volumes initially mounted. One unit is flagged as nonsharable when unit count is less than the number of volume serial numbers specified. If VOLUME=SER is specified and the data set is cataloged in a user catalog, include a JOBCAT or STEPCAT DD statement to identify the catalog to the current job step unless the high-level qualifier of the data set name is also the name of the user catalog.

Figure 27 (Part 2 of 2). JCL DD Parameters

JCL PARAMETERS NOT USED WITH VSAM

VSAM ignores parameters for defining tape data sets; data-set sequence numbers, NSL, NL, BLP, and AL. You cannot use the parameters for a sequential data set (DATA, SYSOUT, and *) for specifying a VSAM data set. DD names that are invalid for VSAM data sets are: JOBLIB, STEPLIB, SYSABEND, SYSUDUMP, and SYSCHK.

DD parameters that are invalid are: UCS, QNAME, DYNAM, TERM, and the forms of DSNAME for ISAM, PAM (partioned access method), and generation data groups. VSAM does not allow for temporary data sets or concatenated data sets.

CODING A DD STATEMENT FOR A CATALOG

The master catalog is always available, without specifying it via JCL. You make other catalogs available by describing them in DD statements with special names for a job (JOBCAT) or a job step (STEPCAT), or by using special naming conventions where the high-level qualifier of the data set name is also the name of the user catalog. You describe a catalog sufficiently by giving its data set name and specifying DISP=SHR. A catalog can be either a JOBCAT or a STEPCAT catalog. If both JOBCAT and STEPCAT catalogs are specified, the STEPCAT catalog is available for the step for which it is specified, and the JOBCAT catalog is available for all steps within the job. VSAM uses a data set's name as a search argument to search a catalog. You can minimize the use of JOBCAT and STEPCAT DD statements for your jobs when you name your data set with a qualified entryname whose first qualifier is the name or alias of the catalog in which the data set is defined. When the catalog is not identified with a DD statement, the scheduler searches the master catalog for the data set's entryname. If the entryname is not found, the system uses the entryname's first qualifier as a search argument and attempts to locate either a catalog entry or a catalog's alias entry in the master catalog. If the system finds a catalog or alias entry whose name is the same as the data set name's first qualifier, the operating system searches that catalog for the data set's catalog record, using the data set's full entryname.

If no catalog is indicated or if the definition is not found in the catalog(s) that are indicated, the master catalog is assumed to contain the definition of the data set described in a DD statement. A catalog is specified either for all the steps of a job or for a particular step. To specify a job catalog, place a DD statement with the ddname JOBCAT before the first EXEC statement after the JOB statement and after a JOBLIB statement, if any:

```
  //EXAMPLE  JOB  ...
  //JOBLIB   DD   DSNAME=USER.LIB,DISP=SHR
  //JOBCAT   DD   DSNAME=usercatalogname,DISP=SHR
  //         EXEC ...
```

To specify a job step catalog, place a DD statement with the ddname STEPCAT after the EXEC statement of the step:

```
  //STEP1    EXEC ...
  //STEPCAT  DD   DSNAME=usercatalogname,DISP=SHR
```

The order in which catalogs are searched when an existing entry is to be located is:

- If a catalog is specified in a CATALOG parameter of the access method services DEFINE command, only that catalog is searched.

- Any catalog(s) specified in the current job step (STEPCAT) or, if none is specified for the job step, any user catalog(s) specified for the current job (JOBCAT). If more than one catalog is specified for the job step or job, the job-step or job catalogs are searched in order of concatenation. For information of the search order, see Access Method Services Reference.

- If the entry is not found and the entry's name is a qualified name and the first qualifier (the first one to eight characters before any period) is the same as the name or alias of a catalog or the alias of an OS CVOL, that catalog or OS CVOL is searched; otherwise, the master catalog is searched.

RESTRICTION: OS CVOLs are not searched (1) when an existing data set is to be deleted except when the data set to be deleted, is a non-VSAM data set, or (2) when an existing data set is to be altered.

CODING THE AMP PARAMETER

VSAM has one additional JCL parameter of its own: AMP. The AMP parameter takes effect when the data set defined by the DD statement is opened. It has subparameters for:

- Overriding operands specified by way of the ACB, EXLST, and GENCB macros
- Supplying operands missing from the ACB or GENCB macro
- Indicating checkpoint/restart options
- Indicating options when using ISAM macros to process a key-sequenced data set
- Indicating that the data set is a VSAM data set when you specify unit and volume information or DUMMY in the DD statement
- Indicating that you want VSAM to supply storage dumps of the access method control block(s) that identify this DD statement

The AMP parameter takes effect when the data set defined by the DD statement is opened.

The format of the AMP parameter is:

//...	**DD**	**...AMP=([‘AMORG’]**
		[,’BUFND=number**’]**
		[,’BUFNI=number**’]**
		[,’BUFSP=number**’]**
		[,’CROPS={RCK|NCK|NRE|NRC}’]
		[,’OPTCD={I|L|IL}’]
		[,’RECFM={F|FB|V|VB}’]
		[,’STRNO=number**’]**
		[,’SYNAD=modulename**’]**
		[,’TRACE’]])

where:

AMORG
specifies that the DD statement defines a VSAM data set. When you specify unit and volume information for a DCB (through the ISAM interface program) or DUMMY in the DD statement, you must specify AMORG. Under these conditions, the system doesn't have to search a catalog to find out what volume(s) are required, and therefore doesn't know that the DD statement defines a VSAM data set. You never have to specify unit and volume information, unless you want to mount a subset of the volumes on which the data set is stored, or want to defer mounting.

BUFND=number
specifies the number of data buffers.

BUFNI=number
specifies the number of index buffers.

BUFSP=number
specifies that one or more of these values is to override whatever was specified in the ACB or GENCB macro, or that one or more of these values is to be provided if not previously specified. For further information on BUFND, BUFNI, and BUFSP, see VSAM Reference.

CROPS=[RCK|NCK|NRE|NRC]
specifies one of four checkpoint/restart options, described in detail in Checkpoint/Restart. If you specify an option that is not applicable for a data set, such as the data-erase test for an input data set, the option is ignored.

RCK
specifies that a data-erase test and data set-post-checkpoint modification tests are to be performed.

NCK
specifies that data set-post-checkpoint modification tests are not to be performed.

NRE
specifies that a data-erase test is not to be performed.

NRC
specifies that neither a data-erase test nor data set-post-checkpoint modification tests are to be performed.

OPTCD={I|L|IL}
specifies the type of processing of records flagged for deletion (binary 1's in the first byte) with an ISAM processing program using the ISAM interface. I and L are described in Appendix B, "Using ISAM Programming with VSAM" on page 149.

RECFM={F|FB|V|VB}
specifies record format in the same way as the DCB (data control block) parameter that is used for processing an indexed-sequential data set. You use it when processing a VSAM data set with an ISAM processing program to indicate what record format the processing program assumes. The options are described in Appendix B, "Using ISAM Programming with VSAM" on page 149.

STRNO=number
specifies a value that is to override the STRNO value specified in the ACB or GENCB macro, or to provide a value if one was not specified.

SYNAD=modulename
specifies a value that is to override the address of a SYNAD exit routine specified in the EXLST or GENCB macro that generates the exit list. The exit list intended is the one whose address is specified in the access method control block that links this DD statement to the processing program. If no SYNAD exit was specified, the SYNAD parameter of AMP is ineffective. You can also use this parameter, when you are processing a VSAM data set with an ISAM processing program, to provide an ISAM SYNAD routine or to replace one with another.

TRACE
specifies that generalized trace facility (GTF) is to be active, along with your processing job, to gather information associated with opening, closing, and end-of-volume processing for the data set defined on this DD statement. You can print the trace output with the IMDPRDMP service program.

Trace also causes the VSAM record management trace facility to be activated. Prompts to the operator from the trace facility will occur when the cluster with this parameter is processed by VSAM. See "Record Management Trace Facility" in the Diagnostic Aids section of VSAM Logic.

Note: See Appendix B, "Using ISAM Programming with VSAM" on page 149 for additional information on the use of the AMP parameter with an ISAM processing program.

If you have more than one subparameter, they must be enclosed in apostrophes. Apostrophes can enclose each individual subparameter or group of subparameters. If you have more than one pair of apostrophes, you must enclose all the subparameters in a pair of parentheses. For example, AMP='AMORG,TRACE' or AMP=('AMORG','TRACE'). If the subparameters continue from one line to another, a pair of apostrophes cannot extend from one line to the next, and you must use a pair of parentheses to enclose all the subparameters.

The AMP parameter cannot be defined as a symbolic parameter (a symbol preceded by an ampersand (&) that stands for a parameter or the value assigned to a parameter or subparameter in a cataloged or in-stream procedure).

CHAPTER 8. USER-WRITTEN EXIT ROUTINES

User-written routines may be supplied to:

- Analyze logical errors
- Analyze physical errors
- Perform end-of-data processing
- Record transactions made against a data set
- Perform user-security verification
- Update date fields

If the exit routine is used by a program that is doing asynchronous processing with multiple request parameter lists or, if the exit routine is used by more than one data set, it must be coded so that it can handle an entry made before the previous entry's processing is completed. A particularly sensitive area is the saving and restoring of registers by the exit routine or by other routines called by the exit routine. The best way to do this is to code the exit routine reentrant; another way is to develop a technique for associating a unique save area with each request parameter list.

If the LERAD, EODAD, or SYNAD exit routine reuses the request parameter list passed to it, the exit routine should be aware that:

- Recursion occurs (that is, the exit routine is called again) if the request that issues the reused RPL results in the same exception condition that caused the exit routine to be entered originally.
- The original feedback code is replaced with the feedback code that indicates the status of the latest request issued against the RPL. If the exit routine returns to VSAM, VSAM (when it returns to the user's program) sets register 15 to also indicate the status of the latest request.

LERAD EXIT ROUTINE TO ANALYZE LOGICAL ERRORS

A LERAD exit routine should examine the feedback field in the request parameter list to determine what logical error occurred. What the routine does after determining the error depends on your knowledge of the kinds of things in the processing program that may have caused the error. When your LERAD exit routine completes processing, return to your main program as described in "Returning to Your Main Program" on page 140. If the error cannot be corrected, close the data set and either terminate processing or return to VSAM.

Figure 28 gives the contents of the registers when VSAM exits to the LERAD exit routine.

Note: A LERAD exit is not taken for RPLFDBC 64(40) because of the inability of the program to use register 14 SAVEAREA in PLH.

Reg.	Contents
0	Unpredictable.
1	Address of the request parameter list that contains the feedback field the routine should examine. The register must contain this address if you return to VSAM.
2-13	Same as when the request macro was issued. Register 13, by convention, contains the address of your processing program's 72-byte save area, which may not be used as a save area by the LERAD routine if the routine returns control to VSAM.
14	Return address to VSAM.
15	Entry address to the LERAD routine. The register doesn't contain the logical-error indicator.

Figure 28. Contents of Registers at Entry to LERAD Exit Routine

If the LERAD exit routine issues GENCB, MODCB, SHOWCB, or TESTCB and returns to VSAM, it must restore registers 1, 13, and 14, which are used by these macros. It must also provide two save areas; one, whose address should be loaded into register 13 before the GENCB, MODCB, SHOWCB, or TESTCB is issued, and the second, to separately store registers 1, 13, and 14.

If a logical error occurs and no LERAD exit routine is provided (or the LERAD exit is inactive), VSAM returns codes in register 15 and in the feedback field of the request parameter list to identify the error.

SYNAD EXIT ROUTINE TO ANALYZE PHYSICAL ERRORS

VSAM exits to a SYNAD routine if a physical error occurs when you request access to data. It also exits to a SYNAD routine when you close a data set if a physical error occurs while VSAM is writing the contents of a buffer out to direct-access storage.

A SYNAD routine should examine the feedback field in the request parameter list to identify the type of physical error that occurred. It should then get the address of the message area, if any, from the request parameter list, so that it can examine the message for detailed information about the error.

The main problem with a physical error is the possible loss of data. You should try to recover your data before continuing to process. Input operations (ACB MACRF=IN) are generally less serious than output or update operations (MACRF=OUT), because your request was not attempting to alter the contents of the data set.

If the routine cannot correct an error, it might print the physical-error message, close the data set, and terminate the program. If the error occurred while VSAM was closing the data set, and if another error occurs after the exit routine issues a CLOSE macro, VSAM doesn't exit to the routine a second time.

When your SYNAD exit routine completes processing, return to your main program as described in "Returning to Your Main Program" on page 140.

If the SYNAD routine returns to VSAM, whether the error was corrected or not, VSAM drops the request and returns to your processing program at the instruction following the last executed instruction. Register 15 is reset to indicate that there was an error, and the feedback field in the request parameter list identifies it.

Physical errors affect positioning: If a GET was issued that would have positioned VSAM for a subsequent sequential GET and an error occurs, VSAM is positioned at the control interval next in key (RPL OPTCD=KEY) or in entry (OPTCD=ADR) sequence after the control interval involved in the error. The processing program can therefore ignore the error and proceed with sequential processing. With direct processing, the likelihood of reencountering the control interval involved in the error depends on your application.

Figure 29 gives the contents of the registers when VSAM exits to the SYNAD routine.

Reg.	Contents
0	Unpredictable.
1	Address of the request parameter list that contains a feedback return code and the address of a message area, if any. If you issued a request macro, the request parameter list is the one pointed to by the macro; if you issued an OPEN, CLOSE, or EOV macro, the request parameter list was built by VSAM to process the OPEN, CLOSE or EOV request. Register 1 must contain this address of the SYNAD routine returns to VSAM.
2-13	Same as when the request macro or CLOSE macro was issued. Register 13, by convention, contains the address of your processing program's 72-byte save area, which may not be used by the SYNAD routine if it returns control to VSAM.
14	Return address to VSAM.
15	Entry address to the SYNAD routine.

Figure 29. Contents of Registers at Entry to SYNAD Exit Routine

If the exit routine issues GENCB, MODCB, SHOWCB, or TESTCB and returns to VSAM, it must provide a save area and restore registers 13 and 14, which are used by these macros.

See "SYNAD Exit Routine to Analyze Physical Errors" on page 130 for the format of a physical-error message that can be written by the SYNAD routine.

If a physical error occurs and no SYNAD routine is provided (or the SYNAD exit is inactive), VSAM returns codes in register 15 and in the feedback field of the request parameter list to identify the error. See "SYNAD Exit Routine to Analyze Physical Errors" on page 130 for a description of these return codes.

EXCEPTION EXIT ROUTINE

You can provide an exception exit routine to monitor I/O errors associated with a data set. You specify the name of your routine via the access method services DEFINE command.

If an I/O error occurs while a program with a specified SYNAD routine is processing a data set with a specified exception exit, the exception exit is taken first.

When your exception exit routine completes processing, return to your main program as described in "Returning to Your Main Program" on page 140.

For information about how exception exits are established, changed, or nullified, see Access Method Services.

EODAD EXIT ROUTINE TO PROCESS END-OF-DATA

VSAM exits to an EODAD routine when an attempt is made to sequentially retrieve or point to a record beyond the last record in the data set (one with the highest key for keyed access and the one with the highest RBA for addressed access). (VSAM doesn't take the exit for direct requests that specify a record beyond the end.) If the EODAD exit isn't used, the condition is considered a logical error (FDBK code X'04') and can be handled by the LERAD routine, if one is supplied.

The typical actions of an EODAD routine are to issue completion messages, close the data set, and terminate processing without returning to VSAM. If the routine returns to VSAM and another GET request is issued for access to the data set, VSAM exits to the LERAD routine. When your EODAD routine completes processing, return to your main program as described in "Returning to Your Main Program" on page 140.

If a processing program retrieves records sequentially with a request defined by a chain of request parameter lists, the EODAD routine must determine whether the end of the data set was reached for the first request parameter list in the chain. If not, then one or more records have been retrieved but not yet processed by the processing program.

Figure 30 gives the contents of the registers when VSAM exits to the EODAD routine.

Reg.	Contents
0	Unpredictable.
1	Address of the request parameter list that defines the request that occasioned VSAM's reaching the end of the data set. The register must contain this address if you return to VSAM.
2-13	Same as when the request macro was issued. Register 13, by convention, contains the address of your processing program's 72-byte save area, which may not be used as a save area by the EODAD routine if it returns control to VSAM.
14	Return address to VSAM.
15	Entry address to the EODAD routine.

Figure 30. Contents of Registers at Entry to EODAD Exit Routine

If the exit routine issues GENCB, MODCB, SHOWCB, or TESTCB and returns to VSAM, it must provide a save area and restore registers 13 and 14, which are used by these macros.

The type of data set whose end was reached can be determined by examining the request parameter list for the address of the access method control block that connects the program to the data set and testing its attribute characteristics.

JRNAD EXIT ROUTINE TO JOURNALIZE TRANSACTIONS

A JRNAD exit routine can be provided to record transactions against a data set and to keep track of changes in the RBAs of records. VSAM takes the JRNAD exit each time the processing program issues a GET, PUT, or ERASE; each time data is shifted right or left in a control interval or is moved to another control interval to accommodate a record's being deleted, inserted, shortened, or lengthened; and each time an I/O error occurs.

Note: Additionally, the JRNAD exit is taken each time an I/O completion occurs; each time a shared or nonshared request is received; and each time the buffer contents are to be changed.

Because the JRNAD is taken for I/O errors, a journal exit may zero out, or otherwise alter, the physical-error return code, so that a series of operations may continue to completion, even though one or more of the operations failed.

You may also want to use the JRNAD exit to maintain shared or exclusive control over certain data or index control intervals; and in some cases, you may reject the request taking the JRNAD. For example, if you used this exit to maintain information about a data set in a shared environment, you might reject the request because a control interval split or a control interval obtained in exclusive control would adversely affect other users of the data set.

Figure 31 gives the contents of the registers when VSAM exits to the JRNAD routine.

Reg.	**Contents**
0	Unpredictable.
1	Address of a parameter list with the following format:

4 bytes	Address of the request parameter list that defines the request that caused VSAM to exit to the routine	
4 bytes	Address of a 5-byte field that identifies the data set being processed. This field has the format:	
	4 bytes	Address of the access method control block that is specified by the request parameter list that defines the request that occasioned the JRNAD exit's being taken
	1 byte	Indication of whether the data set is the data (X'01') or the index (X'02') component
4 bytes	For RBA changes only, the RBA of the first byte of data that is being shifted or moved	
4 bytes	For RBA changes only, the number of bytes of data that is being shifted or moved (this number doesn't include free space, if any, or control information—except for a control area split, when the whole contents of a control interval are moved to a new control interval)	

Figure 31 (Part 1 of 2). Contents of Registers at Entry to JRNAD Exit Routine

Reg.	Contents		
	4 bytes	For RBA changes only, the RBA of the first byte to which data is being shifted or moved	
	1 byte	Indication of the reason VSAM exited to the JRNAD routine:	
		X'00'	GET request
		X'04'	PUT request
		X'08'	ERASE request
		X'0C'	RBA change
		X'10'	Read spanned record segment
		X'14'	Write spanned record segment
		X'18'	Reserved
		X'1C'	Reserved
		For shared resources:[1]	
		X'20'	Control area split
		X'24'	Input error
		X'28'	Output error
		X'2C'	Buffer write
		X'30'	A data or index control interval is about to be read in exclusive control
		X'34'	A data or index control interval is about to be read in shared status
		X'38'	Acquire exclusive control of a control interval already in the buffer pool
		X'3C'	Build a new control interval for the data set and hold it in exclusive control
		X'40'	Exclusive control of the indicated control interval already has been released
		X'44'	Contents of the indicated CI have been made invalid
		X'48'	Read completed
		X'4C'	Write completed
		X'50' to X'FF'	Reserved
		X'FC'	Reserved
	1 byte	Reserved.	
2-13	Unpredictable.		
14	Return address to VSAM.		
15	Entry address to the JRNAD routine.		

[1] Described in "Chapter 6. Options for Advanced Applications" on page 81.

Figure 31 (Part 2 of 2). Contents of Registers at Entry to JRNAD Exit Routine

If the exit routine issues GENCB, MODCB, SHOWCB, or TESTCB, it must restore register 14, which is used by these macros, before it returns to VSAM.

If the exit routine uses register 1, it must restore it with the parameter list address before returning to VSAM. (The routine must return for completion of the request that caused VSAM to exit.)

For journaling transactions (when VSAM exits because of a GET, PUT, or ERASE), you can use the SHOWCB macro to display information in the request parameter list about the record that was retrieved, stored, or deleted, (FIELDS=(AREA,KEYLEN,RBA,RECLEN), for example). You can also use the TESTCB macro to find out whether a GET or a PUT was for update (OPTCD=UPD).

For recording RBA changes, you must calculate how many records there are in the data being shifted or moved, so you can keep track of the new RBA for each. With fixed-length records, you calculate the number by dividing the record length into the number of bytes of data being shifted. With variable-length records, you could calculate the number by using a table that not only identifies the records (by associating a record's key with its RBA), but also gives their length.

Some control interval splits involve data being moved to two new control intervals, and control area splits normally involve many control intervals' contents being moved. In these cases, VSAM exits to the JRNAD routine for each separate movement of data to a new control interval.

You should provide a routine to keep track of RBA changes caused by control interval and control area splits. RBA changes that occur by way of keyed access to a key-sequenced data set must also be recorded if you intend to process the data set later by direct-addressed access.

If your JRNAD routine only journals transactions, it should ignore reason X'0C' and return to VSAM; conversely, it should ignore reasons X'00', X'04', and X'08' if it only records RBA changes.

The JRNAD exit must be indicated as active before the data set for which the exit is to be used is opened, and the exit must not be made inactive during processing. If you define more than one access method control block for a data set and want to have a JRNAD routine, the first ACB you open for the data set must specify the exit list that identifies the routine.

UPAD EXIT ROUTINE FOR USER PROCESSING

You can perform special processing during a VSAM request with the UPAD exit routine. For example, VSAM takes the UPAD exit immediately prior to issuing a WAIT for I/O completion or for a serially reusable resource. VSAM exits to the UPAD routine when the request's RPL specifies OPTCD=(SYN, WAITX) and the ACB specifies MACRF=LSR or MACRF=GSR, or under DFEF, MACRF=ICI.

Note: The UPAD exit is also taken for request resumption (POST) during synchronous requests. This exit will be taken only for users that are in cross-memory mode.

The UPAD exit routine must be active before the data set is opened. The exit must not be made inactive during processing. If the UPAD exit is desired and many ACBs are used for processing the data set, the first ACB that is opened must specify the exit list that identifies the UPAD exit routine.

When the UPAD exit routine is entered, register contents passed by VSAM are:

Reg.	Contents
0	Unpredictable
1	Address of a parameter list built by VSAM
2-12	Unpredictable
13	Reserved
14	Return address to VSAM
15	Entry address of the UPAD routine

The contents of the parameter list built by VSAM, pointed to by register 1, can be examined by the UPAD exit routine:

Offset	Bytes	Description	
0(0)	4	Address of the RPL	
4(4)	4	Address of a 5-byte data set identifier. The first four bytes of the identifier are the ACB address; the last byte identifies the component; data (X'01'), or index (X'02').	
8(8)	4	Address of the request's ECB	
12(0C)	4	Return code for POST UPAD	
	4	Reserved	
16(1C)		Reserved	
20(14)	1	Reason code:	
		X'00'	VSAM is about to wait
		X'04'	VSAM ready to resume request processing
		X'084'-'FC'	Reserved

If the UPAD exit routine modifies register 14 (for example, by issuing a TESTCB, the routine must restore register 14 before returning to VSAM. If register 1 is used, the UPAD exit routine must restore it with the parameter list address before returning to VSAM.

The UPAD routine must return to VSAM under the same TCB from which it was called for completion of the request that caused VSAM to exit. Note that the UPAD exit routine cannot use register 13 as a save area pointer without first obtaining its own save area.

If you are executing in cross-memory mode, you must have a UPAD routine. When posting of an event is required, the UPAD routine is given control (reason code 4) to do post processing; you must return control to VSAM after the post. This call to the UPAD may not be cancelled.

When VSAM regains control from a UPAD exit that was taken for reason code 4, VSAM tests the return code at offset 12 in the parameter list. If it is nonzero and the request is in cross-memory mode, VSAM indicates a logical error rather than attempting to issue a POST. (POST would cause an abend if issued in cross-memory mode.) Therefore, if you are in cross-memory mode, you must ensure that the UPAD exit is specified, that it resumes the indicated request, and that it sets the appropriate return code in the parameter list before returning to VSAM. If you are not in cross-memory mode and the UPAD returns with a nonzero code, VSAM will issue a POST for TCB mode and a SCHEDULE for SRB mode POST.

The UPAD exit routine, when taken prior to a WAIT during LSR or GSR processing, might issue other VSAM requests to obtain better processing overlap (similar to asynchronous processing). However, the UPAD routine must not issue any synchronous VSAM requests that do not specify WAITX, because a started request might issue a WAIT for a resource owned by a starting request. If the UPAD routine starts requests that specify WAITX, the UPAD routine must be reentrant. Once multiple requests have been started, they should be synchronized by waiting for one ECB out of a group of ECBs to be posted complete rather than waiting for a specific ECB or for many ECBs to be posted complete. (Posting of some ECBs in the list might be dependent upon the resumption of some of the other requests that entered the UPAD routine.)

USER-SECURITY-VERIFICATION ROUTINE

If you use VSAM password protection, you may also have your own routine to check a requestor's authority. VSAM transfers control to your routine, which must reside in SYS1.LINKLIB, when a requester gives a correct password other than the master password.

Through the access method services DEFINE command, you may identify your user-security-verification routine (USVR) and associate as many as 256 bytes of your own security information with each data set to be protected. This information—the user security-authorization record (USAR)—is made available to the user-security-verification routine when the routine gets control. You may restrict access to the data set as you choose; for example, you may require that the owner of a data set give ID when defining the data set and then allow only the owner to gain access to the data set.

Figure 32 gives the contents of the registers when VSAM gives control to the user-security-verification routine.

If the user-security-verification routine is being used by more than one task at a time, you must code the user-security-verification routine reentrant or develop another method for handling simultaneous entries.

Reg.	Contents
0	Unpredictable.
1	Address of a parameter list with the following format:

44 bytes	Name of the data set for which authority to process is to be verified (the name you specified when you defined it with access method services).
8 bytes	Prompting code (or 0's).
8 bytes	Owner identification (or 0's).
8 bytes	The password that the requester gave (it has been verified by VSAM).
2 bytes	Length of the user-security-authorization routine (in binary).
—	The user-security-authorization.

2-13	Unpredictable.
14	Return address to VSAM.
15	Entry address to the user-security-verification routine. When the routine returns to VSAM, it indicates by the following codes in register 15 whether the requester has been authorized to gain access to the data set:

0	Authority granted.
not 0	Authority withheld.

Figure 32. Communication with User-Security-Verification Routine

DATESTAMP ROUTINE

The datestamp control module (IDATMSTP) is provided as a dummy module that causes datestamp processing to be skipped. It sets a return code of 0 that causes IDA0192B to skip the processing of the last-referenced date (DS1REFD) in the Format-1 DSCB for VSAM data sets cataloged under Integrated Catalog Facility (ICF). IDATMSTP may be replaced with another module you select that sets a return code of 4 and causes datestamp processing on all specified data sets.

The Format-1 DSCB is read using CVAFDIR, and, if it is modified, it is written back. The DSCB may be modified for either of the following reasons:

- DS1IND02 is changed. DS1IND02 will be changed if you have indicated that the data set is being opened for other than input, and DS1IND02 does not already show that the data set has been opened for other than input.

- DS1REFD is not equal to zero and is earlier than today's date.

  Note: It is your responsibility to set DS1REFD to a nonzero value when you want OPEN to do datestamp processing.

Your module can include code to cause checking of some or all VSAM data sets cataloged in an ICF catalog for periodic migration to other storage media and maintenance of the updated indicator. A return code of 4 from the module indicates that you want the check, and causes the updated indicator to be maintained.

IDATMSTP is passed the addresses of OPWCDDSN, JFCBVOLS, and of a character 'D' to indicate that the data set named in OPWCDDSN, on the volume identified in JFCBVOLS is a data component of a data set cataloged in an ICF catalog. This information is available to your module and can be used to further qualify which data sets should have datestamp processing.

Figure 33 shows the contents of the registers when VSAM gives control to IDATMSTP.

Reg.	**Contents**
0	Unpredictable.
1	Address of a parameter list containing the following addresses of offset: 0 Data set name in the current cluster information area. 4 List of volume serial numbers that contain the data set in the JFCB. 8 Address of a 1-byte indicator set to D to indicate the data set is a base data component.
2 - 12	Unpredictable.
13	Save area address.
14	Return address in VSAM OPEN.
15	Entry address to IDATMSTP.

When IDATMSTP returns to VSAM OPEN, the desired datestamp option is indicated in register 15:

0	No datestamp processing.
4	Datestamp processing desired; updated indicator (DS1IND02) maintained.

Figure 33. Communication with Datestamp Routine

RETURNING TO YOUR MAIN PROGRAM

Five exit routines can be entered when your main program issues a VSAM request macro (GET, PUT, POINT, and ERASE) and the macro fails to complete successfully: LERAD, SYNAD, EODAD, UPAD, or the exception exit routine. When your exit routine completes its processing, it can return to your main program in one of two ways:

1. The exit routine can return to VSAM (via the return address in register 14); VSAM then returns to your program at the instruction following the VSAM request macro that failed to complete successfully.

2. The exit routine can determine the appropriate return point in your program, then branch directly to that point. Note that, when VSAM enters your exit routine, none of the registers contains the address of the instruction following the failing macro.

Method 1 provides the easier way to return to your program. However, there is a special situation that requires you to return via method 2: Your exit routine, during the error recovery and correction process, has issued a GET, PUT, POINT, or ERASE macro that refers to the request parameter list referred to by the failing VSAM macro (that is, the request parameter list has been reissued by the exit routine). In this case, VSAM has lost track of its reentry point to your main program. If the exit routine returns to VSAM, VSAM issues an ABEND.

If your error recovery and correction process needs to reissue the failing VSAM macro against the request parameter list in order to retry the failing request or to correct it:

- Your exit routine can correct the request parameter list (using MODCB), then set a switch to indicate to your main program that the request parameter list is now ready to retry. When your exit routine completes processing, it can return to VSAM (via register 14), which returns to your main program. Your main program can then test the switch and reissue the VSAM macro and request parameter list.

- Your exit routine can issue a GENCB macro to build a request parameter list, and then copy the request parameter list (for the failing VSAM macro) into the newly built RPL. At this point, your exit routine can issue VSAM macros against the newly built RPL. When your exit routine completes processing, it can return to VSAM (via register 14), which returns to your main program.

EXAMPLE: USER-WRITTEN EXIT ROUTINE

This example demonstrates a user-written exit routine. It is a SYNAD exit routine that examines the FDBK field of the RPL checking for the type of physical error that caused the exit. After the checking, special processing may be performed as necessary. The routine returns to VSAM after printing an appropriate error message on SYSPRINT.

```
ACB1     ACB    EXLST=EXITS

EXITS    EXLST  SYNAD=PHYERR

RPL1     RPL    ACB=ACB1,
                MSGAREA=PERRMSG,
                MSGLEN=128

PHYERR   USING  *,15                This routine is nonreentrant.

*                                   Register 15 is entry address.

         .
         .                          Save caller's register
                                    (1, 13, 14).
         .
         LA     13,SAVE             Point to routine's save area.
         .
         .                          If register 1=address of RPL1,
         .                          then error did not occur for a
                                    CLOSE.
         SHOWCB RPL=RPL1,
                FIELDS=FDBK,
                AREA=ERRCODE,
                LENGTH=4

*                                   Show type of physical error.

         .
         .                          Examine error, perform special
         .                          processing.
         PUT    PRTDCB,ERRMSG       Print physical error message.
         .
         .                          Restore caller's registers
         .                          (1, 13, 14).
         BR     14                  Return to VSAM.
         .
         .
         .
ERRCODE  DC     F'0'                RPL error code from SHOWCB.

PERRMSG  DS     0XL128              Physical error message.

         DS     XL12                Pad for unprintable part.

ERRMSG   DS     XL116               Printable format part of
                                    message.
         .
         .
         .
PRTDCB   DCB    .....               QSAM DCB

SAVE     DS     18F                 SYNAD routine's save area

SAVREG   DS     3F                  Save registers 1, 13, 14
```

APPENDIX A. INVOKING ACCESS METHOD SERVICES FROM A PROBLEM PROGRAM

Access method services can be invoked by a problem program through the ATTACH, LINK, or LOAD and CALL macro instructions.

The dynamic invocation of access method services enables respecification of selected processor defaults as well as the ability to manage input/output operations for selected data sets.

AUTHORIZED PROGRAM FACILITY (APF)

For information on APF authorization, see "Security Protection."

INVOKING MACRO INSTRUCTIONS

The following descriptions of the invoking macro instructions are related to Figure 34 on page 145, which describes the argument lists referenced by the invoking macros.

LINK OR ATTACH MACRO INSTRUCTION

Access method services may be invoked through either the LINK or the ATTACH macro instruction.

The format of the LINK or ATTACH macro instruction is:

[name]	LINK|ATTACH	EP=IDCAMS,
		PARAM=(optionaddr
		[,dnameaddr]
		[,pgnoaddr]
		[,iolistaddr]),
		VL=1

EP=IDCAMS
specifies that the program to be invoked is IDCAMS.

PARAM=
specifies the addresses of the parameters to be passed to IDCAMS. These values can be coded:

optionaddr
specifies the address of an option list, which can be specified in the PARM parameter of the EXEC statement and is a valid set of parameters for the access method services PARM command. If you do not wish to specify any options, this address must point to a halfword of binary zeros. Figure 34 on page 145 shows the format of the options list.

dnameaddr
specifies the address of a list of alternate dd names for standard data sets used during IDCAMS processing. If standard dd names are used and this is not the last parameter in the list, it should point to a halfword of binary zeros. If it is the last parameter, it may be omitted. Figure 34 shows the format of the alternate dd name list.

pgnoaddr
specifies the address of a 3-byte to 6-byte area that contains an EBCDIC starting page number for the system output file. If the page number is not specified, but this is not the last parameter in the list, the parameter must point to a halfword of binary zeros.

If it is the last parameter, it may be omitted. If omitted, the default page number is 1. Figure 34 on page 145 shows the format of the page number area.

<u>iolistaddr</u>
specifies the address of a list of externally controlled data sets and the addresses of corresponding I/O routines. If no external I/O routines are supplied, this parameter may be omitted. Figure 34 shows the format of the I/O list.

VL=1
causes the high-order bit of the last address parameter of the PARAM list to be set to 1.

LOAD AND CALL MACRO INSTRUCTIONS

Access method services may also be invoked via a LOAD of the module IDCAMS, followed by a CALL to that module. The format of the LOAD macro instruction is:

[<u>name</u>]	**LOAD**	**{EP=IDCAMS|EPLOC=**<u>address of name</u>**}**

where:

EP=IDCAMS
is the entry point name of the IDCAMS program to be loaded into virtual storage.

EPLOC=<u>address of name</u>
is the address of an 8-byte character string 'IDCAMS bb'.

After loading IDCAMS, register 15 must be loaded with the address returned from the LOAD macro. Then CALL may be used to pass control to IDCAMS. The format of the CALL macro instruction is:

[<u>name</u>]	LR CALL	**15,0** **(15),** **(**<u>optionaddr</u> **[,**<u>dnameaddr</u>**]** **[,**<u>pgnoaddr</u>**]** **[,**<u>iolistaddr</u>**]),** **VL**

where:

15
is the register containing the address of the entry point to be given control.

<u>optionaddr</u>
specifies the address of an options list that can be specified in the PARM parameter of the EXEC statement and is a valid set of parameters for the access method services PARM command. If you do not want to specify any options, this address must point to a halfword of binary zeros. Figure 34 on page 145 shows the format of the options list.

<u>dnameaddr</u>
specifies the address of a list of alternate dd names for standard data sets used during IDCAMS processing. If standard dd names are used and this is not the last parameter in the list, it should point to a halfword of binary zeros. If it is the last parameter, it may be omitted. Figure 34 shows the format of the alternate dd name list.

pgnoaddr
: specifies the address of a 6-byte area that contains an EBCDIC starting page number for the system output file. If the page number is not specified, but this is not the last parameter in the list, the parameter must point to a halfword of binary zeros. If it is the last parameter, it may be omitted. If omitted, the default page number is 1. Figure 34 on page 145 shows the format of the page number area.

iolistaddr
: specifies the address of a list of externally controlled data sets and the addresses of corresponding I/O routines. If no external I/O routines are supplied, this parameter may be omitted. Figure 34 shows the format of the I/O list.

VL
: causes the high-order bit of the last address parameter in the macro expansion to be set to 1.

INVOCATION FROM A PL/I PROGRAM

Access method services may also be invoked from a PL/I program using the facilities of the IBM PL/I Optimizing Compiler Program Product. IDCAMS must be declared to the compiler as an external entry point with the ASSEMBLER and INTER options. The access method services processor is loaded by issuing a FETCH IDCAMS statement, is branched to via a CALL statement, and deleted via a RELEASE IDCAMS statement. The format of the CALL statement is:

CALL	**IDCAMS**	(options[,dnames][,pageno][,iolist]);

where:

options
: specifies a valid set of parameters for the access method services PARM command. If no parameters are to be specified, options should be a halfword of binary zeros. Figure 34 on page 145 shows the format of the options area.

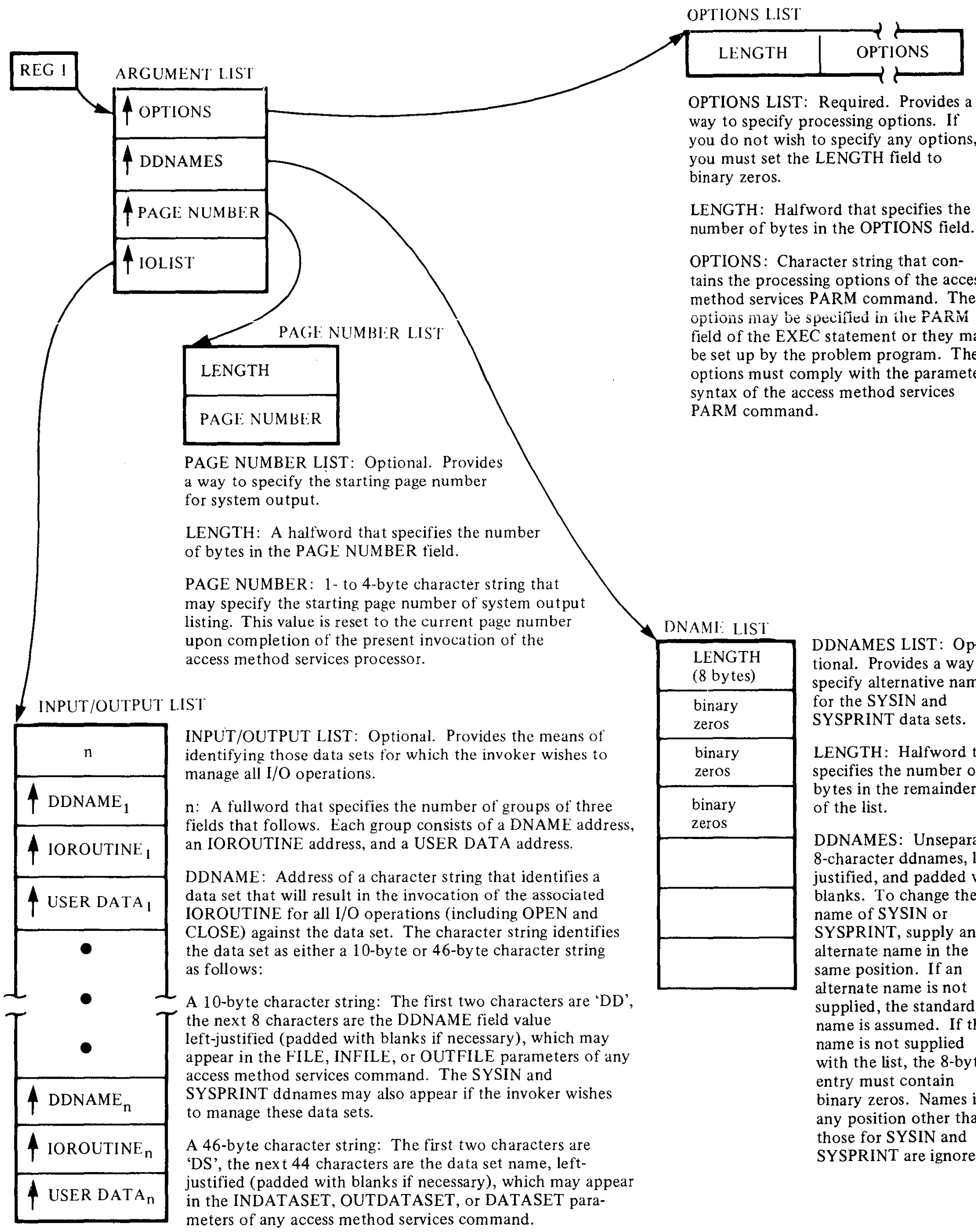
dnames
: specifies a list of alternate dd names for standard data sets used during IDCAMS processing. If standard dd names are used and this is not the last parameter in the list, dnames should be a halfword of binary zeros. If it is the last parameter, it may be omitted. Figure 34 shows the format of the alternate dd names list.

pageno
: specifies a 6-byte field that contains an EBCDIC starting page number for the system output file. If the page number is not specified, but this is not the last parameter in the list, the parameter must be a halfword of binary zeros. If it is the last parameter, it may be omitted. If not specified, the default page number is 1. Figure 34 shows the format of the page number area.

iolist
: specifies a list of externally controlled data sets and the addresses of corresponding I/O routines. If no external I/O routines are supplied, this parameter may be omitted. Figure 34 shows the format of the I/O list.

PROCESSOR INVOCATION

Figure 34 on page 145 shows the processor invocation argument list as it exists in the user's area.



OPTIONS LIST: Required. Provides a way to specify processing options. If you do not wish to specify any options, you must set the LENGTH field to binary zeros.

LENGTH: Halfword that specifies the number of bytes in the OPTIONS field.

OPTIONS: Character string that contains the processing options of the access method services PARM command. The options may be specified in the PARM field of the EXEC statement or they may be set up by the problem program. The options must comply with the parameter syntax of the access method services PARM command.

PAGE NUMBER LIST: Optional. Provides a way to specify the starting page number for system output.

LENGTH: A halfword that specifies the number of bytes in the PAGE NUMBER field.

PAGE NUMBER: 1- to 4-byte character string that may specify the starting page number of system output listing. This value is reset to the current page number upon completion of the present invocation of the access method services processor.

DDNAMES LIST: Optional. Provides a way to specify alternative names for the SYSIN and SYSPRINT data sets.

LENGTH: Halfword that specifies the number of bytes in the remainder of the list.

DDNAMES: Unseparated 8-character ddnames, left-justified, and padded with blanks. To change the name of SYSIN or SYSPRINT, supply an alternate name in the same position. If an alternate name is not supplied, the standard name is assumed. If the name is not supplied with the list, the 8-byte entry must contain binary zeros. Names in any position other than those for SYSIN and SYSPRINT are ignored.

INPUT/OUTPUT LIST: Optional. Provides the means of identifying those data sets for which the invoker wishes to manage all I/O operations.

n: A fullword that specifies the number of groups of three fields that follows. Each group consists of a DNAME address, an IOROUTINE address, and a USER DATA address.

DDNAME: Address of a character string that identifies a data set that will result in the invocation of the associated IOROUTINE for all I/O operations (including OPEN and CLOSE) against the data set. The character string identifies the data set as either a 10-byte or 46-byte character string as follows:

A 10-byte character string: The first two characters are 'DD', the next 8 characters are the DDNAME field value left-justified (padded with blanks if necessary), which may appear in the FILE, INFILE, or OUTFILE parameters of any access method services command. The SYSIN and SYSPRINT ddnames may also appear if the invoker wishes to manage these data sets.

A 46-byte character string: The first two characters are 'DS', the next 44 characters are the data set name, left-justified (padded with blanks if necessary), which may appear in the INDATASET, OUTDATASET, or DATASET parameters of any access method services command.

IOROUTINE: Address of the program that is to be invoked to process I/O operations upon the data set associated with DNAME. This routine, instead of the processor, will be invoked for all operations against the data set. See "User I/O Routines" in this appendix for linkage and interface conventions between the IOROUTINE and access method services.

USER DATA: Address of a data area user may use for any purpose.

Figure 34. Processor Invocation Argument List from a Problem Program

Entry and exit to the access method services processor occurs through IDCSA01, a module of the system adapter. Standard linkage is used; that is, register 1 points to the argument list, register 13 points to a save area, register 14 contains the return address, and register 15 contains the entry point address for IDCSA01. On exit from the access method services processor, register 15 contains the value of MAXCC (see "Processor Condition Codes" later in this chapter).

The argument list, as shown in Figure 34, can be a maximum of four fullword addresses pointing to strings of data. The last address in the list contains a "1" in the sign field. The first three possible strings of data begin with a 2-byte length field. A null element in the list can be indicated by either an address of zeros or a length of zero.

PROCESSOR CONDITION CODES

The processor's condition code is LASTCC, which can be interrogated in the command stream following each functional command. The possible values, their meanings, and examples of causes are:

Code	Meaning
0(0)	The function was executed successfully. Informational messages may have been issued.
4(4)	Some minor problems in executing the complete function were encountered, but it was possible to continue. The results may not be exactly what the user wants, but no permanent harm appears to have been done by continuing. A warning message was issued.
8(8)	A function could not perform all that was asked of it. The function was completed, but specific details were bypassed.
12(C)	The entire function could not be performed.
16(10)	Severe error or problem encountered. Remainder of command stream is flushed and processor returns condition code 16 to the operating system.

LASTCC is set by the processor at the completion of each functional command. MAXCC, which can also be interrogated in the command stream, is the highest value of LASTCC thus far encountered.

USER I/O ROUTINES

User I/O routines enable a user to perform all I/O operations for a data set that would normally be handled by the access methods services processor. This makes it possible, for instance, to control the command input stream by providing an I/O routine for SYSIN.

A user I/O routine is invoked by access method services for all operations against the selected data sets. The identification of the data sets and their associated I/O routines is via the input/output list of the processor invocation parameter list (see Figure 34 on page 145).

When writing a user I/O routine, the user must be aware of three things: First, the processor handles the user data set as if it were a non-VSAM data set that contains variable-length unblocked records (maximum record length is 32 760 bytes) with a physical sequential organization. The processor does not test for the existence of the data set. Second, the user must know the data format so that the user's routine can be coded to handle the correct type of input and format the correct type of output. Third, each user routine must handle errors encountered for data sets it is managing and provide a return code to the processor in register 15. The processor uses the return code to determine what it is to do next.

The permissible return codes are:

Code	**Meaning**
0(0)	Operation successful.
4(4)	End of data for a GET operations.
8(8)	Error encountered during a GET/PUT operation, but continue processing.
12(C)	Do not allow any further calls (except CLOSE) to this routine.

Figure 35 on page 148 shows the argument list used in communication between the user I/O routine and the access method services processor. The user I/O routine is invoked by the processor for OPEN, CLOSE, GET, and PUT routines.

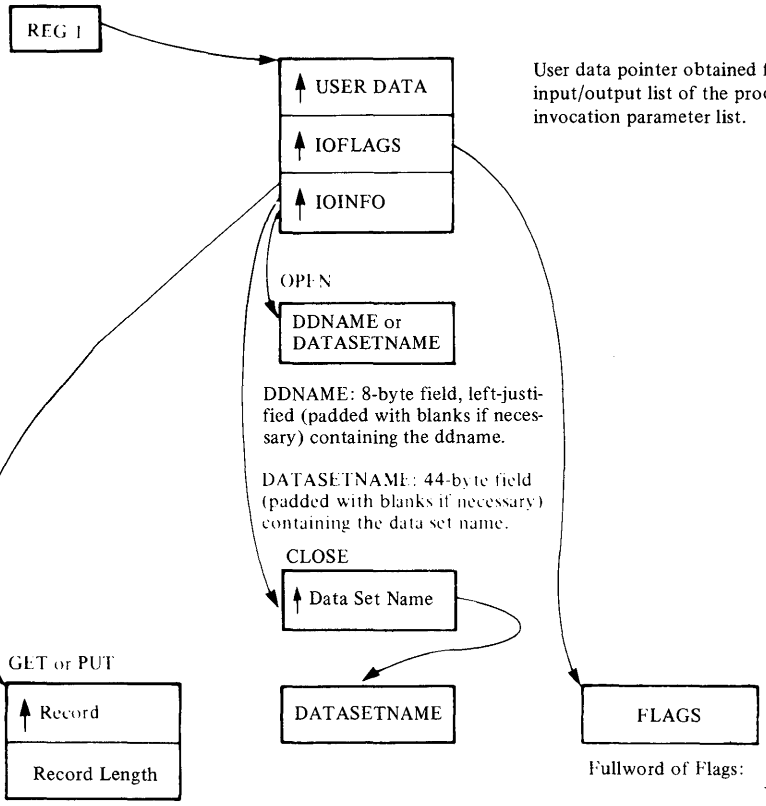
The type of operation to be performed is indicated via IOFLAGS. The IOINFO field indicates, for OPEN and CLOSE operations, the data set name or ddname of the data set; for GET and PUT operations, the IOINFO field is used to communicate the record length and address.

A user I/O routine for SYSPRINT receives control each time the processor issues a PUT against the SYSPRINT data set. If the PUT has been issued to print an IDC message, the unique message number is passed to the routine via IOFLAGS (see Figure 35 on page 148). Each IDC message is in the form IDCsnnnI or IDCsnnnnI, where:

s is a code indicating the severity of the problem

nnn or nnnn is the message number that is unique across all IDC messages

The 2-byte message number passed via IOFLAGS is the nnn or nnnn portion of the message converted to binary.



User data pointer obtained from the input/output list of the processor invocation parameter list.

DDNAME: 8-byte field, left-justified (padded with blanks if necessary) containing the ddname.

DATASETNAME: 44-byte field (padded with blanks if necessary) containing the data set name.

For a GET, this information is returned to the processor by the user's I/O routine in the 8-byte area passed to the routine.

For a PUT, the processor gives this information to the user's I/O routine.

Record (GET): Address of retrieved record.

Record Length (GET): Fullword length of retrieved record.

Record (PUT): Address of record to be written.

Record Length (PUT): Fullword length of record to be written.

DATASETNAME: 44-byte field (padded with blanks if necessary) containing the data set name.

Fullword of Flags:

	Value or Bit Pattern	Meaning
Byte 1 (Operation)	X'00'	OPEN
	X'04'	CLOSE
	X'08'	GET
	X'0C'	PUT
Byte 2	1	OPEN for input
	. 1	OPEN for output
	. . 1	Indicates IOINFO contains the address of a ddname on OPEN.
	. . . 1	Indicates IOINFO contains the address of a data set name on OPEN.
Bytes 3, 4 (Record type for PUT only)	0	Normal data record is to be written.
	n	Message serial number converted to binary if IDC message is to be written. (See "User I/O Routine" for a description of this value.)

Figure 35. Arguments Passed To and From a User I/O Routine

APPENDIX B. USING ISAM PROGRAMMING WITH VSAM

VSAM, through its ISAM interface program, enables a debugged program that processes an indexed-sequential data set to process a key-sequenced data set. The key-sequenced data set may have been converted from an indexed-sequential or a sequential data set (or another VSAM data set) or may have been loaded by one of your own programs. The loading program may be coded with VSAM macros or with ISAM macros or PL/I or COBOL statements. That is, you can load records into a newly defined key-sequenced data set with a program that was coded to load records into an indexed-sequential data set.

There are some minor restrictions on the types of processing an ISAM program may do if it is to be able to process a key-sequenced data set. These restrictions are described in "Restrictions on the Use of the ISAM Interface" on page 162.

Significant performance improvement can be gained by modifying an ISAM program that issues multiple OPEN and CLOSE macros to switch between a QISAM and BISAM DCB. The ISAM program can be modified to open the QISAM and BISAM DCBs at the beginning of the program and to close them when all processing is complete. The performance improvement is proportional to the frequency of OPEN and CLOSE macros in the ISAM program.

Figure 36 on page 150 shows the relationship between ISAM programs processing VSAM data with the ISAM interface and VSAM programs processing the data.

HOW AN ISAM PROGRAM CAN PROCESS A VSAM DATA SET

When a processing program that uses ISAM (assembler-language macros, PL/I, or COBOL) issues an OPEN to open a key-sequenced data set, the ISAM interface is given control to:

- Construct control blocks that are required by VSAM
- Load the appropriate ISAM interface routines into virtual storage
- Initialize the ISAM DCB (data control block) to enable the interface to intercept ISAM requests
- Take the DCB exit requested by the processing program

The ISAM interface intercepts each subsequent ISAM request, analyzes it to determine the equivalent keyed VSAM request, defines the keyed VSAM request in a request parameter list, and initiates the request.

The ISAM interface receives return codes and exception codes for logical and physical errors from VSAM, translates them to ISAM codes, and routes them to the processing program or error-analysis (SYNAD) routine by way of the ISAM DCB or DECB. Figure 37 on page 151 shows QISAM error conditions and the meaning they have when the ISAM interface is being used.

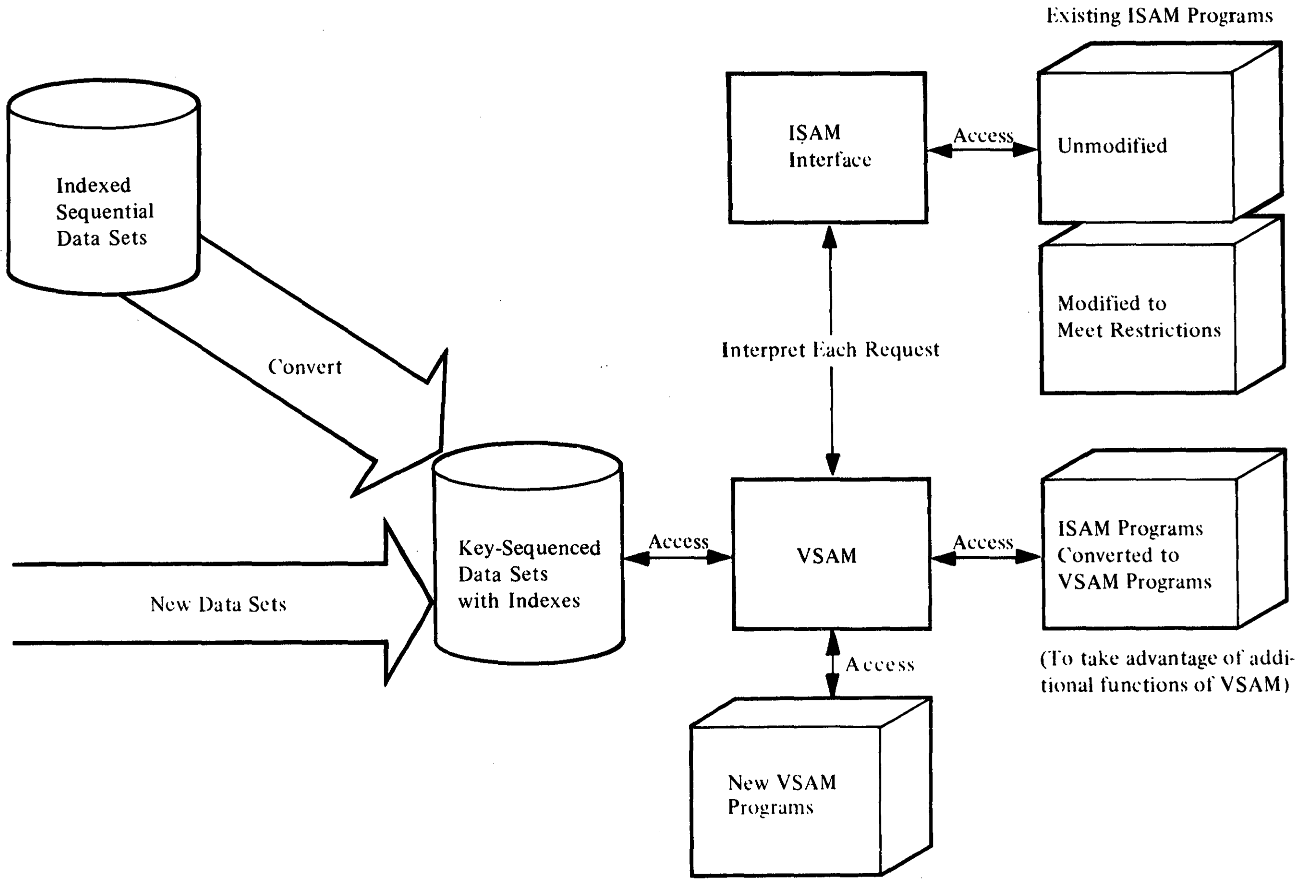

Existing ISAM Programs
Indexed Sequential Data Sets
ISAM Interface
Access
Unmodified
Modified to Meet Restrictions
Interpret Each Request
Convert
Key-Sequenced Data Sets with Indexes
Access
VSAM
Access
ISAM Programs Converted to VSAM Programs
New Data Sets
Access
(To take advantage of additional functions of VSAM)
New VSAM Programs

Figure 36. Use of ISAM Processing Programs

Byte and Offset	QISAM Meaning	Error Detected By	Request Parameter List Error Code	Interface/VSAM Meaning
DCBEXCD1				
Bit 0	Record not found	Interface		Record not found (SETL K for a deleted record)
		VSAM	16	Record not found
		VSAM	24	Record on nonmountable volume
Bit 1	Invalid device address	—	—	Always 0
Bit 2	Space not found	VSAM	28	Data set cannot be extended
		VSAM	40	Virtual storage not available
Bit 3	Invalid request	Interface		Two consecutive SETL requests
		Interface		Invalid SETL (I or ID)
		Interface		Invalid generic key (KEY=0)
		VSAM	4	Request after end-of-data
		VSAM	20	Exclusive use conflict
		VSAM	36	No key range defined for insertion
		VSAM	64	Placeholder not available for concurrent data-set positioning
		VSAM	96	Key change attempted
Bit 4	Uncorrectable input error	VSAM	4	Physical read error (register 15 contains a value of 12) in the data component
		VSAM	8	Physical read error (register 15 contains a value of 12) in the index component
		VSAM	12	Physical read error (register 15 contains a value of 12) in the sequence set of the index
Bit 5	Uncorrectable output error	VSAM	16	Physical write error (register 15 contains a value of 12) in the data component
		VSAM	20	Physical write error (register 15 contains a value of 12) in the index component
		VSAM	24	Physical write error (register 15 contains a value of 12) in the sequence set of the index

Figure 37 (Part 1 of 2). QISAM Error Conditions

Byte and Offset	QISAM Meaning	Error Detected By	Request Parameter List Error Code	Interface/VSAM Meaning
Bit 6	Unreachable block input	VSAM		Logical error not covered by other exception codes
Bit 7	Unreachable block (output)	VSAM		Logical error not covered by other exception codes
DEBEXCD2				
Bit 0	Sequence check	VSAM	12	Sequence check
		Interface		Sequence check (occurs only during resume load)
Bit 1	Duplicate record	VSAM	8	Duplicate record
Bit 2	DCB closed when error routine entered	VSAM		Error in close error routine entered
Bit 3	Overflow record	Interface	—	Always 1
Bit 4	Length of logical record is greater than DCBLRECL (VLR only)	Interface	—	Length of logical record is greater than DCBLRECL (VLR only)
		VSAM	108	Invalid record length
Bits 5-7	Reserved		—	Always 0

Figure 37 (Part 2 of 2). QISAM Error Conditions

Figure 38 shows BISAM error conditions and the meaning they have when the ISAM interface is being used.

If invalid requests occur in BISAM that didn't occur previously and the request parameter list indicates that VSAM isn't able to handle concurrent data-set positioning, the value specified for the STRNO AMP parameter should be increased. If the request parameter list indicates an exclusive-use conflict, reevaluate the share options associated with the data.

Byte and Offset	QISAM Meaning	Error Detected By	Request Parameter List Error Code	Interface/VSAM Meaning
DCBEXC1				
Bit 0	Record not found	VSAM	16	Record not found
		VSAM	24	Record on non-mountable volume
Bit 1	Record length check	VSAM	108	Record length check
Bit 2	Space not found	VSAM	28	Data set cannot be extended
Bit 3	Invalid request	Interface	—	No request parameter list available
		VSAM	20	Exclusive-use conflict
		VSAM	36	No key range defined for insertion
		VSAM	64	Placeholder not available for concurrent data-set positioning
		VSAM	96	Key change attempted
Bit 4	Uncorrectable I/O	VSAM	—	Physical error (register error 15 will contain a value of 12)
Bit 5	Unreachable block	VSAM	—	Logical error not covered by any other exception code
Bit 6	Overflow record	Interface	—	Always one for a successful READ request
Bit 7	Duplicate record	VSAM	8	Duplicate record
DECBEXC2				
Bits 0-5	Reserved		—	Always 0
Bit 6	Channel program initiated by an asynchronous routine		—	Always 0
Bit 7	Previous macro was READ KU	Interface	—	Previous macro was READ KU

Figure 38. BISAM Error Conditions

Figure 39 gives the contents of registers 0 and 1 when a SYNAD routine specified in a DCB gets control.

Reg.	BISAM	QISAM
0	Address of the DECB	0, or, for a sequence check, the address of a field containing the higher key involved in the check
1	Address of the DECB	0

Figure 39. Register Contents for DCB-Specified ISAM SYNAD Routine

You may also specify a SYNAD routine by way of the DD AMP parameter (see "JCL for Processing with the ISAM Interface" later in this chapter). Figure 40 gives the contents of registers 0 and 1 when a SYNAD routine specified by way of AMP gets control.

Reg.	BISAM	QISAM
0	Address of the DECB	0, or, for a sequence check, the address of a field containing the higher key involved in the check
1	Address of the DCB	Address of the DCB

Figure 40. Register Contents for AMP-Specified ISAM SYNAD Routine

If your SYNAD routine issues the SYNADAF macro, registers 0 and 1 are used to communicate. When you issue SYNADAF, register 0 must have the same contents it had when the SYNAD routine got control and register 1 must contain the address of the DCB.

When you get control back from SYNADAF, the registers have the same contents they would have if your program were processing an indexed-sequential data set: Register 0 contains a completion code, and register 1 contains the address of the SYNADAF message.

The completion codes and the format of a SYNADAF message are given in Data Management Macro Instructions.

Figure 41 shows abend codes issued by the ISAM interface when there is no other method of communicating the error to the user.

If a SYNAD routine specified by way of AMP issues the SYNADAF macro, the operand ACSMETH may specify either QISAM or BISAM, regardless of which of the two is used by your processing program.

A dummy DEB is built by the ISAM interface to support:

- References by the ISAM processing program
- Checkpoint/restart
- Abend

Figure 42 shows the DEB fields that are supported by the ISAM interface; field meanings are the same as in ISAM, except as noted.

ABEND Code	Error Detected By	DCB/DECB Set By Module/Routine	Abend Issued By	Error Condition
03B	OPEN	OPEN/OPEN ACB and VALID CHECK	OPEN	Validity check; either (1) access method services and DCB values for LRECL, KEYLE, and RKP do not correspond, (2) DISP=OLD, the DCB was opened for output, and the number of logical records is greater than zero (RELOAD is implied), or (3) OPEN ACB error code 116 was returned for a request to open a VSAM structure.
031	VSAM	SYNAD	SYNAD	SYNAD (ISAM) was not specified and a VSAM physical and logical error occurred.
	VSAM	SCAN/GET and SETL	SYNAD	SYNAD (ISAM was not specified and an invalid request was found.
	LOAD	LOAD/RESUME	LOAD	SYNAD (ISAM) was not specified and a sequence check occurred.
	LOAD	LOAD	LOAD	SYNAD (ISAM) was not specified and the RDW (record descriptor word) was greater than LRECL.
039	VSAM	SCAN/EODAD	SCAN	End-of-data was found, but there was no EODAD exit.
001	VSAM	SYNAD		I/O error detected.
	BISAM	SYNAD	BISAM	I/O error detected during check.
	BISAM	BISAM	BISAM	Invalid request.

Figure 41. Abend Codes Issued by the ISAM Interface

DEB Section	Bytes	Fields Supported
PREFIX	16	LNGTH
BASIC	32	TCBAD, OPATB, DEBAD, OFLGS (DISP ONLY), FLGS1 (ISAM-interface bit), AMLNG (104), NMEXT(2), PRIOR, PROTG, DEBID, DCBAD, EXSCL (0-DUMMY DEB), APPAD
ISAM Device	16	EXPTR, FPEAD
Direct Access	16	UCBAD (VSAM UCB)
Access Method	24	WKPT5 (ISAM-interface control block pointer), FREED (pointer to IDAIIFBF)

Figure 42. DEB Fields Supported by ISAM Interface

CONVERTING AN INDEXED-SEQUENTIAL DATA SET

Access method services is used to convert an indexed-sequential data set to a key-sequenced data set. Assuming that a master and/or user catalog has been defined, define a key-sequenced data set with the attributes and performance options you want. Then use the access method services REPRO command to convert the indexed-sequential records and load them into the key-sequenced data set. (See Catalog Users Guide for information about defining a key-sequenced data set and about converting an indexed-sequential data set.) VSAM builds the index for the key-sequenced data set as it loads the data set.

Each volume of a multivolume component must be on the same type of device; the data component and the index component, however, may be on volumes of devices of different types.

When you define the key-sequenced data set into which the indexed-sequential data set is to be copied, you must specify the attributes of the VSAM data set for variable- and fixed-length records.

For variable-length records:

- VSAM record length equals ISAM DCBLRECL-4.
- VSAM key length equals ISAM DCBKEYLE.
- VSAM key position equals ISAM DCBRKP-4.

For fixed-length records:

- VSAM record length (average and maximum must be the same) equals ISAM DCBLRECL (+ DCBKEYLE, if ISAM DCBRKP equals 0 and records are unblocked).
- VSAM key length equals ISAM DCBKEYLE.
- VSAM key position equals ISAM DCBRKP.

Care should also be taken with the level of sharing allowed when the key-sequenced data set is defined. If the ISAM program opens multiple DCBs pointing to different DD statements, a share-options value of 1, which is the default, allows only the first DD statement to be opened. See "Sharing" for a description of the share-options values.

JCL FOR CONVERTING FROM ISAM TO VSAM

JCL is used to identify data sets and volumes for allocation. Data sets can also be allocated dynamically. For a description of dynamic allocation, see JCL and System Programming Library: System Modifications.

If JCL is used to describe an indexed-sequential data set to be converted to VSAM using the access method services REPRO command, include DCB=DSORG=IS. The key-sequenced data set that is to receive the converted data set need not be described in JCL if it is to reside in a previously defined data space. If it is to reside alone in a data space, the data set is either allocated dynamically by name (in which case the volume on which it is to reside must be mounted). Use a STEPCAT or JOBCAT DD statement as described in the chapter "Chapter 7. Job Control Language" on page 121 to make user catalogs available; you may also use dynamic allocation.

With ISAM, deleted records are flagged as deleted, but are not actually removed from the data set. To avoid reading VSAM records that are flagged as deleted (X'FF'), code DCB=OPTCD=L. If your program depends upon a record's only being flagged and not actually removed, you may want to keep these flagged records when you convert and continue to have your programs process these records. The access method services REPRO command has a parameter (ENVIRONMENT) that causes VSAM to keep the flagged records when you convert.

JCL FOR PROCESSING WITH THE ISAM INTERFACE

To execute your ISAM processing program to process a key-sequenced data set, replace the ISAM DD card with a VSAM DD card using the DDNAME that was used for ISAM. The VSAM DD card names the key-sequenced data set and gives any necessary VSAM parameters (by way of AMP). Specify DISP=MOD for resume loading and DISP=OLD or SHR for all other processing. You don't have to specify anything about the ISAM interface itself. The interface is automatically brought into action when your processing program opens a DCB whose associated DD statement describes a key-sequenced data set (instead of an indexed-sequential data set). If you have defined your VSAM data set in a user catalog, specify the user catalog in a JOBCAT or STEPCAT DD statement.

The DCB parameter in the DD statement that identifies a VSAM data set is invalid and must be removed. If the DCB parameter is not removed, unpredictable results can occur. Certain DCB-type information may be specified in the AMP parameter, which is described later in this chapter.

Figure 43 shows the DCB fields supported by the ISAM interface.

Field Name	Meaning
BFALN	Same as in ISAM; defaults to a doubleword
BLKSI	Set equal to LRECL if not specified
BUFCB	Same as in ISAM
BUFL	The greater value of AMDLRECL or DCBLRECL if not specified
BUFNO	For QISAM, one; for BISAM, the value of STRNO if not specified
DDNAM	Same as in ISAM
DEBAD	During the DCB exit, contains the address of the OPEN work area; after the DCB exit, contains the address of the dummy DEB built by the ISAM interface
DEVT	Set from the VSAM UCB TYPE
DSORG	Same as in ISAM
EODAD	Same as in ISAM
ESETL	Address of the ISAM interface ESETL routine
EXCD1	See the QISAM exception codes
EXCD2	See the QISAM exception codes
EXLST	Same as in ISAM (except that VSAM does not support the JFCBE exit)
FREED	Address of the ISAM-interface dynamic buffering routine (IDAIIFBF)
GET/PUT	For QISAM LOAD, the address of the ISAM-interface PUT routine; for QISAM SCAN, 0, the address of the ISAM-interface GET routine; 4, the address of the ISAM-interface PUTX routine; and 8, the address of the ISAM-interface RELSE routine
KEYLE	Same as in ISAM
LRAN	Address of the ISAM-interface READ K/WRITE K routine
LRECL	Set to the maximum record size specified in the access method services DEFINE command if not specified (adjusted for variable-length, fixed, unblocked, and RKP=0 records)
LWKN	Address of the ISAM-interface WRITE KN routine
MACRF	Same as in ISAM
NCP	For BISAM, defaults to one
NCRHI	Set to a value of 8 before DCB exit
OFLGS	Same as in ISAM

Figure 43 (Part 1 of 2). DCB Fields Supported by ISAM Interface

Field Name	Meaning
OPTCD	Bit 0 (W), same as in ISAM; bit 3 (I), dummy records are not to be written in the VSAM data set; bit 6 (L), VSAM-deleted records (X'FF') are not read; dummy records are to be treated as in ISAM; all other options ignored
RECFM	Same as in ISAM; default to unblocked, variable-length records
RKP	Same as in ISAM
RORG1	Set to a value of 0 after DCB exit
RORG2	Set to a value of X'7FFFF' after DCB exit
RORG3	Set to a value of 0 after DCB exit
SETL	For BISAM, address of the ISAM-interface CHECK routine; for QISAM, address of the ISAM-interface SETL routine
ST	Bit 1 (key-sequence check), same as in ISAM; bit 2 (loading has completed), same as in ISAM
SYNAD	Same as in ISAM
TIOT	Same as in ISAM
WKPT1	For QISAM SCAN, WKPT1 +112=address of the W1CBF field pointing to the current buffer
WKPT5	Address of the ISAM-interface control block (IICB)
WKPT6	For QISAM LOAD, address of the dummy DCB work area vector pointers; the only field supported is ISLVPTRS+4=pointer to KEYSAVE

Figure 43 (Part 2 of 2). DCB Fields Supported by ISAM Interface

AMP PARAMETER SPECIFICATION

When an ISAM processing program is run with the ISAM interface, the AMP parameter enables you to specify:

- That a VSAM data set is to be processed (AMORG)
- The need for extra index buffers for simulating the residency of the highest level(s) of an index in virtual storage (BUFNI)
- The need for additional data buffers to improve sequential performance (BUFND)
- Whether to remove records flagged (OPTCD)
- What record format (RECFM) is used by the processing program
- The number of concurrent BISAM and QISAM (basic and queued indexed-sequential access methods) requests that the processing program may issue (STRNO)
- The name of an ISAM exit routine to analyze physical and logical errors (SYNAD)

The AMP parameter has some subparameters that are peculiar to the ISAM interface. The other subparameters of AMP (BUFSP, CROPS, and TRACE), which can also be used with the interface, are described in "Chapter 7. Job Control Language" on page 121. The format of the AMP parameter (with the subparameters discussed here) is:

```
//...        DD          [AMP=AMORG
                         [,'BUFND=number']
                         [,'BUFNI=number']
                         [,'OPTCD={I|L|IL}']
                         [,'RECFM={F|FB|V|VB}']
                         [,'STRNO=number']
                         [,'SYNAD=modulename']]
```

where:

AMORG
specifies that a VSAM data set is to be processed. When you specify unit and volume information for a DCB (through the ISAM interface program) or when you specify DUMMY in the DD statement, you must specify AMORG. Under these conditions, the system doesn't have to search a catalog to find out which volume(s) are required, and therefore doesn't know that the DD statement defines a VSAM data set. You never have to specify unit and volume information unless you want to have mounted some, but not all, of the data set's volumes, or you want to defer the volume mounting.

BUFND=number
specifies the number of I/O buffers VSAM is to use for data records. The minimum number you may specify is 1 plus the number specified for STRNO (if you omit STRNO, BUFND must be at least 2, because the default for STRNO is 1).

BUFNI=number
specifies the number of I/O buffers VSAM is to use for index records. If you don't specify BUFNI, VSAM uses as many index buffers as the number specified for STRNO (1 if you don't specify STRNO). You may specify for BUFNI a number 1 greater than STRNO (2 if you don't specify STRNO) to simulate having the highest level of an ISAM index resident. If you specify for BUFNI a number 2 or more greater than STRNO, you simulate having intermediate levels of the index resident.

OPTCD={I|L|IL}
specifies how records flagged for deletion are to be treated. The values that can be specified are:

L
specifies that a record marked for deletion by your processing program is to be kept in the data set. Although this parameter has the same meaning and restrictions for the ISAM interface as it has for ISAM, it may have to be specified in the AMP parameter when it wasn't previously needed in the ISAM job control language. It is required when OPTCD=L is not specified in the DCB in the processing program because OPTCD is not merged into the DSCB when the ISAM interface is used.

I
specifies, when coded along with OPTCD=L in the DCB, that records marked for deletion by your processing program are not written into the data set by the ISAM interface. If OPTCD=I is specified in the AMP parameter, but OPTCD=L isn't specified in the processing program's DCB, records flagged for deletion are treated as any other records: that is, AMP='OPTCD=I', without L anywhere specified, has no effect.

IL
specifies that, if your processing program writes a record marked for deletion, the ISAM interface is not to put the record into the data set. (It issues a VSAM ERASE to delete the old record if your processing program had previously read the record for update.) The result of this parameter is the same as when AMP='OPTCD=I' is coded with OPTCD=L in the DCB in the processing program.

RECFM={F|FB|V|VB}
specifies the ISAM record format that your processing program is coded for. Although this parameter has the same meaning and restrictions for the ISAM interface as it has for ISAM, it may have to be specified in the AMP parameter when it wasn't previously required in the ISAM job control language. RECFM is required when it is not specified in the DCB in the processing program because RECFM is not merged into the DSCB when the ISAM interface is used. All VSAM requests are for unblocked records. If your program issues a request for blocked records, the ISAM interface sets the overflow-record indicator for each record to indicate that each is being passed to your program unblocked. If RECFM isn't specified in the AMP parameter or in the processing program's DCB, V is the default.

STRNO=_number_
specifies the number of request parameter lists the processing program can use concurrently. Neither VSAM nor the ISAM interface can anticipate the number, so you should indicate it in the STRNO parameter. Specify a number at least equal to the number of BISAM and QISAM requests that your program can issue concurrently. (If you have subtasks, add together the number of such requests for each subtask, plus an additional one for each subtask that sequentially processes the same data set.) In a create step, STRNO cannot be greater than 1. The ISAM interface uses a request parameter list to describe a request that your program issues. The interface uses the same request parameter list over and over:

- With BISAM, a READ for update uses a request parameter list until a WRITE or FREEDBUF is issued (at which time the interface issues an ENDREQ for the request parameter list).

- With QISAM, a request parameter list is used until an ESETL is issued (at which time the interface issues ENDREQ).

If the processing program issues an ISAM request when no more request parameter lists are available, the ISAM interface returns an ISAM code that indicates an invalid request. If you're running subtasks, it's possible to reissue the invalid request and have it complete successfully when another subtask frees a request parameter list.

SYNAD=_modulename_
specifies the name of a routine that the ISAM interface is to load and exit to if a physical or logical error occurs when you are gaining access to the key-sequenced data set. If your processing program already indicates a SYNAD routine, the routine specified in the AMP SYNAD parameter replaces it.

The SYNAD routine must not issue VSAM macros or check for VSAM return codes. The ISAM interface translates all VSAM codes to appropriate ISAM codes.

You need not modify or replace a SYNAD routine that issues only a CLOSE, ABEND, SYNADAF, or SYNADRLS macro or that merely examines DCB or DECB exception codes.

RESTRICTIONS ON THE USE OF THE ISAM INTERFACE

Some restrictions were indicated earlier in this chapter that may require you to modify an ISAM processing program to process a key-sequenced data set. All operating system and VSAM restrictions apply to the use of the ISAM interface; for example:

- VSAM doesn't allow the OPENJ macro: If your program issues it, remove it or replace it with the OPEN macro.

- If your processing program was coded on the assumption that the indexed-sequential data set it was processing was a temporary data set, you may need to modify the program: A VSAM data set cannot be temporary.

Additional restrictions are:

- A program must run successfully under ISAM using standard ISAM interfaces; the interface doesn't check for parameters that are invalid for ISAM.

- If your DCB exit list contains an entry for a JFCBE exit routine, remove it. The interface doesn't support the use of a JFCBE exit routine. If the DCB exit list contains an entry for a DCB open exit routine, that exit is taken.

- If your ISAM program creates dummy records with a maximum key to avoid overflow, remove that code for VSAM.

- If your program counts overflow records to determine reorganization needs, its results will be meaningless with VSAM data sets.

- The work area into which data records are read must not be shorter than a record. If your processing program is designed to read a portion of a record into a work area, you must change the design. The interface takes the record length indicated in the DCB to be the actual length of the data record. The record length in a BISAM DECB is ignored, except when you are replacing a variable-length record with the WRITE macro.

- You may share data among subtasks that specify the same DD statement in their DCB(s), and VSAM ensures data integrity. But, if you share data among subtasks that specify different DD statements for the data, you are responsible for data integrity. The ISAM interface doesn't ensure DCB integrity when two or more DCBs are opened for a data set. All of the fields in a DCB cannot be depended on to contain valid information.

- When a data set is shared by several jobs (DISP=SHR), you must use the ENQ and DEQ macros to ensure exclusive control of the data set. Exclusive control is necessary to ensure data integrity when your program adds or updates records in the data set. You can share the data set with other users (that is, relinquish exclusive control) when reading records.

- If your processing program issues the SETL I or SETL ID instruction, you must modify the instruction to some other form of the SETL or remove it. The ISAM interface cannot translate a request that depends on a specific block or device address.

- Although asynchronous processing may be specified in an ISAM processing program, all ISAM requests are handled synchronously by the ISAM interface; WAIT and CHECK requests are always satisfied immediately. The ISAM CHECK macro doesn't result in a VSAM CHECK macro's being issued but merely causes exception codes in the DECB (data event control block) to be tested.

- For processing programs that use locate processing, the ISAM interface constructs buffers to simulate locate processing.

- For blocked-record processing, the ISAM interface simulates unblocked-record processing by setting the overflow-record indicator for each record. (In ISAM, an overflow record is never blocked with other records.) Programs that examine ISAM internal data areas (for example, block descriptor words (BDW) or the MBBCCHHR address of the next overflow record) must be modified to use only standard ISAM interfaces. The ISAM RELSE instruction causes no action to take place.

- If your ISAM SYNAD routine examines information that cannot be supported by the ISAM interface (for example, the IOB), specify a replacement ISAM SYNAD routine in the AMP parameter of the VSAM DD statement.

- Your ISAM program (on TSO) cannot dynamically allocate a VSAM data set (use LOGON PROC).

- CATALOG/DADSM macros in the ISAM processing program must be replaced with access method services commands.

- The ISAM interface uses the same RPL over and over, thus, for BISAM, a READ for update uses up an RPL until a WRITE or FREEDBUF is issued (at which time the interface issues an ENDREQ for the RPL). (When using ISAM you may merely issue another READ if you don't want to update a record after issuing a BISAM READ for update.)

- ISAM programs will run, with sequential processing, if the key length is defined as smaller than it actually is. This is not permitted with the ISAM interface.

- VSAM path processing is not supported by the ISAM interface.

- The ISAM interface does not support RELOAD processing. RELOAD processing is implied when an attempt is made to open a VSAM data set for output, specifying DISP=OLD, and, in addition, the number of logical records in the data set is greater than zero.

Example: Converting a Data Set

In this example, the indexed-sequential data set to be converted (ISAMDATA) is cataloged either in the system catalog or in a VSAM catalog. A key-sequenced data set, VSAMDATA, has previously been defined in user catalog USERCTLG. Because both the indexed-sequential and key-sequenced data set are cataloged, unit and volume information need not be specified.

ISAMDATA contains records flagged for deletion; these records are to be kept in the VSAM data set.

```
//CONVERT   JOB  ...
//JOBCAT    DD   DISP=SHR,DSNAME=USERCTLG
//STEP      EXEC PGM=IDCAMS
//SYSPRINT  DD   SYSOUT=A
//ISAM      DD   DISP=OLD,DSNAME=ISAMDATA,DCB=DSORG=IS
//VSAM      DD   DISP=OLD,DSNAME=VSAMDATA
//SYSIN     DD   *
     REPRO -
          INFILE(ISAM  ENVIRONMENT(DUMMY)) -
          OUTFILE(VSAM)
/*
```

To drop records flagged for deletion in the indexed-sequential data set, omit ENVIRONMENT(DUMMY).

Example: Issuing a SYNADAF Macro

The following example illustrates how a SYNAD routine specified by way of AMP may issue a SYNADAF macro without preliminaries—registers 0 and 1 already contain what SYNADAF expects to find.

```
AMPSYN   CSECT

         USING  *,15              Register 15 contains the entry
                                  address to AMPSYN.

         SYNADAF ACSMETH=QISAM    Either QISAM or BISAM may be
                                  specified.

         STM    14,12,12(13)

         BALR   7,0               Load address of next instruction
                                  into register 7 for base register.

         USING  *,7

         L      15,132(1)         The address of the DCB is stored
                                  132 bytes into the SYNADAF message.

         L      14,128(1)         The address of the DECB is stored
                                  128 bytes into the SYNADAF message.

         TM     42(15),X'40'      Does the DCB indicate QISAM scan?

         BO     QISAM             Yes.

         TM     43(15),X'40'      Does the DCB indicate QISAM load?

         BO     QISAM             Yes.

BISAM    TM     24(14),X'10'      Does the DECB indicate an invalid
                                  BISAM request?

         BO     INVBISAM          Yes.

.                                 The routine might print the SYNADAF
.                                 message or issue ABEND.
.
QISAM    TM     80(15),X'10'      Does the DCB indicate an invalid
                                  QISAM request?

         BO     INVQISAM          Yes.

.                                 The routine might print the SYNADAF
.                                 message or issue ABEND.
.
INVBISAM EQU    *

INVQISAM EQU    *

         LM     14,12,12(13)

         DROP   7

         USING  AMPSYN,15

         SYNADRLS

         BR     14

         END    AMPSYN
```

When the processing program closes the data set, the interface issues VSAM PUT macros for ISAM PUT locate requests (in load mode), deletes the interface routines from virtual storage, frees virtual-storage space that was obtained for the interface, and gives control to VSAM.

The following terms are defined as they are used in this book. If you do not find the term you are looking for, refer to the index or to the IBM Vocabulary for Data Processing, Telecommunications, and Office Systems, GC20-1699.

access method services. A multifunction service program that is used to define VSAM data sets and allocate space for them, convert indexed-sequential data sets to key-sequenced data sets, modify data set attributes in the catalog, reorganize data sets, facilitate data portability between operating systems, create backup copies of data sets, help make inaccessible data sets accessible, list the records of data sets and catalogs, define and build alternate indexes, and convert OS CVOLs and VSAM catalogs to ICF catalogs.

acquire. To allocate space on a staging drive and to stage data from an MSS cartridge to the staging drive.

addressed-direct access. The retrieval or storage of a data record identified by its RBA, independent of the record's location relative to the previously retrieved or stored record. (See also keyed-direct access, addressed-sequential access, and keyed-sequential access.)

addressed-sequential address. The retrieval or storage of a data record in its entry sequence relative to the previously retrieved or stored record. (See also keyed-sequential access, addressed-direct access, and keyed-direct access.)

alternate index. A collection of index entries organized by the alternate keys of its associated base data records. It provides an alternate means of locating records in the data component of a cluster on which the alternate index is based.

alternate key. One or more consecutive characters taken from a data record and used to build an alternate index or to locate one or more base data records via an alternate index. (See also generic key, key, and key field.)

alternate index cluster. The data and index components of an alternate index.

APF. (See authorized program facility.)

application. As used in this publication, the use to which an access method is put or the end result that it serves; contrasted to the internal operation of the access method.

authorized program facility. A facility that permits the identification of programs that are authorized to use restricted functions.

base cluster. A key-sequenced or entry-sequenced data set over which one or more alternate indexes are built.

base RBA. The RBA stored in the header of an index record that is used to calculate the RBAs of data or index control intervals governed by the index record.

BIND. (1) An attribute of a data set that keeps the data set on one or more MSS staging drives until the data set is released by the user regardless of the length of time or the demands for space. (2) An attribute of a mass storage volume that reserves an entire staging pack for the mass storage volume whenever the volume is mounted.

CA. (See channel adapter.)

catalog. (See master catalog and user catalog.)

catalog recovery area. (See CRA.)

CBIC. Control blocks in common, a facility that allows a user to open a VSAM data set so the VSAM control blocks are placed in the common service area (CSA) of the MVS operating system. This provides the capability for multiple memory accesses to a single VSAM control structure for the same VSAM data set.

chained RPL. (See RPL string.)

channel adapter. A communication controller hardware unit used to attach the controller to a System/360 or a System/370 data channel.

CI. (See control interval.)

CIDF. (See control interval definition field.)

CKDS. In the Programmed Cryptographic Facility, cryptographic key data set.

cluster. A named structure consisting of a group of related components (for example, a data component with its index component). A cluster may consist of a single component. (See also base cluster and alternate index cluster.)

collating sequence. An ordering assigned to a set of items, such that any two sets in that assigned order can be collated. As used in this publication, the order defined by the System/370 8-bit code for alphabetic, numeric, and special characters.

component. A named, cataloged collection of stored records. A component, the lowest member of the hierarchy of data structures that can be cataloged, contains no named subsets.

control area. A group of control intervals used as a unit for formatting a data set before adding records to it. Also, in a key-sequenced data set, the set of control intervals pointed to by a sequence-set index record; used by VSAM for distributing free space and for placing a sequence-set index record adjacent to its data.

control area split. The movement of the contents of some of the control intervals in a control area to a newly created control area, to facilitate the insertion or lengthening of a data record when there are no remaining free control intervals in the original control area.

control interval. A fixed-length area of auxiliary storage space in which VSAM stores records. It is the unit of information transmitted to or from auxiliary storage by VSAM.

control interval access. The retrieval or storage of the contents of a control interval.

control interval definition field. In VSAM, the 4-byte control information field at the end of a control interval that gives the displacement from the beginning of the control interval to free space and the length of the free space. If the length is 0, the displacement is to the beginning of the control information.

control interval split. The movement of some of the stored records in a control interval to a free control interval, to facilitate the insertion or lengthening of a record that won't fit in the original control interval.

control volume. A volume that contains one or more indexes of the catalog.

CRA. Catalog recovery area. An entry-sequenced data set that exists on each volume owned by a recoverable catalog, including the catalog itself. The CRA contains self-describing records that are duplicates of catalog records that describe the volume.

CVOL. (See control volume.)

cylinder fault. A condition that occurs when the operating system requires data that has not been staged. The cylinder fault causes a cylinder of data to be staged.

DASD. (See direct access storage device.)

data integrity. Preservation of data or programs for their intended purpose. As used in this publication, the safety of data from inadvertent destruction or alteration.

data record. A collection of items of information from the standpoint of its use in an application, as a user supplies it to VSAM for storage.

data security. Prevention of access to or use of data or programs without authorization. As used in this publication, the safety of data from unauthorized use, theft, or purposeful destruction.

data set. The major unit of data storage and retrieval in the operating system, consisting of data in a prescribed arrangement and described by control information to which the system has access. As used in this publication, a collection of fixed- or variable-length records in auxiliary storage, arranged by VSAM in key sequence or in entry sequence. (See also key-sequenced data set and entry-sequenced data set.)

data space. A storage area defined in the volume table of contents of a direct access volume for the exclusive use of VSAM to store data sets, indexes, and catalogs.

DD statement. data definition statement

DES. The United States National Bureau of Standards data encryption standard.

destage. To move data from a staging drive to a mass storage volume.

direct access. The retrieval or storage of data by a reference to its location in a data set rather than relative to the previously retrieved or stored data. (See also addressed-direct access and keyed-direct access.)

direct access storage device. A device in which the access time is effectively independent of the location of the data.

distributed free space. Space reserved within the control intervals of a key-sequenced data set for inserting new records into the data set in key sequence; also, whole control intervals reserved in a control area for the same purpose.

EBDIC. Extended binary-coded decimal interchange code. A coded character set consisting of 8-bit coded characters.

entry sequence. The order in which data records are physically arranged (according to ascending RBA) in auxiliary storage, without respect to their contents. (Contrast with key sequence.)

entry-sequenced data set. A data set whose records are loaded without respect to their contents, and whose RBAs cannot change. Records are retrieved and stored by addressed access, and new records are added at the end of the data set.

EOD. end of data

EOKR. end-of-key range

EOV. end of volume

ESDS. (See entry-sequenced data set.)

field. In a record or a control block, a specified area used for a particular category of data or control information.

free control interval entry. In a sequence-set index record, a vertical pointer that gives the location of a free control interval in the control area governed by the record.

front compression. The elimination, from the front of a key, of characters that are the same as the characters in the front of the preceding key.

GDG. (See generation data group.)

GENDSP. An option of LOCATE to obtain the control interval number of the catalog record of each object.

generation data group. A collection of data sets that are kept in chronological order; each data set is called a generation data set.

generic key. A high-order portion of a key, containing characters that identify those records that are significant for a certain application. For example, it might be desirable to retrieve all records whose keys begin with the generic key AB, regardless of the full key values.

global shared resources. An option for sharing I/O buffers, I/O-related control blocks, and channel programs among VSAM data sets in a resource pool that serves all address spaces in the system.

GSR. (See global shared resources.)

header, index record. In an index record, the 24-byte field at the beginning of the record that contains control information about the record.

header entry. In a parameter list of GENCB, MODCB, SHOWCB, or TESTCB, the entry that identifies the type of request and control block and gives other general information about the request.

horizontal pointer. In the header of an index record, the RBA of the index record in the same level as this one that contains keys next in ascending sequence after the keys in this one.

ICF. (See integrated catalog facility.)

index. As used in this publication, an ordered collection of pairs, each consisting of a key and a pointer, used by VSAM to sequence and locate the records of a key-sequenced data set.

index level. A set of index records that order and give the location of all the control intervals in the next lower level or in the data set that it controls.

index record. A collection of index entries that are retrieved and stored as a group. (Contrast to data record.)

index record header. In an index record, the 24-byte field at the beginning of the record that contains control information about the record.

index replication. The use of an entire track of direct access storage to contain as many copies of a single index record as possible; reduces rotational delay.

index set. The set of index levels above the sequence set. The index set and the sequence set together comprise the index.

integrated catalog facility. The name of the catalog associated with the Data Facility Product program product.

ISAM. indexed sequential access method

ISAM interface. A set of routines that allow a processing program coded to use ISAM (indexed sequential access method) to gain access to a key-sequenced data set.

JCL. (See job control language.)

job catalog. A catalog made available for a job by means of the JOBCAT DD statement.

job control language. A problem-oriented language designed to express statements in a job that are used to identify the job or describe its requirements to an operating system.

job step catalog. A catalog made available for a job by means of the STEPCAT DD statement.

key. One or more characters within an item of data that are used to identify it or control its use. As used in this publication, one or more consecutive characters taken from a data record, used to identify the record and establish its order with respect to other records. (See also key field and generic key.)

key compression. The elimination of characters from the front and the back of a key that VSAM does not need to distinguish the key from the preceding or following key in the index record; reduces storage space for an index.

key field. A field located in the same position in each record of a data set, whose contents are used for the key of a record.

key sequence. The collating sequence of data records, determined by the value of the key field in each of the data records. May be the same as, or different from, the entry sequence of the records.

key-sequenced data set. A data set whose records are loaded in key sequence and controlled by an index. Records are retrieved and stored by keyed access or by addressed access, and new records are inserted in the data set in key sequence by means of distributed free space. RBAs of records can change.

keyed-direct access. The retrieval or storage of a data record by use of either an index that relates the record's key to its relative location in the data set or a relative record number, independent of the record's location relative to the previously retrieved or stored record. (See also addressed-direct access, keyed-sequential access, and addressed-sequential access.)

keyed-sequential access. The retrieval or storage of a data record in its key or relative record sequence relative to the previously retrieved or stored record, as defined by the sequence set of an index. (See also addressed-sequential access, keyed-direct access, and addressed-direct access.)

key-sequenced data set. A VSAM file (data set) whose records are loaded in key sequence and controlled by an index. Records are retrieved and stored by keyed access or by addressed access, and new records are inserted in key sequence by means of distributed free space. Relative byte addresses of records can change, because of control interval or control area splits.

KSDS. (See key-sequenced data set.)

level number. For the index of a key-sequenced data set, a binary number in the header of an index record that indicates the index level to which the record belongs.

local shared resources. An option for sharing I/O buffers, I/O-related control blocks, and channel programs among VSAM data sets in a resource pool that serves one partition or address space.

LSR. (See local shared resources.)

Mass Storage System. The name for the entire storage system, consisting of the Mass Storage Facility and all devices that are defined to the Mass Storage Control. The abbreviation is MSS.

master catalog. A catalog that contains extensive data set and volume information that VSAM requires to locate data sets, to allocate and deallocate storage space, to verify the authorization of a program or operator to gain access to a data set, and to accumulate usage statistics for data sets.

operating system. Software that controls the execution of programs; an operating system may provide services such as resource allocation, scheduling, input/output control, and data management.

OS. (See operating system.)

password. A unique string of characters stored in a catalog that a program, a computer operator, or a terminal user must supply to meet security requirements before a program gains access to a data set.

path. A named, logical entity composed of one or more clusters (an alternate index and its base cluster, for example).

physical record. A physical unit or recording on a medium. For example, the physical unit between address markers on a disk.

pointer. An address or other indication of location. For example, an RBA is a pointer that gives the relative location of a data record or a control interval in the data set to which it belongs.

portability. The ability to use VSAM data sets with different operating systems. Volumes whose data sets are cataloged in a user catalog can be demounted from storage devices of one system, moved to another system, and mounted on storage devices of that system. Individual data sets can be transported between operating systems using access method services.

prestage. To move data from an MSS cartridge to a staging drive before the data is needed by the processing program.

prime index. The index component of a key-sequenced data set that has one or more alternate indexes. (See also index and alternate index.)

prime key. (See key.)

QSAM. (See queued sequential access method.)

queued sequential access method. An extended version of the basic sequential access method (BSAM). When this method is used, a queue is formed of input data blocks that are awaiting processing or output data blocks that have been processed and are awaiting transfer to auxiliary storage or to an output device.

RACF. Resource Access Control Facility.

random access. (See direct access.)

RBA. Relative byte address. The displacement (expressed as a fullword binary integer) of a data record or a control interval from the beginning of the data set to which it belongs; independent of the manner in which the data set is stored.

RDF. (See record definition field.)

rear compression. The elimination, from a key, of characters to the right of the first character that is unequal to the corresponding character in the following key.

record. (See index record, data record, stored record.)

record definition field. A field stored as part of a stored record segment; it contains the control information required to manage stored record segments within a control interval.

recoverable catalog. A catalog defined with the recoverable attribute. Duplicate catalog entries are put into CRAs that can be used to recover data in the event of catalog failure. (See also CRA.)

relative byte address. (See RBA.)

relative record data set. A data set whose records are loaded into fixed-length slots.

relative record number. A number that identifies not only the slot, or data space, in a relative record data set but also the record occupying the slot. Used as the key for keyed access to a relative record data set.

replication. (See index replication.)

resource pool, VSAM. (See VSAM resource pool.)

reusable data set. A VSAM data set that can be reused as a work file, regardless of its old contents. Must not be a base cluster.

RPL string. A set of chained RPLs (the set may contain one or more RPLs) used to gain access to a VSAM data set by action macros (GET, PUT, etc). Two or more RPL strings may be used for concurrent direct or sequential requests made from a processing program or its subtasks.

RRDS. (See relative record data set.)

RRN. A number (expressed as a fullword binary integer) which represents the position of a record in a relative record data set. The record is located in the data set based on its relative record number.

SAM. (See sequential access method.)

security. (See data security.)

sequence checking. The process of verifying the order of a set of records relative to some field's collating sequence.

sequence set. The lowest level of the index of a key-sequenced data set; it gives the locations of the control intervals in the data set and orders them by the key sequence of the data records they contain. The sequence set and the index set together comprise the index.

sequential access. The retrieval or storage of a data record in either its entry sequence, its key sequence, or its relative record number sequence, relative to the previously retrieved or stored record. (See also addressed-sequential access and keyed-sequential access.)

sequential access method. An access method for storing or retrieving data blocks in a continuous sequence, using either a sequential access or a direct access device.

shared resources. A set of functions that permit the sharing of a pool of I/O-related control blocks, channel programs, and buffers among several VSAM data sets open at the same time.

skip-sequential access. Keyed-sequential retrieval or storage of records here and there throughout a data set, skipping automatically to the desired record or collating position for insertion: VSAM scans the sequence set to find a record or a collating position. Valid for processing in ascending sequences only.

slot. For a relative record data set, the data area addressed by a relative record number which may contain a record or be empty.

spanned record. A logical record whose length exceeds control interval length, and as a result, crosses, or spans, one or more control interval boundaries within a single control area.

stage. To move data from an MSS cartridge to a staging drive.

step catalog. A catalog made available for a step by means of the STEPCAT DD statement.

stored record. A data record, together with its control information, as stored in auxiliary storage.

terminal monitor program. In TSO, a program that accepts and interprets commands from the terminal, and causes the appropriate command processors to be scheduled and executed.

time sharing option. An optional configuration of the operating system that provides conversational time sharing from remote stations.

TMP. (See terminal monitor program.)

transaction ID. A number associated with each of several request parameter lists that define requests belonging to the same data transaction.

TSO. (See time sharing option.)

update number. For a spanned record, a binary number in the second RDF of a record segment that indicates how many times the segments of a spanned record should be equal. An inequality indicates a possible error.

upgrade set. All the alternate indexes that VSAM has been instructed to update whenever there is a change to the data component of the base cluster.

user buffering. The use of a work area in the processing program's address space for an I/O buffer; VSAM transmits the contents of a control interval between the work area and direct access storage without intermediary buffering.

user catalog. An optional catalog used in the same way as the master catalog and pointed to by the master catalog. It also lessens the contention for the master catalog and facilitates volume portability.

vertical pointer. A pointer in an index record of a given level that gives the location of an index record in the next lower level or the location of a control interval in the data set controlled by the index.

virtual volume. The data from a mass storage volume while it is located on a staging drive.

virtual storage access method. An access method for direct or sequential processing of fixed and variable-length records on direct access devices. The records in a VSAM data set or file can be organized in logical sequence by a key field (key sequence), in the physical sequence in which they are written on the data set or file (entry sequence), or by relative record number.

VSAM. (See virtual storage access method.)

VSAM resource pool. A virtual storage area that is used to share I/O buffers, I/O-related control blocks, and channel programs among VSAM data sets. A resource pool is local or global; it serves tasks in one partition or address space or tasks in all address spaces in the system.

VSAM shared information. Blocks that are used for cross-system sharing.

VSI. (See VSAM shared information.)

INDEX

GC26-4066-0

MVS/370 VSAM Users Guide (File No. S370-30) Printed in U.S.A. GC26-4066-0

MVS/370 VSAM Users Guide
GC26-4066-0

Reader's Comment Form

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

List TNLs here:

If you have applied any technical newsletters (TNLs) to this book, please list them here:

Last TNL ____________________

Previous TNL ________________

Previous TNL ________________

Fold on two lines, tape, and mail. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.) Thank you for your cooperation.

GC26-4066-0

Reader's Comment Form

Fold and tape Please do not staple Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
P.O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and tape Please do not staple Fold and tape

IBM®

MVS/370 VSAM Users Guide (File No. S370-30) Printed in U.S.A. GC26-4066-0

MVS/370 VSAM Users Guide
GC26-4066-0

Reader's Comment Form

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

List TNLs here:

If you have applied any technical newsletters (TNLs) to this book, please list them here:

Last TNL ____________________

Previous TNL ____________________

Previous TNL ____________________

Fold on two lines, tape, and mail. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.) Thank you for your cooperation.

GC26-4066-0

Reader's Comment Form

Fold and tape | Please do not staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
P.O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and tape | Please do not staple | Fold and tape

IBM®

MVS/370 VSAM Users Guide (File No. S370-30) Printed in U.S.A. GC26-4066-0