GC26-3838 -3
File No. S370-30

Systems

OS/VS Virtual Storage Access Method (VSAM) Programmer's Guide

VS1 Release 6
VS2 Release 3.7

Includes Selectable Units:

MVS Supervisor Performance #2	**VS2.03.807**
MVS Data Management	**VS2.03.808**

IBM

USING THIS PUBLICATION

This publication describes the use of VSAM (Virtual Storage Access Method), an access method of OS/VS (Operating System/Virtual Storage). It is intended for Assembler language programmers who intend to use VSAM macro instructions to process data and for higher-level language programmers who may want to use ISAM programs to process data. The publication has the following major divisions:

- "Introduction," which introduces basic VSAM concepts that the reader needs in order to use VSAM.
- "Processing Options and Considerations," which describes the types of access that can be used with VSAM.
- "Sharing a VSAM Data Set," which 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.
- "Optimizing VSAM's Performance," which describes many of the concepts and parameters that influence VSAM performance.
- "Job Control Language," which describes JCL for VSAM, including DD statements for catalogs and the DD AMP parameter.
- "Macro Instruction Descriptions and Return Codes," which briefly describes the macros used to open and close a data set, manage control blocks, and issue data management requests. This chapter also contains all the macro return codes.
- "Macro Instruction Formats and Examples," which describes the syntax of each macro and includes coded examples of each one.
- "Using ISAM Programming with VSAM," which describes how data sets can be converted to VSAM's format and can be processed using an ISAM processing program.
- "User-Written Exit Routines," which describes how to write the exit routines that can be used with VSAM.
- "Appendix A: Summary of Macros," which summarizes, for ease of reference, the format of the macros used to communicate with VSAM.
- "Appendix B: List, Execute, and Generate Forms of GENCB, MODCB, SHOWCB, and TESTCB," which explains how to code reentrant programs with the macros that generate, modify, test, and display control blocks at execution.
- "Appendix C: Operand Notation for GENCB, MODCB, SHOWCB, and TESTCB," which gives all of the ways of coding operands in the macros that generate, modify, test, and display control blocks at program execution time.
- "Glossary," which defines VSAM terms.
- "Index," which 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, only one of which may be specified.
- An underlined item, item, is the default when you don't specify anything for an operand.
- Ellipses, ..., indicate that you may repeat the preceding item.
- Capitalized **BOLD** expressions, parentheses, commas, and equal signs must be entered as shown, except that, unless otherwise noted, parentheses aren't required if you specify only one item.
- Lowercase *italic* expressions are variables for which you may specify one of a number of expressions.

VSAM and Access Method Services Publications

This publication makes frequent references to the VSAM and Access Method Services publications. Rather than list all the titles at each reference point, the text will refer to the *appropriate publication* for the particular subject.

The VSAM and Access Method Services publications are:

- *OS/VS Virtual Storage Access Method (VSAM) Options for Advanced Applications,* GC26-3819, which provides information about advanced applications of VSAM. The topics, which do not apply to normal use of VSAM, 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; and displaying fields of the catalog.
- *OS/VS1 Access Method Services,* GC26-3840, and *OS/VS2 Access Method Services,* GC26-3841, which contain a complete description of the commands that are used to copy, print, and load data sets. They also describe the relationships among components, the structure of components, the use of the catalog, and Access Method Services commands that are used to define and delete data sets, list catalog entries, and move data sets from one operating system to another. These publications are directed to the person responsible for establishing and maintaining data sets in an installation.
- *OS/VS1 Access Method Services Logic,* SY35-0008, and *OS/VS2 Access Method Services Logic,* SY35-0010, which describe the internal logic of Access Method Services.
- *OS/VS1 Virtual Storage Access Method (VSAM) Logic,* SY26-3841, which describes the internal logic of VSAM and of VSAM catalog management.
- *OS/VS2 Catalog Management Logic,* SY26-3826, which describes the internal logic of MVS VSAM catalog management.

- *OS/VS2 Virtual Storage Access Method (VSAM) Logic,* SY26-3825, which describes the internal logic of VSAM, excluding VSAM catalog management.

Other Required Publications

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

- *OS/VS1 Data Management Services Guide,* GC26-3874, and *OS/VS2 MVS Data Management Services Guide,* GC26-3875, which presents basic concepts such as access method. direct-access storage, and the distinction between data-set organization and data-set processing.
- *OS/VS1 JCL Reference,* GC24-5099, which describes the JCL parameters for VS1 referred to in this publication.
- *OS/VS2 JCL,* GC28-0692, which describes the JCL parameters for VS2 referred to in this publication and describes dynamic allocation.
- *OS/VS2 System Programming Library: Job Management,* GC28-0627, which describes dynamic allocation.

Related Publications

The reader may need to be familiar with some of the information presented in the following publications:

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

Other publications referred to in this publication are:

- *OS/VS1 Checkpoint/Restart,* GC26-3876
- *OS/VS2 MVS Checkpoint/Restart,* GC26-3877
- *OS/VS Message Library: VS1 System Messages,* GC38-1001
- *OS/VS Message Library: VS2 System Messages,* GC38-1002
- *OS/VS1 Utilities,* GC26-3901
- *OS/VS2 MVS Utilities, GC26-3902*
- *OS/VS1 System Data Areas,* SY28-0605
- *OS/VS2 Data Areas,* SYB8-0606
- *OS/VS2 System Programming Library: Debugging Handbook Volume 1,* GC28-0708, and *Volume 2,* GC28-0709. (Both manuals may be ordered using GBOF-8211)
- *OS/VS2 MVS Resource Access Control Facility (RACF) General Information Manual, GC28-0722*

CONTENTS

FIGURES

OS/VS1 SUMMARY OF AMENDMENTS

Service and Editorial Changes

- Control Interval Split Interruption: Before VSAM splits a control interval, VSAM writes the control interval to the direct-access device with the CIDF "busy flag" set. When VSAM completes the control interval split, VSAM resets the busy flag. Whenever a control interval with a busy flag is accessed, VSAM detects a previous control interval split failure. VSAM attempts to remove any duplicated records from that control interval.
- A new chapter, "Sharing a VSAM Data Set," has been added, describing VSAM share options.
- A description of the OPEN/CLOSE/TCLOSE Message Area has been included.
- The section "Buffer Space"has been expanded to include more detail about buffer management.
- Numerous other service and editorial changes have been made throughout the book.

Release 6

- A new keyword, HALCRBA, can be specified in the FIELDS operand of the SHOWCB-ACB macro. This keyword allows a user to determine the high-allocated RBA for either the data or index component.
- UPAD, a new exit routine, allows the user to perform special processing during a VSAM request.

Release 5

There are no system changes for this release; however, the book has been reorganized and new chapters and topics have been added.

New Chapters and Reorganization

- A new chapter, "Optimizing VSAM's Performance," describes control interval and control area sizes and how they are determined, buffer management, distributed free space, index options, the SPEED and RECOVERY options, and performance measurement.
- A new chapter, "Processing Options and Considerations," includes some processing information from the previous edition and also some new subjects, such as multi-string processing, record insertion, deferred and forced writing of buffers, and key ranges.
- All macro instructions, regardless of type, are presented in alphabetic order in a new chapter, "Macro Instruction Formats and Examples."
- All return codes, regardless of type, are described in the chapter "Macro Instruction Descriptions and Return Codes."

Miscellaneous Chang

Release 4

New Programming Support

The IBM 3850 Mass Storage System (MSS) is supported with this release. The MSS virtual volumes are functionally equivalent to the 3330/3333 Disk Storage, Model 1. For information on MSS, see *OS/VS Mass Storage System (MSS) Planning Guide,* GC35-0011.

New VSAM Functions and Data Structures

Support for the new functions and data structures listed below has been added to this publication.

- Multiple indexes enable subsets of a single data set to be uniquely named, mounted, and processed.
- Control interval size is no longer dependent upon the size of the largest record in a data set. Logical records may span control intervals within a single control area.
- New options allow the sharing of I/O-related control blocks, channel programs, and buffers among several VSAM data sets open at the same time. Definitive descriptions of these options are published in *OS/VS Virtual Storage Access Method (VSAM) Options for Advanced Applications.*
- Relative record data sets, a new type of data organization, permit the arithmetic calculation of the control interval containing a required record and of the record's position within the control interval.
- GET-previous processing, a variation of sequential retrieval, returns the previous record (relative to current positioning) rather than the next record.
- Improved control-interval processing is an optional performance enhancement that reduces the time and the number of CPU instructions required to gain access to a control interval. This processing is described in *OS/VS Virtual Storage Access Method (VSAM) Options for Advanced Applications.*
- Enhanced space reclamation makes it possible to reuse a single data set many times as a work file.

Service and Editorial Changes

- Control Interval Split Interruption: Before VSAM splits a control interval, VSAM writes the control interval to the direct-access device with the CIDF "busy flag" set. When VSAM completes the control interval split, VSAM resets the busy flag. Whenever a control interval with a busy flag is accessed, VSAM detects a previous control interval split failure. VSAM attempts to remove any duplicated records from that control interval.
- A new chapter, "Sharing a VSAM Data Set," has been added, describing VSAM share options.
- A description of the OPEN/CLOSE/TCLOSE Message Area has been included.
- The section "Buffer Space"has been expanded to include more detail about buffer management.
- Numerous other service and editorial changes have been made throughout the book.

OS/VS2 MVS Data Management (VS2.03.808)

Alternate Key Support

Feedback code 08 (paired with the 0 indicator in register 15) has been changed. For GET requests, the code indicates that a duplicate key follows; for PUT requests, it indicates that a duplicate key was created in an alternate index with the nonunique attribute.

OS/VS2 MVS Supervisor Performance #2 (VS2.03.807)

Provides a description of Resource Access Control Facility (RACF), an IBM program product that allows additional access-control measures on MVS system only.

For a complete list of publications that support Supervisor Performance #2, see *OS/VS2 MVS Supervisor Performance #2 System Information,* GC28-0727.

Release 3.7

The Enhanced VSAM functions and data structures, available previously as an independent component, are a part of this release. In addition, this publication has been reorganized and new chapters and topics have been added.

Enhanced VSAM

- Multiple indexes enable subsets of a single data set to be uniquely named, mounted, and processed.
- Control interval size is no longer dependent upon the size of the largest record in a data set. Logical records may span control intervals within a single control area.
- New options allow the sharing of I/O-related control blocks, channel programs, and buffers among several VSAM data sets open at the same time. Definitive descriptions of these options are published in *OS/VS Virtual Storage Access Method (VSAM) Options for Advanced Applications.*
- Relative record data sets, a new type of data organization, permit the arithmetic calculation of the control interval containing a required record and of the record's position within the control interval.
- GET-previous processing, a variation of sequential retrieval, returns the previous record (relative to current positioning) rather than the next record.
- Improved control interval processing is an optional performance enhancement that reduces the time and the number of CPU instructions required to gain access to a control interval. This processing is described in *OS/VS Virtual Storage Access Method (VSAM) Options for Advanced Applications.*
- Enhanced space reclamation makes it possible to reuse a single data set many times as a work file.
- A new keyword, HALCRBA, can be specified in the FIELDS operand of the SHOWCB-ACB macro. This keyword allows a user to determine the high-allocated RBA for either the data or the index component.
- UPAD, a new exit routine, allows the user to perform special processing during a VSAM request.
- Control Blocks in Common (CBIC) option (MVS only). This option allows a user to place the VSAM control blocks associated with a VSAM data set into the Common Service Area (CSA) of MVS. This option is described in *OS/VS Virtual Storage Access Method (VSAM) Options for Advanced Applications.*

New Chapters and Reorganization

- A new chapter, "Optimizing VSAM's Performance," describes control interval and control area sizes and how they are determined, buffer management, distributed free space, index options, the SPEED and RECOVERY options, and performance measurement.
- A new chapter, "Processing Options and Considerations," includes some processing information from the previous edition and also some new subjects, such as multi-string processing, record insertion, deferred and forced writing of buffers, and key ranges.
- All macro instructions, regardless of type, are presented in alphabetic order in a new chapter, "Macro Instruction Formats and Examples."
- All return codes, regardless of type, are described in the chapter "Macro Instruction Descriptions and Return Codes."

Return Code Changes

- Two new OPEN return codes have been added:

  200(C8)—Format-4 indication of an unusable volume

  236(EC)—MSS staging error
- One new CLOSE return code has been added:

  236 (EC)—MSS destaging error
- One new control block return code has been added:

  21(15)—Block to be displayed or tested does not exist because the data set is a dummy data set
- Two new FDBK codes have been added:

  120(78)—Request was issued under an incorrect TCB

  208(D0)—ENDREQ was issued against an RPL that has an outstanding WAIT
- A table has been added that describes (for each FDBK code) whether VSAM was able to maintain positioning.
- A table has been added that indicates (for various OPTCDs) the return code you can expect when a search argument is greater than the highest key in the data set.

Miscellaneous Changes

- Exit routines that are used by a program doing asynchronous processing with multiple RPLs must be able to handle entries made before the previous entry's processing is complete. See "User-Written Exit Routines."
- Descriptions of the ACB MACRF options have been expanded.
- Examples have been added to illustrate: (1) keyed-sequential retrieval in a backward (OPTCD=BWD) mode, (2) addressed-sequential retrieval of records in a relative record data set, (3) retrieval of records by way of an alternate index path with the nonunique key option, (4) creating a relative record data set using SKP and DIR, and (5) keyed-sequential insertion of records into a relative record data set.

Release 3

New Programming Support

The IBM 3850 Mass Storage System (MSS) is supported with this release. MSS virtual volumes are functionally equivalent to the 3330/3333 Disk Storage, Model 1. For information on MSS, See *OS/VS Mass Storage (MSS) Planning Guide,* GC35-0011.

There are no significant VSAM changes in this release.

INTRODUCTION

VSAM gives you both 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.

With VSAM you can build one or more alternate indexes over a single base data set, so that you need not keep multiple copies of the same information organized in different ways for different applications. In terms of access, an alternate index serves the same purpose as a primary index, but it does not account for space in the base data set and does not require unique keys.

The Access Method Services commands are used to establish and maintain data sets and to copy and print data sets. These commands are described in the appropriate Access Method Services publication.

The macros described in this publication include control block macros and request macros. The macros that are used to build control blocks, described in the chapter "Macro Instruction Descriptions and Codes," are:

- ACB, which is used to build an access-method control block at assembly time.
- EXLST, which is used to build an exit list, which identifies available exit routines; the exit list is built at assembly time.
- RPL, which is used to build a request parameter list at assembly time.
- GENCB, which is used to build an access-method control block, an exit list, or a request parameter list at execution time.

The macros that are used to modify, display, and test the contents of control blocks, also described in the chapter "Macro Instruction Descriptions and Codes," are:

- MODCB, which is used to modify an access-method control block, an exit list, or a request parameter list at execution time.
- SHOWCB, which is used to display fields in an access-method control block, an exit list, or a request parameter list at execution time.
- TESTCB, which is used to test the contents of fields in an access-method control block, an exit list, or a request parameter list at execution time.

The macros used to store, retrieve, and erase records, to position VSAM in a data set, to suspend processing, and to terminate requests, described in the chapter "Macro Instruction Descriptions and Return Codes," are:

- GET, which is used to retrieve a record.
- PUT, which is used to store a record.
- ERASE, which is used to delete a record.
- POINT, which is used to position VSAM at a record.
- CHECK, which is used to suspend processing.
- ENDREQ, which is used to terminate a request.

Types of Data Sets

VSAM has key-sequenced, entry-sequenced, and relative record data sets. The primary difference among the three is the sequence in which data records are loaded into them.

Records are loaded into a *key-sequenced data set* in key sequence: that is, 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, such as employee number or invoice number, in the key field. To determine where to insert a new record into the data set in key sequence, VSAM uses an index that pairs the key of a record with the record's location.

Records are loaded into an *entry-sequenced data set* without respect to the contents of the records. Their sequence is determined by the order in which they are stored: their entry sequence. Each new record is stored after the last record in the data set.

Records are loaded into a *relative record data set* in relative record number sequence, or you can supply the relative record number of each record and load them in random order. The data set may be described as a string of fixed-length slots, each of which is identified by a relative record number. When a new record is sequentially inserted, VSAM assigns the record either the next available relative record number in sequence or the number you supplied.

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.

When you define a VSAM data set, your data set may be either *unique* or *suballocated.* A unique data set occupies an entire VSAM data space. This space is acquired and assigned for the data set concurrent with the data set definition. A suballocated data set, on the other hand, is a data set which gets its space from a previously defined VSAM data space. There can be several suballocated VSAM data sets in a VSAM data space.

Any suballocated VSAM data set that does not have an alternate index and is not associated with key ranges may be used as a work file. 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.

All VSAM data sets are stored on direct-access storage devices. The records of a data set need not be stored in a continuous area of storage. From your point of view, the area is continuous, starting at address 0. VSAM addresses a point in the area by its displacement, in bytes, from 0, called its *RBA (relative byte address).* 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. RBAs are independent of a data set's being stored in nonadjacent areas on a volume or on several volumes.

All VSAM data sets must be cataloged in a VSAM catalog. See *Planning for Enhanced VSAM under OS/VS* for a description of the catalog.

The total space of a data set is considered to be divided into a continuous set of areas called control areas, which are further divided into *control intervals.* When you retrieve a record, the contents of the control interval in which it is stored are read in by VSAM. A control interval is thus the unit of data transmission between virtual and auxiliary storage.

Key-sequenced and entry-sequenced data records whose lengths exceed control interval size may cross, or span, one or more control interval boundaries within a single control area. Such records are called *spanned records*. You must specify your intent to use spanned records when you define the data set.

The following sections briefly describe VSAM data sets and alternate indexes. For detailed information about these data structures, see *Planning for Enhanced VSAM under OS/VS*.

Key-Sequenced Data Set

A key-sequenced data set always includes an index, which is a mechanism for keeping track of records. An index relates key values to the relative locations of the records.

The RBAs of records can change in a key-sequenced data set when records are added, deleted, shortened, or lengthened.

A key-sequenced data set permits the full range of options for gaining access to data: retrieval, insertion, deletion, and changing the length of a record. Records in a key-sequenced data set can be accessed using keyed access, addressed access, and control interval access. VSAM keeps track of records in a key-sequenced data set by key field, so that you need refer to a record only by its key field, and not in some location-dependent manner.

A key-sequenced data set must be created in sequential mode.

Entry-Sequenced Data Set

The records in an entry-sequenced data set are in the order they are stored in time. That is, each new record is stored at the end; none is inserted. Records cannot be shortened, lengthened, or moved from one location to another. Records cannot be deleted, although they can be replaced with records of the same length. Once a record is added to an entry-sequenced data set, it stays there and keeps its original RBA. An entry-sequenced data set is essentially a sequential data set, but one whose records can be retrieved at random by direct access and can be updated. The search argument for direct retrieval is a record's RBA.

An entry-sequenced data set is appropriate for applications that require no special ordering of data by the contents of a record. It is appropriate for a log or a journal, because its order corresponds to the chronology of events. To retrieve records randomly from an entry-sequenced data set, you must keep track of the records' RBAs and associate RBAs with the contents of records.

Relative Record Data Set

A relative record data set has no index. It is a string of fixed-length slots, each of which is identified by a relative record number from 1 to n, where n is 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.

Records in a relative record data set are grouped together in control intervals, just as they are in a key-sequenced or an entry-sequenced data set. Each control interval contains the same number of slots. The size of each slot is the record length you specified when you defined the data set.

Relative record data sets can be processed by key or by control interval. With keyed access, a relative record number is treated like a key. You can update records in place, delete records, and insert new records into empty slots. Control-interval processing is described in *OS/VS Virtual Storage Access Method (VSAM) Options for Advanced Applications.*

You can use a relative record data set in much the same way you would use a BDAM (basic direct-access method) data set in which the data records are not ordered by their contents or their entry sequence.

Figure 1 compares key-sequenced, entry-sequenced, and relative record 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 the 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, is unique, or exceeds 16 extents per volume	Can be reused as a work file unless it has an alternate index or exceeds 16 extents per volume	Can be reused as a work file unless it exceeds 16 extents per volume

Figure 1. Comparison of Key-Sequenced, Entry-Sequenced, and Relative Record Data Sets

Alternate Indexes

An alternate index provides a unique way to gain access to a related base data set, so that you need not keep multiple copies of the same information organized in different ways for different applications. For example, a payroll data set indexed by employee number can also be indexed by other fields such as employee name or department number. See the appropriate Access Method Services publication for a complete description of the commands used to define and build an alternate index.

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.* The base cluster can be a key-sequenced or an entry-sequenced data set, but not a relative record or a reusable data set.

Alternate-Index Clusters

The alternate index is an indexed cluster (the *alternate-index 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 cluster. Likewise, the format of the alternate-index data component is identical to the base data set. Therefore, each entry in the sequence set of an alternate-index index component points to a control interval in the alternate-index data component.

When building an alternate index, you can use as the *alternate key* any field in the base data set's records having a fixed length and a fixed position within 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 alternate key then the alternate index record contains a pointer to each base data record. These duplicate, or *nonunique* keys are discussed in the section "Alternate-Index Keys."

Alternate-Index Paths

A *path* logically relates a base 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 you can give it a password, if you choose. The path name subsequently refers to the base cluster/alternate-index pair. This means that when you refer to a path (by way of the OPEN macro, for example), both the base cluster and the alternate index are affected (opened).

Alternate-Index Records

Each record in the data component of an alternate-index cluster is variable-length and contains system header information, the alternate key, and at least one pointer to a base data record. Data component records may span control intervals.

System Header Information

System header information is fixed length and indicates:

- Whether the alternate index record contains (1) prime keys or RBA pointers and (2) unique or nonunique keys
- The length of each pointer
- The length of the alternate key
- The number of pointers

Alternate-Index Keys

Unless the base data records span control intervals, any field in the base data record 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 ordered 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. An alternate index key 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. That is, the entire key is represented in the alternate-index 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. In other words, there will be several prime-key pointers (employee numbers) 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, multiple pointers are associated with it. The pointers are ordered by their arrival times. That is, if a base data record is updated with a key change, or if a new record is inserted with the same alternate key value 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 record length for spanned records is not exceeded.

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 alternate indexes or you can have VSAM maintain them. When the data set is defined with the UPGRADE attribute, VSAM will update 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 and updates them if necessary.

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, any modifications made to the base data or to another alternate index are nullified, and the request that caused the upgrade is rejected.

If you specify NOUPGRADE 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, the base cluster and all the alternate indexes in the upgrade set are allocated. If updating an alternate index is unnecessary, you can specify NOUPDATE when you define the path. In this case, only the base cluster is allocated and VSAM does no automatic upgrading.

PROCESSING OPTIONS AND CONSIDERATIONS

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, say, 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. See the chapter "Control Block Macros" for a description of the macro instructions and of the specification of the processing options.

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

VSAM can keep track concurrently 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 positoning 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 suballocated 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. 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) 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 *OS/VS Virtual Storage Access Method (VSAM) Options for Advanced Applications.*

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 the chapter "Request Macros" later in this publication.

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 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 (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 non-update 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 the chapter "Sharing a VSAM Data Set."

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 non-error return code in the RPL. If the alternate index has the UNIQUE 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, since 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 set is created (loaded), 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-seqeunced 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.

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

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

  1) Open the empty data set for create mode processing
  2) 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.

- The only request macros allowed during create mode are PUT and CHECK.
- ICI (Improved Control Interval) processing is not permitted.
- Create mode must be terminated via CLOSE before the data set can be used for other processing.
- You cannot specify more than one string in the ACB (STRNO>1 is not permitted).
- LSR (Local Shared Resources) or GSR (Global Shared Resources) cannot be specified.
- 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:

- Logical error (LERAD), which is used when the processing program makes an invalid request for access to data.
- Physical error (SYNAD), which is used to handle physical-error conditions.
- Exception handling (EXCEPTIONEXIT), which monitors physical-error conditions on a data set basis. This exit is specified via the Access Method Services DEFINE command and it is taken before a SYNAD exit if both are specified.
- End of data set (EODAD), which is used when the processing program has attempted to point to or retrieve sequentially a record beyond the last record in the data set.
- Journalizing a transaction or keeping track of RBA change (JRNAD), which 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), which 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. See the chapter "Control Block Macros" for information on how the exit list is created, modified, tested, and displayed. See the chapter "User-Written Exit Routines" for detailed information about the exit routines. For information about exception exits, see the appropriate Access Method Services publication.

Deferred and Forced Wr

i
v
w
ta
ba
fol

Force
Defer
An E

Record Insertions

Record

Type I
Type II
Type III
Type IV

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 of the other types use the direct insert strategy. Note that 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, then 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 that 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 that when referring to an index or data control interval,

VSAM checks its buffers to see if the control interval is already present. 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 has control of any record in a control interval, that 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.

Since VSAM doesn't 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 quiesce activity and clear the conflict. If the RPL that encountered 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 reissuing the request or (2) issuing an ENDREQ against the RPL holding exclusive control to force it to release control immediately.

Multi-String Index Buffers

Each string requires one index buffer. If there are four strings active, then there will be 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 non-sequential strings, if the entire high-level index won't conveniently fit in storage.

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+STRNO'=9, where STRNO' is the number of non-sequential strings. It is usually best to round this number up to the next 4K multiple. That is, BUFNI=12 (12K of index buffers). If you wanted

to keep the entire high-level index in storage, then BUFNI would set to 1+50+4=55. This would be rounded to BUFNI=56 and would require 56K of index buffers.

Multi-String Data Buffers

One data buffer per string, plus 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 will use one data and one index buffer. The sequential string will use one index and three data buffers. There will be one data buffer reserved for insert requests that cause a control interval split, and the index set will use the extra five index buffers.

Request Positioning

Some operations will retain positioning while others will release it. In a similar way, some operations will hold onto a buffer and others will release it with its contents. The following table shows you which 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)
Buffers and Positioning Retained	X	X	X	
Buffers and Positioning Released				X

Notes:

- GET SEQ for new control intervals releases the previous buffer
- The ENDREQ and ERASE macros release data buffers and positioning.
- Certain options which retain positioning and buffers upon normal completion may not do so if the request fails with an error code. Refer to the table in the section "FDBK Codes (Logical Errors)" to determine whether or not positioning will be maintained in the case of a logical error.

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

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

Utility Functions Carried Out by Access Method Services

Access Method Services is a multifunction service program that is used to define a VSAM data set and load records into it, convert a sequential or an indexed-sequential data set to the VSAM format, list VSAM catalog information or data-set records, copy a data set for reorganization, create a backup copy of a data set, recover from certain types of damage to a data set, and make a data set portable from one operating system to another. Definitive descriptions of all Access Method Services commands are in *OS/VS1 Access Method Services* and *OS/VS2 Access Method Services.*

Processing a VSAM Data Set with an ISAM Program

VSAM provides an interface program that permits you to use programs coded to use ISAM (indexed-sequential access method) to process VSAM data sets. To use the ISAM interface, you must convert indexed-sequential data sets to VSAM data sets, convert ISAM JCL to VSAM JCL, and ensure that your existing ISAM programs meet the restrictions for using the interface.

To convert an indexed-sequential data set to a VSAM data set that you can process either with an ISAM program by way of the ISAM interface or with a VSAM program, you use Access Method Services to define a key-sequenced data set in a VSAM catalog and allocate space for it. You may use an ISAM program by way of the ISAM interface to load records into the data set, or you may use Access Method Services REPRO command. For more details about the procedure, see the chapter "Using ISAM Programming with VSAM."

Using the Time Sharing Option (TSO) with VSAM

TSO is a subsystem of OS/VS2 that provides conversational time sharing from remote terminals. You can use TSO with VSAM to:

- Execute Access Method Services commands directly as TSO commands (in MVS only).
- Execute a program to process a VSAM data set.
- Execute a program to call Access Method Services.
- Dynamically allocate a VSAM data set and use VSAM macros to process the data set (in MVS only).
- Allocate a VSAM data set by way of a LOGON procedure and use either VSAM or ISAM macros to process the data set.

For details about writing and executing programs and allocating data sets with TSO, see *OS/VS2 TSO Terminal User's Guide*, and *OS/VS2 TSO Command Language Reference.* For information about dynamic allocation, see *OS/VS2 System Programming Library: Job Management.*

SHARING A VSAM DATA SET

A VSAM data set can be shared by different operating systems, by different jobs in a single operating system, and by different subtasks in an address space or partition. There are guidelines for accessing shared data sets and provisions for controlling data sharing that are intended to prevent the loss of data.

When you define a data set, you can select the level of sharing you intend to allow for it. 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 might result when one or more of the data set's users do not adhere to recommended guidelines 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 that you have exclusive control over the control interval until it is written back to the data set. However, 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't ensure that your program has exclusive control over the data set and you must obtain exclusive control yourself. (One way of obtaining exclusive control is by using ENQ and DEQ macros.)

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

- The SHAREOPTIONS value, specified when the VSAM data set is defined (using the Access Method Services DEFINE command), which indicates the level at which the data set can be shared by users in the same operating system (cross-region sharing) and by users in different operating systems (cross-system sharing).
- The use of the DISP=SHR and DISP=OLD parameters in the DD statement that identifies the data set to be opened.
- The type of processing (specified with the ACB's MACRF field) for which the data set is opened.
- Your program's use of the ENQ and DEQ macros to obtain exclusive control of the data set within your program's operating system.
- The ability of the data set's direct-access device to be accessed by more than one processor (and, consequently, by more than one operating system). If the data set's direct-access device can be shared between two or more operating systems, your program should use the RESERVE and RELEASE macros to obtain exclusive control of the device.

If the VSAM data set cannot be shared and is not available when your program issues the OPEN macro, the request is terminated and a return code is set in the ACB's ERROR field. If the VSAM data set can be opened, the Open routine issues an ENQ macro to gain exclusive control over all components of the data set.

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

- In a shared environment, VSAM does not allow you to process the data set in create or reset mode. (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.
- The user's program must serialize all VSAM requests against the data set, using the ENQ and DEQ macros (or a similar function).
- The user's program must use the ENDREQ macro to:
  - Force VSAM to immediately write sequential update and insert requests.
  - Release VSAM's positioning within the data set before releasing the data set from exclusive control via the DEQ macro.
- VSAM invalidates buffers used with SHAREOPTION 4 data sets, but does not invalidate buffers used with SHAREOPTION 3 data sets. When a buffer is marked "invalid" (that is, it is invalidated), it is identified as a buffer that VSAM must refresh (that is, read in a fresh copy of the control interval) before your program can use the buffer's contents.
- The GSR or LSR user's program can invalidate and force writing of buffers using the MRKBFR and WRTBFR macros. (See *OS/VS Virtual Storage Access Method (VSAM) Options for Advanced Applications* for more information on the MRKBFR and WRTBFR macros.)
- When the data set is shared under cross-system or cross-region SHAREOPTION 4, VSAM doesn't allow the data set's control areas to be split. Instead, VSAM returns a logical error control to the user's program, which the user's program can use to detect that a control area split was unsuccessful.

When a shared data set is opened with DISP=OLD, or is opened for create or reset processing, the SHAREOPTION values specified in the data set's catalog record are ignored. The data set is processed as though it were defined with SHAREOPTIONS(1,3). VSAM provides your program with exclusive control of the data set within your region. If the data set can be shared between operating systems, a user's program in another system might concurrently access the data set. Before you open the data set with DISP=OLD, you can issue the RESERVE macro to prevent another system's user from accessing the direct-access device containing your data set. After your program closes the data set, you can issue the RELEASE macro to relinquish exclusive control of the device.

Subtask Sharing

Subtasks within a region can share a data set by identifying it with a single DD statement, or by using a separate DD statement for each subtask using the data set.

When separate DD statements are used and the ACB DSN option is not specified, and one or more subtasks intend to perform output processing, each DD statement must specify DISP=SHR. In addition, the cross-region share option specified must be consistent with the level of sharing to be performed. Each subtask should adhere to the guidelines that apply to the specified cross-region share option.

When a single DD statement is used, or when separate DD statements are used and the ACB's DSN option is specified, several subtasks can update the data set concurrently, independently of the DISP specification. If DISP=SHR is specified for the data set, subtask sharing and cross-region sharing can occur concurrently.

Subtask Sharing in a Single Control Block Structure

A data set can be shared by many subtasks, each of which accesses the data set through a single control block structure or each of which has a separate control block structure. The following discussion applies only when the data set is described with a single control block structure.

When your program issues a GET-for-update request to be followed by a PUT-update request, VSAM obtains exclusive control over the control interval that will contain the record. VSAM also ensures that your program obtains exclusive control over the control interval will contain the record when your program issues a PUT request. When a subtask has exclusive control of a control interval, another subtask's GET-for-update and PUT requests for that control interval are refused. The other subtask's requests can be reissued after the first subtask releases exclusive control.

VSAM relinquishes control over the control interval when the controlling subtask accesses a record that is outside the control interval or issues an ENDREQ request.

When a subtask issues GET requests, the data set can be shared so that other subtasks are allowed to issue GET and PUT requests. When a subtask issues update or output requests, however, VSAM ensures that the control interval being updated is not accessed by other subtasks. Subtasks that share the data set are prevented from updating it until the update request completes; the subtasks are allowed to read the data set.

Cross-Region Sharing

Independent job steps in an operating system 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 SHAREOPTION value:

- Cross-region SHAREOPTION 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 SHAREOPTION 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 by obtaining exclusive control for a control interval when it is to be updated. If a user desires read integrity, it is his responsibility to use the ENQ and DEQ macros appropriately to provide read integrity for the data his program obtains.
- Cross-region SHAREOPTION 3: The data set can be fully shared by any number of users. With this option, each user is responsible for maintaining both read and write integrity for the data his 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 responsibility on each user sharing the data set.
- Cross-region SHAREOPTION 4: The data set can be fully shared by any number of users and buffers used for *direct processing* are refreshed for each request. This option *requires* your program to use the ENQ and DEQ macros to maintain data integrity while sharing the data set. Improper use of the ENQ macro can cause problems similar to those described under SHAREOPTION 3. When a shared data set is opened with DISP=SHR, VSAM doesn't allow:
  - PUT requests which cause the data set's control areas to be split
  - PUT requests which add to the end of the data set
  - PUT requests which add to the end of a key range

  If the above conditions are encountered, VSAM returns a logical error code to the user's program indicating the data set may not be extended.

Read Integrity During Cross-Region Sharing

The user is responsible for ensuring read integrity when the data set is opened for sharing with cross-region SHAREOPTION 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 might also obtain a copy of the same control interval, and might 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.

When your program requires that no updating occur before it completes processing the requested data record, your program can issue the ENQ macro 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.) When your program completes processing, it can relinquish control of the data set with a DEQ macro. When your program is only reading data and not updating it, your program usually won't require that updating activity by other users of the data set cease.

Your program might issue a GET request while another program issues a PUT request that results in a control area or control interval split. If this occurs, your GET request might result in a "no record found". If you share a VSAM data set with other users, your program should retry the GET request (on a "no record found" error) before issuing an error message.

Write Integrity During Cross-Region Sharing

The user is responsible for ensuring write integrity if a data set is opened with cross-region SHAREOPTION 3 or 4. When your program issues a GET-for-update request or a PUT request, VSAM doesn't usually write your program's copy of the control interval into the data set immediately. Another program sharing the data set might 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 should ensure that there is no activity against the data set until your program completes updating the control interval. Your program can issue the ENQ macro to obtain exclusive control over the VSAM data set. When your program completes updating the control interval's data, it can issue an ENDREQ macro to force writing the control interval. Your program can then relinquish control over the data set with a DEQ macro.

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

- 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.

In addition to serializing requests:

- When your program issues sequential GET and PUT requests it must issue the ENDREQ macro to force VSAM to write each sequential PUT-update and PUT-insert request, and to release VSAM's positioning within the data set before your program releases control of the data set.
- When your program processes a shared data set asynchronously, you must issue the CHECK macro to ensure that your I/O request has completed before you release control of the data set.

Cross-System Sharing

Job steps of two or more OS/VS operating systems can gain access to the same VSAM data set regardless of the disposition specified in each step's DD statement for the data set. To get exclusive control of the data set's volume, a task in one system issues the RESERVE macro. The level of cross-system sharing allowed by VSAM applies only in a multiple operating system environment. It is established with the SHAREOPTION value when the data set is defined:

- Cross-system SHAREOPTION 3: The data set can be fully shared. With this option, each user is responsible for maintaining both read and write integrity for the data his program accesses. User programs that ignore write-integrity guidelines can cause VSAM program checks, uncorrectable data set failures, and other unpredictable results. This option places heavy responsibility on each user sharing the data set.
- Cross-system SHAREOPTION 4: This option is only valid if the data set to be shared resides on a shared DASD device. The data set can be fully shared. Buffers used for *direct processing* are refreshed for each request. This option *requires* that you use the RESERVE and RELEASE macros to maintain data integrity while sharing the data set. Control area splits within a data set are not permitted. Writing is limited to PUT-update and PUT-insert processing that does not change the high-used RBA if your program opens the data set with DISP=SHR. A PUT request which causes the high-used RBA to change will return a logical error code to the user indicating the data set cannot be extended. Data set integrity cannot be maintained unless all jobs accessing the data set in a cross-system environment specify DISP=SHR. Improper use of the RESERVE macro can cause problems similar to those described under SHAREOPTION 3.

OPTIMIZING VSAM'S PERFORMANCE

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 *OS/VS1 Access Method Services* and *OS/VS2 Access Method Services.* 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

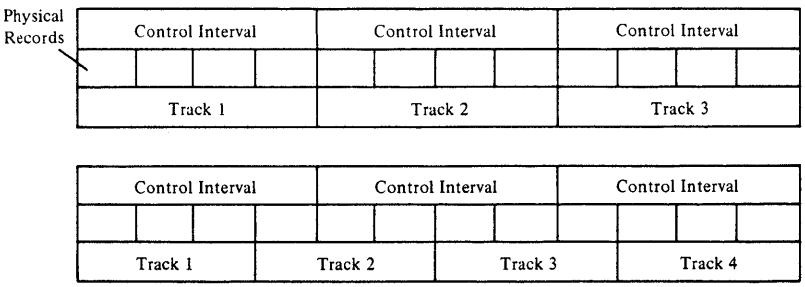
Control interval size, which can be specified for VSAM data sets, affects record-processing speed and storage requirements in these ways:

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

In the first place, 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 4096 bytes does not apply to the IBM 2314/2319 Disk Storage; 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, 10240, 12288, ..., 32768. 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.

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 4096 bytes does not apply to the IBM 2314/2319 Disk Storage; 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. The figure below illustrates the independence of control intervals from physical records.



The table below shows the physical blocksize selected by VSAM for all of the possible control interval sizes for the IBM 2314, 3330, 3340, and 3350 direct-access devices, together with the resulting track utilization (all numbers are in K-bytes):

	Physical Blocksize				Track Space Used			
CI Size	2314	3340	3330	3350	2314	3340	3330	3350
.5	.5	.5	.5	.5	5.5	6	10	13.5
1	1	1	1	1	6	7	11	15
1.5	.5	.5	.5	.5	5.5	6	10	13.5
2	2	1	2	2	6	7	12	16
2.5	.5	.5	.5	.5	5.5	6	10	13.5
3	1	1	1	1	6	7	11	15
3.5	.5	.5	.5	.5	5.5	6	10	13.5
4	2	4	4	4	6	8	12	16
4.5	.5	.5	.5	.5	5.5	6	10	13.5
5	1	1	1	1	6	7	11	15
5.5	.5	.5	.5	.5	5.5	6	10	13.5
6	2	1	2	2	6	7	12	16
6.5	.5	.5	.5	.5	5.5	6	10	13.5
7	1	1	1	1	6	7	11	15
7.5	.5	.5	.5	.5	5.5	6	10	13.5
8	2	4	4	4	6	8	12	16
10	2	1	2	2	6	7	12	16
12	2	4	4	4	6	8	12	16
14	2	1	2	2	6	7	12	16
16	2	4	4	4	6	8	12	16
18	2	1	2	2	6	7	12	16
20	2	4	4	4	6	8	12	16
22	2	1	2	2	6	7	12	16
24	2	4	4	4	6	8	12	16
26	2	1	2	2	6	7	12	16
28	2	4	4	4	6	8	12	16
30	2	1	2	2	6	7	12	16
32	2	4	4	4	6	8	12	16

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 recordsize 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 will provide for buffers.

BUFFERSPACE, if you specify it, limits control interval size to values such that the buffer space can hold at lease two data control intervals and one index control interval. If you don't 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 intervals, 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 systemdetermines 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 of 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 will be reasonably good regardless of the DASD device used, processing patterns, or CPU model. There are some special considerations that might affect this choice.

Data Space Utilization

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 data buffer space, it is better to read two 8K control intervals with one I/O operation than four 4K control intervals with two I/O operation. After insertions have occurred, very large

data control intervals often result in fewer out-of-sequence control intervals than small control intervals than do small control intervals.

The table that follows summarizes the generally acceptable data control interval sizes.

Accessing Pattern	Condition or Device	Data CI Size
Random	3340	1024
	2314	1024 or 2048
	3330	1024 or 2048
Sequential	3340	4096 or 8192
	2314	4096 or 6144
	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

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 big 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 big 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 will result in poor DASD space utilization: more control information than fixed length records and free space that cannot be used.

Keep in mind that if you specify the UNIQUE attribute in your DEFINE command for a key-sequenced cluster (indicating that the cluster is to uniquely occupy its own data space) VSAM allocates a whole cylinder to each area in the data component. 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 your 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 bufferspace, 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 and if your data space is nonunique, 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 will decrease 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 secondary allocation. The size of 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, or if the data set is defined as unique (the only one in its data space), 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.

Aside from specifying space in terms of cylinders or defining a data set as unique, 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 that 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 your primary and secondary space requirements without overcommitting space for this data set.

An index record must be large enough to address all of 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. Since an index record can contain only so many entries, more index records, and more important, probably more index levels will be 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, one-third of the direct access storage space is required for sequence sets. If the control area on a 3340 is a cylinder, only one-twelfth the DASD space is required. If the data control interval size is 4K, the smaller control area would force another level of indexing at 232, rather than 1276 control intervals.

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; therefore, control interval size is limited by the size of I/O buffers. When you define a data set, you can specify enough buffer space for the control intervals in the data set to be large enough for your largest stored record. 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 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 that require 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 Key-Sequenced Data Set

VSAM allocates buffers to a key-sequenced data set according to the following parameters and values:

- The amount of space for buffers (in the catalog record, established when the data set is defined):

  BUFFERSPACE = n

- Buffer and processing options specified for the data set (with the ACB in the user's program):

  MACRF = (SEQ | DIR | SKP)
  STRNO = n
  BUFSP = n
  BUFND = n
  BUFNI = n

- Buffer and processing options specified for the data set (with the AMP parameter in the user's JCL DD statement):

  STRNO = n
  BUFSP = n
  BUFND = n
  BUFNI = n

When a key-sequenced VSAM data set is opened, VSAM determines the size and number of data and index buffers to allocate for use when the data set is accessed. The process used by VSAM to determine this is described below.

1. VSAM determines the current values for BUFND, BUFNI, and BUFSP:

   A. If the parameters are specified with JCL (using the AMP parameter), VSAM uses the JCL-specified value. Otherwise,

   B. If the parameters are specified with the ACB, VSAM uses the ACB-specified value. Otherwise,

   C. Via catalog values specified at define time. Otherwise,

   D. VSAM assigns default values:

      If STRNO is not specified, its default is 1.

      BUFND = STRNO + 1
      BUFNI = STRNO
      BUFSP is not assigned a value at this time.

2. VSAM determines the minimum amount of space (MINSPACE) needed to support the data set:

   A. VSAM determines the minimum number of data buffers (MINBUFD) and index buffers (MINBUFI):

      MINBUFD = STRNO + 1
      MINBUFI = STRNO

   B. VSAM adjusts the values of BUFND and BUFNI, so that

      BUFND is the larger of BUFND or MINBUFD, and
      BUFNI is the larger of BUFNI or MINBUFI.

   C. VSAM calculates the minimum buffer space requirement (MINSPACE) based on data control interval size (CISIZE.DATA) and index control interval size (CISIZE.INDEX):

      MINSPACE = (MINBUFD × CISIZE.DATA) + (MINBUFI × CISIZE.INDEX)

3. VSAM also determines the requested amount of buffer space (REQSPACE), using the values of BUFND and BUFNI as adjusted in step 2B:

   REQSPACE = (BUFND × CISIZE.DATA) + (BUFNI × CISIZE.INDEX)

4. The user can specify the amount of buffer space he wants allocated by using the BUFSP parameter with JCL or with the ACB. When the data set is defined, the user can specify an amount of buffer space with the BUFFERSPACE parameter. If neither BUFSP nor BUFFERSPACE is specified, VSAM obtains space and allocates buffers based on the REQSPACE value.

5. If the user specified BUFFERSPACE and did not specify BUFSP:

   A. And if REQSPACE is less than BUFFERSPACE, VSAM increases the number of buffers (see step 7), obtains the space, and allocates the buffers based on the BUFFERSPACE value. Otherwise,

   B. VSAM obtains space and allocates buffers based on the REQSPACE value.

6. If the user specified BUFFERSPACE and BUFSP, or BUFSP alone:

   A. VSAM adjusts the value of BUFSP so that BUFSP is the larger of BUFSP or BUFFERSPACE.

   B. If BUFSP is smaller than MINSPACE, VSAM terminates processing with an error (ERROR code = 136). Otherwise, BUFSP is larger than MINSPACE, and

   C. VSAM compares BUFSP to REQSPACE:

      If BUFSP is larger than REQSPACE, VSAM increases the number of buffers (see step 7), obtains the space, and allocates buffers based on the BUFSP value.

      If BUFSP is equal to REQSPACE, VSAM obtains the space and allocates the buffers based on the BUFSP value.

      If BUFSP is smaller than REQSPACE, VSAM reduces the number of buffers (see step 8), obtains the space, and allocates buffers based on the BUFSP value.

7. **Increase number of buffers.** When calculating the buffer space requirements for the data set, VSAM determined that the user allowed for more space than was actually needed for the BUFND and BUFNI values (see step 3). Therefore, VSAM increases the BUFND or BUFNI values to account for the extra space:

   A. If the data set is being opened for direct processing (that is, SEQ=OFF and SKP=OFF), VSAM increases BUFNI until the amount of space used for the data and index buffers equals (as nearly as possible) BUFSP.

      The effect of this is to support the data set for direct processing with as many index buffers as possible, so that as much of the index as possible is in virtual storage.

B. If the data set is being opened for sequential or skip sequential processing:

- VSAM increases BUFNI by 1.
- VSAM increases BUFND until the amount of space used for the data and index buffers equals (as nearly as possible) BUFSP.

The effect of this is to support the data set with as many data buffers as possible, to maximize the number of control intervals accessed during a single I/O operation.

8. **Reduce number of buffers.** When calculating the buffer space requirements for the data set, VSAM determined that the user allowed enough space for minimum buffer requirements, but not enough space for the number of data and index buffers he specified. Consequently, VSAM reduces the BUFND and BUFNI values until the amount of bufferspace is equal to (as nearly as possible) BUFSP:

   A. If the data set is being opened for direct processing only (that is, SEQ=OFF and SKP=OFF), VSAM reduces BUFND until either:

   - The space required for buffers equals (as nearly as possible) BUFSP, or
   - BUFND = MINBUFD (the minimum number of data buffers needed for the data set)

   If BUFND = MINBUFD and BUFSP is still less than the space needed for buffers, VSAM reduces BUFNI until the space required for buffers equals (as nearly as possible) BUFSP.

   B. If the data set is being opened for sequential or skip sequential processing:

   - VSAM reduces BUFNI until either:

     The space required for buffers equals (as nearly as possible) BUFSP, or

     BUFNI = MINBUFI + 1

   - If BUFNI = MINBUFI + 1 and BUFSP is still less than the space needed for buffers, VSAM reduces BUFND until either:

     The space required for buffers equals (as nearly as possible) BUFSP, or

     BUFND = MINBUFD

   - If BUFSP is still less than the space needed for buffers, VSAM reduces BUFNI by 1. The values of BUFND and BUFNI are now at their minimum and can be accommodated by BUFSP.

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 might 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) allows for more buffers (VSAM assumes direct processing and extra buffers are allocated between data and index components accordingly).

If the path's alternate index is a member of the upgrade set, the minimum buffer allocation is increased 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. Two data buffers and one index buffer are always allocated to each alternate index in the upgrade 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 and increase BUFND to three if paging is not a problem. For straight sequential processing environments, start with four data buffers.
- 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 will be used to support the read-ahead function. When shareoption 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 SHR(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 data buffers are needed.
- For optimum operation, specify the number of index buffers equal to the number of index set control intervals plus one to contain the entire index set and one sequence set control interval in virtual storage. 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 aim for 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 might 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 both sequential and direct 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 an assembler-language coded program 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.

Bufferspace is aligned on page boundaries. VSAM checks to determine if an index or data control interval is in storage before rereading it, unless shareoption 4 is specified for the data set.

When processing the data set sequentially, VSAM will read ahead and provide 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. POINT SEQ causes read-ahead processing. When POINT completes, only one data buffer is filled.

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 in storage. Index buffers are loaded when the index is referred to. When many index buffers are provided, index control intervals are not reread until a desired index control interval is not in storage. If you provide as many index buffers as there are index control intervals (assuming a non-imbedded index), the data set's entire index 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/alternate index data, and 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 imbedded sequence set records. Assuming 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-imbedded 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:

  CYL(5,10)—Results in a 1-cylinder control area

  TRK(100,3)—Results in a 3-track control area

  REC(2000,5)—Assuming 10 records would fit on a track, results in a 1-track control area (minimum control area is 1 track)

  TRK(3,100)—Results in a 3-track control area

To force VSAM to select cylinder control areas:

- Define the data set using the CYLINDERS parameter or
- Define the data set as unique 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.

- If allocation is specified at the cluster/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 imbedded 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 index set extents.
- A data set with the unique attribute can have a maximum of 16 extents per volume.
- A spanned record cannot be longer than a control area less the control information (10 bytes per control interval), so don't specify large spanned records and small primary or secondary allocation.
- 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.

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 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 increases 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 will be placed in a control interval. Also, if the control area percentage free space is not zero, but is less than one control interval, the result is one free control interval in the control area.

Since a control interval contains logical records, free space, and control information (CIDFs and RDFs), a 4K control interval cannot contain four 1K logical records. A 4K control interval with (25,0) free space specified will contain at least 1K free space. Only two 1K fixed-length records could be loaded in the control interval, and only one more 1K 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.

Let
NDS = original data set size
GDS = grown data set size
PCTG = growth percentage = (GDS-NDS)/NDS
HALFG = PCTG/2
CICA = control intervals per control area
RCI = records per control interval = (CISZ-10)/RECSZ
(fixed length records)
FSPC1 = free space within control interval percentage
FSPC2 = free space within control area percentage
NCI = number of free control intervals per control area
NREC = number of free records per control interval
CEIL = Result rounded up to the nearest integer
FLOOR = Result rounded down to the nearest integer

Determine: NDS and GDS
Find: HALFG,CICA, and RCI
Compute:

```
NCI=CEIL(HALFG*CICA)
FSPC2=CEIL(NCI*100/CICA)
X=((PCTG*CICA)-(FSPC2*CICA/100))/CICA
NREC=CEIL(X*RCI)
FSPC1=FLOOR(NREC/RCI)
```

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.

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 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 ten copies of an index record fit on a track, average rotational delay is only one-twentieth of the time it takes for the volume to complete one revolution.

On an IBM 3340, the time usually is reduced by 50%. On a 3330 and a 2314, the time is reduced to 1/n, where n is the number of times the index is replicated on the track.

Since 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. Since 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 imbedded 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 one-twelfth of the data space on a 3340, but only one-nineteenth of the data space on a 3330. IMBED must be specified explicitly to get the performance benefits of a replicated, imbedded sequence-set.

Index Option Summary

On a 3340, place the index and data on separate volumes and do not replicate or imbed. Provide index buffer space to hold the entire index set plus one sequence set when doing random processing. If direct-access storage space is not a problem, if index and data cannot be put on different volumes, or if index buffer space is not available, specify REPL IMBED for random processing.

On a 2314 or a 3330, arbitrarily specify REPL IMBED and ensure that the allocation unit is in cylinders.

The SPEED and RECOVERY Options

The SPEED and RECOVERY options may improve performance when a data set is loaded. RECOVERY preformats (clears to zeros) each control area before any records are loaded into it and allows you to find the software end-of-file if an abnormal termination occurs during data set load. RECOVERY is most useful when your program has a recovery procedure that allows you to resume loading after a system failure.

SPEED reduces the number of writes required when records are initially loaded into a data set, and since data set load is probably not a significant part of your total processing time, it is normally a good choice. If you specify SPEED and a system failure does occur, the data set can be deleted, redefined, and loaded again from the beginning.

VSAM Catalogs

Sharing Services With User Catalogs

A large number of concurrent requests for information (that is, for catalog entries) from a VSAM catalog might result in some of the requests being answered more slowly than they would be if the entries were distributed among several user catalogs. You might have the VSAM master catalog primarily contain pointers to user catalogs, which would contain entries for most data sets, indexes, and volumes. Be decentralizing data set entries, you also reduce the time required to search a given catalog and minimize the effect of a catalog's being inoperative or unavailable.

Improving Catalog Peformance in MVS

To improve catalog performance in an MVS system, you can:

- Mount the catalog volume on an unsharable direct-access device.
- Define entries into a catalog that is not protected with an update-level password. This results in greatly improved performance when you are using the CNVTCAT command to convert OS catalog entries to VSAM catalog entries.
- Specify a large buffer space value when defining your catalog. This is an important factor in catalog performance, especially if there are many concurrent users. The value specified via BUFFERSPACE helps determine the number of catalog RPLs to be set up at catalog OPEN time. Insufficient RPLs cause subsequent users to have to wait until an RPL is available. Also, a large buffer space can result in catalog index control intervals staying in main storage, thus avoiding I/O to the index. Note that maximum buffer value is 8096.

For more details about catalog performance and operation, see the *Access Method Services* publication for MVS.

Performance Measurement

VSAM keeps statistical information about a data set in its catalog record. Some statistics, such as number of extents in a data set, number of records retrieved, added, deleted, and updated, and number of control-interval splits, can help you decide when to take action, such as reorganizing a data set or altering the type of processing, to improve performance.

You can list the entire catalog record, the statistics and the parameters selected when the data set was defined, by using the LISTCAT command. You can use the SHOWCB and TESTCB macros in a processing program to display or test one or more data set statistics. These statistics include:

- Control interval size
- Percent of free control intervals per control area
- Number of bytes of available space (includes distributed free control intervals and allocated space beyond the last used control interval)
- Length and displacement of the key
- Maximum record length
- Number of levels in the index

- Number of extents
- Number of records retrieved, added, deleted, and updated
- Number of control interval splits in the data and in the sequence set of the index
- Number of EXCPs that VSAM has issued for access to a data set

Note: When a cluster or component is exported, that is, is named in an EXPORT command, the statistics are exported with the catalog record. When the cluster is imported (with the IMPORT command), it is reorganized. Its old statistics aren't lost—they just don't apply to the reorganized data. When the cluster is loaded (as a result of IMPORT), its statistics are revised to reflect the newly-loaded cluster.

JOB CONTROL LANGUAGE

This chapter describes the job control language, required in VS1 and optional in MVS, used to connect a data set and the program that is to use it, how to code the VSAM AMP parameter, and how to identify user catalogs for jobs or job steps.

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, and thus connect program and data. JCL is used in MVS to catalog, uncatalog, and delete nonVSAM data sets in a VSAM catalog. Also in MVS, 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. See *OS/VS2 JCL* and *OS/VS2 System Programming Library: Job Management* for an explanation of dynamic allocation.

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 are sufficient to 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. You can specify DISP=SHR in the DD statements of separate jobs to enable two or more job steps to share a data set, provided the data set's share options allow the type of sharing your program anticipates. For more details on sharing data sets, see the chapter, "Sharing a VSAM Data Set."

How to Code JCL

All VSAM data sets are cataloged in a VSAM catalog. To identify a VSAM data set through JCL, it is sufficient to specify a DD statement of the form:

//*ddname* **DD DSNAME=***dsname***,DISP={OLD | SHR}**

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 are to 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.

Because the operating system does not disallow OS/VS DD parameters and subparameters that don't apply to a VSAM data set, you should be aware of the DD parameters and subparameters that have clear and unambiguous meaning when used with VSAM. Figure 2 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 2 should be avoided. For an explanation of potential problems you may encounter with those parameters and subparameters, see the appropriate *Access Method Services* publication.

To 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. 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 as a VSAM data set. If you specify VOLUME and UNIT to open a VSAM ACB, AMORG is not required.

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. VSAM does not allow you to concatenate STEPCAT and JOBCAT DD statements.

Coding a DD Statement for a User Catalog

The master catalog is always available, without specifying it via JCL. You make user catalogs available by describing them in DD statements with special names for a job (JOBCAT) or a job step (STEPCAT). You describe a catalog sufficiently by giving its data set name and specifying DISP=SHR. A user 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 for which STEPCAT is not specified. VSAM uses a data set's name as a search argument to search a catalog. In OS/VS2 MVS, 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 OS/VS2 MVS 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 user-catalog entry or a user catalog's alias entry in the master catalog. If the system finds a user catalog or alias entry whose name is the same as the data set name's first qualifier, the system searches that user catalog for the data set's catalog record, using the data set's full entryname.

Parameter	Subparameter	Comment
DDNAME	*ddname*	Works as in OS/VS.
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 the appropriate Access Method Services publication for a full description of data-set sharing.
	OLD	Works as in OS/VS; if specified for a VSAM catalog, however, defaults to SHR.
	PASS	For VSAM, KEEP is assumed for PASS.
DSNAME	*dsname*	Works as in OS/VS.
DUMMY		Works as in OS/VS, except that 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" later in this chapter).
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	Must be a group supported by VSAM. If not, OPEN will fail.
	p	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.
	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 shareable and have mounted on them volumes containing parts of the data set to be processed, they will also be allocated to this data set.
	DEFER	Works as in OS/VS.

Figure 2 (Part 1 of 2). JCL DD Parameters

The master catalog is assumed to contain the definition of the data set described in a DD statement if no user catalog is indicated or if the definition is not found in the user catalog(s) that are indicated. A user catalog is specified either for all of the steps of a job or for a particular step. To specify a job user 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 ...
```

VOLUME	**PRIVATE**	Works as in OS/VS.
	RETAIN	Works as in OS/VS.
	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. If VOLUME=SER and UNIT=type are specified, only those volumes specifically named are initially mounted. Other volumes may be mounted when they're needed if at least one of the units allocated to the data set is not shareable 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 made unshareable when unit count is less than the number of volume serial numbers specified or when DEFER is 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.

Figure 2 (Part 2 of 2). JCL DD Parameters

To specify a job-step user catalog, place a DD statement with the ddname STEPCAT after the EXEC statement of the step:

```
//         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. In VS1, if a catalog is specified in the CATALOG parameter and it is not also a master, STEPCAT, or JOBCAT catalog, the *dname* parameter must also be specified in CATALOG.
- Any user 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 VS1, if the entry is not found, the master catalog.
- For VS1, if the entry is not found, the system catalog.
- For VS2, if the entry is not found and the entry's name is a qualified name and the first qualifier (that is, the first one to eight characters before any period) is the same as the name or alias of a user catalog or the alias of a control volume, that user catalog or control volume is searched; otherwise, the master catalog.

Restriction: Control volumes are not searched when (1) an existing data set is to be deleted except when the data set to be deleted is a nonVSAM data set, or (2) when an existing data set is to be altered.

Coding the AMP Parameter

VSAM uses one additional JCL parameter: AMP. 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 have a subset of the volumes on which the data set is stored mounted or want to cause mounting to be deferred. If volume and unit information is coded on the DD card, a STEPCAT or JOBCAT DD card is also required if the data set is cataloged in a usercatalog.

BUFND=*number*
BUFNI=*number*
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.

CROPS={RCK | NCK | NRE | NRC}
specifies one of four checkpoint/restart options, which are described in detail in *OS/VS 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 1s in the first byte) with an ISAM processing program using the ISAM interface. I and L are described in the chapter "Using ISAM Programming with VSAM."

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 the chapter "Using ISAM Programming with VSAM."

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 and closing data sets and end-of-volume processing. You can print the trace output with the IMDPRDMP service program.

Trace also causes the VSAM Record Management Trace Faclity 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 the appropriate (VS1 or VS2) *VSAM Logic* manual.

Note: See "AMP Parameter Specification" in the chapter "Using ISAM Programming with VSAM" for additional information on the use of the AMP parameter with an ISAM processing program.

If you have more than one subparameters 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 of 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 therefore use a pair of parentheses to enclose all of 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).

Defining a VSAM Data Set

When you define a data set, no DD statement is required if Access Method Services can allocate space for the data space from an existing data space. If a data space must be created to allocate spae for the data set that you're defining, you need a DD statement (in OS/VS1) for OS/VS job management to provide device allocation: you specify storage unit, volume and DISP=OLD. You never specify space parameters (SPACE, SPLIT, or SUBALLOC, DELETE, CATLG, UNCATLG, or DISP=NEW), since you use Access Method Services to define and delete all VSAM objects.

MACRO INSTRUCTION DESCRIPTIONS AND RETURN CODES

This chapter identifies and briefly describes the macro instructions that are used to open and close a data set, manage VSAM control blocks, and issue data processing requests. The return codes you may get from these macros are also described here. Format descriptions and examples of each macro are in the following chapter.

Opening a Data Set

Before your processing program can gain access to a data set, it must issue the OPEN macro to open the data set for processing. OPEN causes VSAM to construct the control blocks it needs to process your requests, mount the volume(s) on which the data set is stored, verify that the data set matches the one you have identified via ACB or GENCB, and check the password that your program specified against the password (if any) in the catalog definition of the data set.

Return Codes from OPEN

When your program receives control after it has issued an OPEN macro, register 15 indicates whether all of the VSAM data sets were opened successfully:

Reg. 15	Condition
0	All data sets were opened successfully.
4	All data sets were opened successfully, but one or more warning messages were issued (codes less than X'80').
8	At least one data set (VSAM or nonVSAM) was not opened successfully; the access-method control block was restored to the contents it had before OPEN was issued, or, if the data set was already open, the access-method control block remains open and usable, and is not changed.
12	A nonVSAM data set was not opened successfully when a nonVSAM and a VSAM data set were being opened at the same time; the nonVSAM data control block was not restored to the contents it had before OPEN was issued (and the data set cannot be opened without the control block's being restored).

If register 15 contains 4, 8, or 12, you can find out whether a VSAM data set had a warning message, or wasn't opened successfully and why, by issuing SHOWCB to display the ERROR field in each access-method control block specified in OPEN. (See "SHOWCB Macro (Display an Access-Method Control Block)" in the chapter "Macro Instruction Descriptions and Return Codes.") Figure 3 shows the possible return codes that you may get from OPEN in the ERROR field in the access-method control block. In addition to these return codes, VSAM writes a message to the operator console and the programmer's listing to further explain the error. See *OS/VS Message Library: VS1 System Messages* and *OS/VS Message Library: VS2 System Messages* for a listing of VSAM messages for VS1 and VS2, respectively.

Code	Condition
0(0)	When register 15 = 0, no error. When register 15 = 8, one of the following conditions exists: • VSAM is processing the access-method control block for some other request. • The access-method control block is already open. • DDNAME was not specified correctly in the access-method control block. • The access-method control block address is invalid.
4(4)	The data set indicated by the access-method control block is already open.
96(60)	Warning message: an unusable data set was opened for input.
100(64)	Warning message: OPEN encountered an empty alternate index that is part of an upgrade set.
104(68)	Warning message: the time stamp of the volume on which a data set is stored doesn't match the system time stamp in the data set's catalog record; this indicates that extent information in the catalog record may not agree with the extents indicated in the volume's VTOC.
108(6C)	Warning message: the time stamps of a data component and an index component do not match; this indicates that either the data or the index has been updated separately from the other.
116(74)	Warning message: the data set was not properly closed. A previous VSAM program may have abnormally terminated. Data may be lost if processing continues; the Access Method Services VERIFY command may be used to cause the data set to be properly closed. See the appropriate Access Method Services publication for a description of the VERIFY command. In a cross-system shared DASD environment, a return code of 116 can have two meanings: (1) Data set was not properly closed, or (2) the data set is opened for output on another CPU.
128(80)	DD statement for this statement is missing.
132(84)	An uncorrectable I/O error occurred while VSAM was reading the job file control block (JFCB).
136(88)	Not enough virtual-storage space is available in your program's address space for work areas, control blocks, or buffers.
144(90)	An uncorrectable I/O error occurred while VSAM was reading or writing a catalog record.
148(94)	No record for the data set to be opened was found in the available catalog(s), or an unidentified error occurred while VSAM was searching the catalog.
152(98)	Security verification failed; the password specified in the access-method control block for a specified level of access doesn't match the password in the catalog for that level of access.
160(A0)	The operands specified in the ACB or GENCB macro are inconsistent with each other or with the information in the catalog record.
164(A4)	An uncorrectable I/O error occurred while VSAM was reading the volume label.
168(A8)	The data set is not available for the type of processing you specify, or an attempt was made to open a reusable data set with the reset option while another user had the data set open.
176(B0)	An error occurred while VSAM was attempting to fix a page of virtual storage in real storage.

Figure 3 (Part 1 of 2). OPEN Return Codes in the ERROR Field of the Access-Method Control Block

Code	Condition
180(B4)	A VSAM catalog specified in JCL either does not exist or is not open, and no record for the data set to be opened was found in any other catalog.
184(B8)	An uncorrectable I/O error occurred while VSAM was completing an I/O request.
188(BC)	The data set indicated by the access-method control block is not of the type that may be specified by an access-method control block.
192(C0)	An unusable data set was opened for output.
196(C4)	Access to data was requested via an empty path.
200(C8)	The format-4 DSCB indicates that the volume is unusable. There was an error in CONVERTV to convert the volume from either real to virtual or virtual to real.
204(CC)	The ACB MACRF specification is GSR and caller is not operating in supervisor protect key 0 to 7, or ACB MACRF specification is CBIC (Control Blocks in Common)[1] and caller is not operating in supervisor state with protect key 0 to 7.
208(D0)	The ACB MACRF specification is GSR and caller is using a VS1 system[1].
212(D4)	The ACB MACRF specification is GSR or LSR and the data set requires create processing[1].
216(D8)	The ACB MACRF specification is GSR or LSR and the key length of the data set exceeds the maximum key length specified in BLDVRP[1].
220(DC)	The ACB MACRF specification is GSR or LSR and the data set's control interval size exceeds the size of the largest buffer specified in BLDVRP[1].
224(E0)	Improved control interval processing is specified and the data set requires create mode processing[1].
228(E4)	The ACB MACRF specification is GSR or LSR and the VSAM Shared Resource Table (VSRT) does not exist (no buffer pool is available).
232(E8)	Reset was specified for a nonreusable data set and the data set is not empty.
236(EC)	A permanent staging error occurred in MSS (ACQUIRE).
240(F0)	Format-4 DSCB and volume timestamp verification failed during volume mount processing for output processing.
244(F4)	The volume containing the catalog recovery area was not mounted and verified for output processing.

[1] Options and restrictions are described in *OS/VS Virtual Storage Access Method (VSAM) Options for Advanced Applications.*

Figure 3 (Part 2 of 2). OPEN Return Codes in the ERROR Field of the Access-Method Control Block

Closing a Data Set

The CLOSE macro disconnects your program from a data set and causes VSAM to put back into the catalog the updated information that was brought into virtual storage when the data set was opened; write records in the SMF data set, if you are using SMF; and write out buffers of data or index whose contents have changed and which haven't already been written out.

If your program enters an abnormal termination routine, all data sets remaining open are closed. The VSAM CLOSE invoked by ABEND does not update the data set's catalog information, it does not complete outstanding I/O requests, and buffers are not flushed. This means that the catalog might not properly reflect the cluster's status, and the index might not accurately reference some of the data records. If your VSAM data set is closed as a result of ABEND, you should issue an Access Method Services VERIFY command to restore the data set's end-of-file values.

Return Codes from CLOSE

When your program receives control after it has issued a CLOSE macro, register 15 indicates whether all of the VSAM data sets were closed successfully:

Reg. 15	Condition
0	All data sets were closed successfully.
4	At least one data set (VSAM or nonVSAM) was not closed successfully.

If register 15 contains 4, you can use SHOWCB to display the ERROR field in each access-method control block to find out whether a VSAM data set wasn't closed successfully and why. (See "SHOWCB Macro (Display an Access-Method Control Block)" in the chapter "Macro Instruction Descriptions and Return Codes.") Figure 4 gives the return codes that the ERROR field may contain following CLOSE. In addition to these return codes, VSAM writes a message to the operator's console and the programmer's listing to further explain the error. See *OS/VS Message Library: VS1 System Messages* and *OS/VS Message Library: VS2 System Messages,* for a listing of messages for VS1 and VS2, respectively.

Code	Condition
0(0)	No error (set when register 15 contains 0).
4(4)	The data set indicated by the access-method control block is already closed.
132(84)	An uncorrectable I/O error occurred while VSAM was reading the job file control block (JFCB).
136(88)	Not enough virtual storage was available in your program's address space for a work area for CLOSE.
144(90)	An uncorrectable I/O error occurred while VSAM was reading or writing a catalog record.
148(94)	An unidentified error occurred while VSAM was searching the catalog.
184(B8)	An uncorrectable I/O error occurred while VSAM was completing outstanding I/O requests.
236(EC)	A permanent destaging error occurred in MSS (RELINQUISH). With temporary CLOSE, a destaging error or a staging error (ACQUIRE) occurred.

Figure 4. CLOSE Return Codes

OPEN/CLOSE/TCLOSE Message Area

During the execution of an OPEN, CLOSE, or TCLOSE macro, more than one error condition may be detected. However, the ACB error flag field can only accommodate one warning or error condition. In order to receive multiple error or warning conditions, you may specify an optional message area. VSAM will accumulate error messages from OPEN, CLOSE, or TCLOSE in this message area.

Multiple messages will be supplied when you specify nonzero values in the MAREA and MLEN parameters of the ACB. If MAREA or MLEN is not specified or is zero, no error or warning information is stored into the message area. The ACB error flag field is then the only indication for errors or warnings. If MAREA and MLEN are specified and if the message area is too small to accommodate all messages, the last incoming messages are dropped. However, you will be given an indication of the number of warnings and messages which occurred.

The message information provided by VSAM is subdivided into two parts:

- The message area header, and
- The message list

The message area header contains statistical, pointer, and general information. Its contents are unrelated to the individual messages. The format of the message area header is as follows:

Byte 0	Flag Byte	
	bit 0=1	Full message area header has been stored.
	byte 0=0	Only flag byte of message area header has been stored. (Implies that no messages have been stored.)
	bits 1-7	Reserved (set to binary zeros)
Bytes 1-2	Length of message area header (includes flag-and length-byte)	
Byte 3	Request type code	
	X '01'	OPEN
	X'02'	CLOSE
	X'03'	TCLOSE
Bytes 4-11	ddname used for ACB	
Bytes 12-13	Total number of messages (error or warning conditions) issued by OPEN/CLOSE/TCLOSE	
Bytes 14-15	Number of messages stored by OPEN/CLOSE/TCLOSE into message area	
Bytes 16-19	Address of message list, i.e., of first message in message area	

The function of the ACB error flag field remains unchanged regardless of whether or not this optional message area is specified. It contains, at the end of an OPEN, CLOSE, or TCLOSE, either X'00' indicating no error or warning condition occurred or a nonzero code. The nonzero code stored into the ACB error flag byte is the OPEN/CLOSE/TCLOSE error code

corresponding to the error or warning condition that occurred with the highest severity.

Message area header information is only stored when a warning or error condition is detected, that is, the ACB error flag field is set to nonzero. Furthermore, the header information will consist of the flag byte only, if the length of the message area (MLEN) is not large enough to accommodate the full message area header. In this case, bit 0 of the flag byte will be zero. Before accessing the message header information (bytes 1-19), you must test byte 0 to see whether further information is stored or not. If MLEN=0, no header information is stored at all, not even the flag byte. If the full message area header is stored, bytes 1-2 contain the actual length of the message area header, and your program should be sensitive to this length when interrogating the message area header.

The message list contains the individual messages corresponding to the warning or error conditions detected. The message list does not start at a fixed location in the message area. It is pointed to by bytes 16-19 of the message area header. This implies that the message list is not provided if the message area header is not stored completely (bit 0 of byte 0 being 0). Within the message list, individual messages are stored continuously one after another in the form of variable-length records. The number of messages stored is contained in the message area header (bytes 14-15). Before investigating the message list, you must check whether or not the stored-message count is zero. A nonzero content of the ACB error flag byte and the setting of bit 0 of byte 0 of the message area header do not guarantee that messages are stored: for example, if MLEN is not large enough to allow at least one message to be stored, no message will be stored.

The format of the individual messages is as follows:

Bytes 0-1 Length of message including these two

Byte 2 ACB error flag code corresponding to the warning condition represented by this message.

Byte 3 Function type code:

Specifies whether and which dsname is stored in bytes 4-47 of the message.

X'00' - No dsname stored. Bytes 4-47 of the message contain binary zeros. The error warning condition is not clearly related to a component, or VSAM was unable to identify or obtain the cluster name of the component in error. This code is used only if, in addition, the ddname of the ACB does not identify a valid DD statement or VSAM was unable to obtain the dsname contained in the DD statement.

X'01' - dsname contained in DD statement is stored. The error or warning condition is not clearly related to a component, or VSAM was unable to identify or obtain the cluster name of the component in error.

X'02' - dsname (cluster name) of base cluster stored. Error occurred during OPEN/CLOSE/TCLOSE for base cluster.

X'03' - dsname (cluster name) of AIX component stored. Error occurred during OPEN/CLOSE/TCLOSE for AIX component.

X'04'-	dsname (cluster name) of member of upgrade set stored. Error occurred during OPEN/CLOSE/TCLOSE for this member of the upgrade set.
Bytes 4-47	Binary zeros (function type code=X'00') or a dsname as described by byte 3.

Bytes 0 and 1 of each message specify the actual length of the individual message and you must inspect the length. Your processing should take the variable-length nature of the message into account.

You cannot conclude from the ACB error flag code in byte 2 of a message whether there is a dsname. Neither can you tell what type of dsname is contained in bytes 4-47 of the message. For one and the same ACB error flag code, none or different types of dsnames (DD, base cluster, AIX, or upgrade set member) may be stored depending on the individual condition which raised the appropriate ACB error flag code (the same condition may be detected when opening the base cluster when opening a member of the upgrade set; for example, an I/O error may occur when trying to obtain the dsname for the component in error). You must inspect byte 3 of the message to learn whether a dsname has been stored and to determine the type of dsname stored.

Control Block Macros

The control block macros are used to build control blocks and to modify, display, and test their contents. Some of these macros work at assembly time; others work at execution time. The macros are:

- ACB, which is used to generate an access-method control block at assembly time. An access-method control block must exist before a data set can be opened.
- EXLST, which is used to generate an exit list at assembly time. An exit list is a list of user-written routines that can be associated with one or more access-method control blocks. Except for an exception exit, which is specified via Access Method Services, a user-written routine must be identified in an exit list to be available to handle unusual conditions.
- RPL, which is used to generate a request parameter list at assembly time. A request parameter list is required for use with the request macros to define the characteristics of the request.
- GENCB, which is used to generate an access-method control block, an exit list, or a request parameter list at execution time. An access-method control block must exist before a data set can be opened. An exit list identifies any user-written routines provided. A request parameter list is required to define (specify processing) an action for a request macro.
- MODCB, which is used to modify or make an addition to an access-method control block, an exit list, or a request parameter list at execution time.
- SHOWCB, which is used to display fields in an access-method control block, an exit list, or a request parameter list at execution time.
- TESTCB, which is used to test the contents of fields in an access-method control block, an exit list, or a request parameter list at execution time.

Generating a control block at assembly time (ACB, EXLST, and RPL macros) has the advantage of generating the control block only once. When a control block is generated at assembly time, you are, however, exposed to the possibility of having to reassemble your program if you adopt a new version of VSAM in which the format of a control block has changed. Generating a control block at execution time (GENCB macro) has the advantage of not requiring reassembly of a program if the format of a control block changes in a subsequent version of VSAM. It has the disadvantage of having to execute the macro each time the program is executed.

Specifying Options at Assembly or Execution

For specifying processing options, you can use macros that generate control blocks when your program is assembled (ACB, EXLST, and RPL macros) or use a macro that generates control blocks when the program is executed (GENCB macro).

The macros that work at assembly time allow you to specify values for operands as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants. The macros that work at execution allow you to specify them in those ways and also in:

- Register notation, where the expression designating a register from 2 through 12 is enclosed in parentheses; for example, (2) and (REG), where REG is a label equated to a number from 2 through 12
- An expression of the form (S,scon), where scon is an expression valid for an S-type address constant, including the base-displacement form
- An expression of the form (*,scon), where scon is an expression valid for an S-type address constant, including the base-displacement form, and the address specified by scon is indirect—that is, it gives the location of the area that contains the value for the operand

For most programming applications, you can conveniently use register notation or absolute numeric expressions for numbers, character strings for names, and register notation or expressions that generate valid A-type address constants for addresses. "Appendix C: Operand Notation for GENCB, MODCB, SHOWCB, and TESTCB" gives all the ways of coding each operand for the macros that work at execution time.

You can write a reentrant program only with execution-time macros. "Appendix B: List, Execute, and Generate Forms of GENCB, MODCB, SHOWCB, and TESTCB" describes alternate ways of coding these macros for reentrant programs. The standard form of these macros is described in this chapter.

Return Codes from the GENCB, MODCB, SHOWCB, and TESTCB Macros

The GENCB, MODCB, SHOWCB, and TESTCB macros are executable (unlike the ACB, EXLST, and RPL macros): they cause control to be given to VSAM to perform the indicated task. VSAM indicates the task was completed by a return code in register 15:

Reg. 15	Condition
0	Task completed.
4	Task not completed.
8	An attempt was made to use the execute form of a macro (see "Appendix B: List, Execute, and Generate Forms of GENCB, MODCB, SHOWCB, and TESTCB") to modify a keyword that isn't in the parameter list.

When register 15 contains 4, register 0 contains an error code indicating the reason VSAM couldn't perform the task. Figure 5 describes each error code that can be returned in register 0.

These macros build a parameter list that describes in codes the actions indicated by the operands you specify. The parameter list is passed to VSAM to take the indicated actions. An error can occur because you specified the operands incorrectly or, if you constructed the parameter list yourself, because the parameter list was encoded incorrectly. If you construct the list yourself you can get in register 0 error codes 1, 2, 3, 10, 14, 20, and 21. See *OS/VS Virtual Storage Access Method (VSAM) Options for Advanced Applications* for an explanation of these error codes and for an explanation of how to construct parameter lists for GENCB, MODCB, SHOWCB, and TESTCB.

Error Code	Applicable Macros*	Reason VSAM Couldn't Perform the Task
0(1)	G,M,S,T	The request type (generate, modify, show, or test) is invalid.
2(2)	G,M,S,T	The block type (access-method control block, exit list, or request parameter list) is invalid.
3(3)	G,M,S,T	One of the keyword codes in the parameter list is invalid.
4(4)	M,S,T	The block at the address indicated is not of the type you indicated (access-method control block, exit list, or request parameter list).
5(5)	S,T	Access-method control block fields were to be shown or tested, but information available only when the data set is an open VSAM data set, and either it isn't open or it is not a VSAM data set.
6(6)	S,T	Access-method control block information about an index was to be shown or tested, but no index was opened with the data set.
7(7)	M,S	An exit list was to be modified, but the list was not large enough to contain the new entry; or an exit was to be modified or shown, but the specified exit wasn't in the exit list. (With TESTCB, if the specified exit address isn't present you get an unequal condition when you test for it.)
8(8)	G	There isn't enough virtual storage in your program's address space to generate the access-method control block(s), exit list(s), or request parameter list(s) and no work area outside your address space was specified.
9(9)	G,S	The work area specified was too small for generation or display of the indicated control block or fields.
10(A)	G,M	With GENCB, exit-list control-block type was specified and you specified an exit without giving an address. With MODCB, exit-list control-block type was specified and you specified an exit without giving an address; in this case, either active or inactive must be specified, but load cannot be specified.
11(B)	M	Either (1) a request parameter list was to be modified, but the request parameter list defines an asynchronous request that is active (that is, no CHECK or ENDREQ has been issued on the request) and thus cannot be modified, or (2) MODCB is already issued for the control block, but hasn't yet completed.
12(C)	M	An access-method control block was to be modified, but the data set identified by the access-method control block is open and thus cannot be modified.
13(D)	M	An exit list was to be modified, and you attempted to activate an exit without providing a new exit address. Because the exit list indicated does not contain an address for that exit, your request cannot be honored.
14(E)	G,M,T	One of the option codes (for MACRF, ATRB, or OPTCD) has an invalid combination of option codes specified (for example, OPTCD=(ADR,SKP)).
15(F)	G,S	The work area specified did not begin on a fullword boundary.
16(10)	G,M,S,T	A VTAM keyword or subparameter was specified but the AM=VTAM parameter was not specified. AM=VTAM must be specified in order to process a VTAM version of the control block.
19(13)	M,S,T	A keyword was specified which refers to a field beyond the length of the control block located at the address indicated (for example, a VTAM keyword, but the control block pointed to is a shorter, nonVTAM block.
20(14)	S	Keywords were specified which apply only if MACRF=LSR or GSR.
21(15)	S,T	The block to be displayed or tested does not exist because the data set is a dummy data set.

*G=GENCB, M=MODCB, S=SHOWCB, T=TESTCB

Figure 5. GENCB, MODCB, SHOWCB, and TESTCB Error Codes

Request Macros

This section describes the macro instructions that cause some action to be taken regarding data or processing. The request macros are:

- GET, which causes a record to be retrieved.
- PUT, which causes a record to be stored.
- ERASE, which causes a record previously retrieved for update to be deleted from a key-sequenced data set.
- POINT, which causes VSAM to position at the desired record.
- CHECK, which causes processing to be suspended to await the completion of some event.
- ENDREQ, which causes a specified request to be terminated.

GETIX and PUTIX, which are used to process the index of a key-sequenced data set, and VERIFY, which ensures that the catalog end-of-data-set information agrees with the actual end of data set, are described in *OS/VS Virtual Storage Access Method (VSAM) Options for Advanced Applications.*

Each request macro makes use of a request parameter list; the request parameter list defines the action to be taken. For example, when a GET macro points to a request parameter list that specifies synchronous, sequential retrieval, the next record in sequence is retrieved. When an ENDREQ macro points to a request parameter list, any current request (for example, a PUT) for that request parameter list is ended immediately.

A return code in register 15 and a code in the feedback field of the request parameter list indicate what happened as a result of a request macro. The return codes in register 15, feedback-field codes, and the request macros are described below.

Return Codes from Request Macros

After you issue a request macro for access to data or a CHECK or ENDREQ macro, register 15 contains a return code. The meaning of the return code depends on whether processing is asynchronous or synchronous.

After you issue an asynchronous request for access to a data set, VSAM indicates in register 15 whether the request was accepted, as follows:

Reg. 15	Condition
0	Request was accepted.
4	Request was not accepted because the request parameter list indicated by the request (RPL=address) was active for another request.

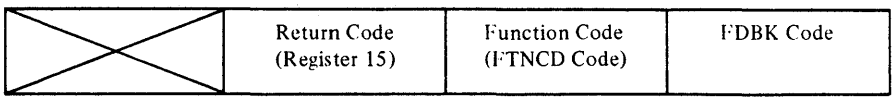
If the asynchronous request was accepted, you issue a CHECK after doing your other processing so VSAM can indicate in register 15 whether the request was completed successfully, set a return code in the feedback field, and exit to any appropriate exit routine. If the request was not accepted, you should either wait until the other request is complete (for example, by issuing a CHECK on the request parameter list) or terminate the other request (using ENDREQ). Then you can reissue the rejected request.

After a synchronous request, or a CHECK or ENDREQ macro, register 15 indicates whether the request was completed successfully, as follows:

Reg. 15	Condition
0	Request completed successfully.
4	Request was not accepted because the request parameter list indicated by the request (RPL=address) was active for another request.
8	Logical error; specific error is indicated in the feedback field in the RPL.
12	Physical error; specific error is indicated in the feedback field in the RPL.

Feedback-Field Codes

The feedback field in the RPL consists of three bytes. Each byte represents an error code as follows:



The Return Code is the code returned to you in register 15. Function and FDBK codes are described later in this chapter. Refer to the appropriate logic manual, *OS/VS1 Virtual Storage Access Method (VSAM) Logic* or *OS/VS2 Virtual Storage Access Method (VSAM) Logic* for more information on the feedback field of the RPL.

Paired with the 0, 8, and 12 indicators in register 15 are codes in the feedback field of the request parameter list. The feedback codes for the 0, or successful, indicator in register 15, which doesn't cause VSAM to exit to an exit routine, are:

FDBK Code When Register 15=0	Condition
0	Request completed successfully.
4	Request completed successfully. For retrieval, VSAM mounted another volume to locate the record; for storage, VSAM allocated additional space or mounted another volume.
8	For GET requests, indicates a duplicate alternate key exits (applies only when accessing a data set using an alternate index that allows nonunique keys); for PUT requests, indicates that a duplicate key was created in an alternate index with the nonunique attribute.
12	Write-buffer suggested (shared resources only).
16	The sequence-set record does not have enough space to allow it to address all of the control intervals in the control area that should contain the record. The record was written into a new control area.
20	Reserved
24	Reserved
28	Control interval split indicator was detected during an addressed GET NUP request.

You can examine the function and FDBK codes of the feedback field of the request parameter list with the SHOWCB or TESTCB macro. You may code your examination routine immediately following the request macro. However, logical errors, physical errors, and reaching the end of the data set all cause VSAM to exit to the appropriate exit routine, if you provide it.

Coordinate error checking in your program with your error-analysis exit routines. If they terminate the program, for instance, you would not need to code a check for an error after a request. But if a routine returns to VSAM to continue processing, you might check register 15 after a request to determine whether there was an error. Even though the error was handled by an exit routine, you may want to modify processing in light of the error.

Function Codes

When a logical or physical error occurs, VSAM provides a code in the RPL that identifies the function being attempted when the error occurred and indicates whether the alternate-index upgrade set is correct following the request that failed. The function code can be displayed and tested by the SHOWCB and TESTCB macros. The codes and their meanings are:

FTNCD Code	Function	Upgrade Set Status
0(0)	An attempt to access the base cluster	Correct
1(1)	An attempt to access the base cluster	May be incorrect
2(2)	An attempt to access the alternate index over a base cluster	Correct
3(3)	An attempt to access the alternate index over a base cluster	May be incorrect
4(4)	Upgrade processing	Correct
5(5)	Upgrade processing	May be incorrect

FDBK Code (Logical Errors)

If a logical error occurs and you have no LERAD routine (or the LERAD exit is inactive), VSAM returns control to your program following the last executed instruction. Register 15 indicates a logical error (8), and the feedback field in the request parameter list contains a code identifying the error. Register 1 points to the request parameter list. Figure 6 gives the logical-error return codes in the feedback field and explains what each one means.

FDBK Code When Register 15=8	Condition
4(4)	End of data set encountered (during sequential or skip-sequential retrieval), or the search argument is greater than the high key of the data set. Either no EODAD routine is provided, or one is provided and it returned to VSAM and the processing program issued another GET.
8(8)	You attempted to store a record with a duplicate key, or there is a duplicate record for an alternate index with the unique key option.
12(C)	A key sequence check was performed and an error was detected in one of the following processing conditions: • For a KSDS - PUT sequential or skip-sequential processing - GET sequential, single string input only - GET skip-sequential processing and the previous request is not a POINT For an RRDS - GET skip-sequential processing - PUT skip-sequential processing
16(10)	Record not found.
20(14)	Control Interval for record already held in exclusive control by another requester.
24(18)	Record resides on a volume that can't be mounted.
28(1C)	Data set cannot be extended because VSAM can't allocate additional direct-access storage space. Either there is not enough space left to make the secondary allocation request or you attempted to increase the size of a data set while processing with SHROPT=4 and DISP=SHR.
32(20)	You specified an RBA that doesn't give the address of any data record in the data set.
36(24)	Key ranges were specified for the data set when it was defined, but no range was specified that includes the record to be inserted.
40(28)	Insufficient virtual storage in your address space to complete the request.
44(2C)	Work area not large enough for the data record (GET with OPTCD=MVE).
64(40)	As many requests are active as the number specified in the STRNO parameter of the ACB macro; therefore, another request cannot be activated.
68(44)	You attempted to use a type of processing (output or control-interval processing) that was not specified when the data set was opened.
72(48)	You made a keyed request for access to an entry-sequenced data set, or you issued a GETIX or PUTIX to an entry-sequenced or relative record data set.
76(4C)	You issued an addressed or control-interval PUT to add to a key-sequenced data set, or you issued a control-interval PUT to a relative record data set.

Figure 6 (Part 1 of 3). Logical-Error Return Codes in the Feedback Field of the Request Parameter List

FDBK Code When Register 15-8	Condition
80(50)	You issued an ERASE request for access to an entry-sequenced data set, or you issued an ERASE request for access to an entry-sequenced data set via a path.
84(54)	You specified OPTCD=LOC for a PUT request or in a request parameter list in a chain of request parameter lists.
88(58)	You issued a sequential GET request without having caused VSAM to be positioned for it, or you changed from addressed access to keyed access without causing VSAM to be positioned for keyed-sequential retrieval; there was no positioning established for sequential PUT insert for a relative record data set, or you attempted an illegal switch between forward and backward processing.
92(5C)	You issued a PUT for update or an ERASE without a previous GET for update or a PUTIX without a previous GETIX.
96(60)	You attempted to change the prime key or key of reference while making an update.
100(64)	You attempted to change the length of a record while making an addressed update.
104(68)	The RPL options are either invalid or conflicting in one of the following ways: (1) SKP was specified and either KEY was not specified or BWD was specified (2) BWD was specified for CNV processing (3) FWD and LRD were specified (4) Neither ADR, CNV, nor KEY was specified in the RPL (5) WRTBFR, MRKBFR, or SCHBFR was issued, but either TRANSID was greater than 31 or the shared resource option was not specified (6) ICI processing was specified, but a request other than a GET or a PUT was issued (7) MRKBFR MARK=OUT or MARK=RLS was issued but the RPL did not have a data buffer associated with it. (8) The RPL specified WAITX, but the ACB did not specify LSR or GSR.
108(6C)	RECLEN specified was larger than the maximum allowed, equal to 0, or smaller than the sum of the length and the displacement of the key field; RECLEN was not equal to record (slot) size specified for a relative record data set.
112(70)	KEYLEN specified was too large or equal to 0.
116(74)	During initial data-set loading (that is, when records are being stored in the data set the first time it's opened), GET, POINT, ERASE, direct PUT, skip-sequential PUT, or PUT with OPTCD=UPD is not allowed. For initial loading of a relative record data set, the request was other than a PUT insert.
120(78)	The request was operating under an incorrect TCB. For example, an end-of-volume call or a GETMAIN would have been necessary to complete the request, but the request was issued from a job step other than the one that opened the data set. The request can be resubmitted from the correct task, if the new request reestablishes positioning.
132(84)	An attempt was made in locate mode to retrieve a spanned record.
136(88)	You attempted an addressed GET of a spanned record in a key-sequenced data set.

Figure 6 (Part 2 of3). Logical-Error Return Codes in the Feedback Field of the Request Parameter List

FDBK Code When Register 15-8	Condition
140(8C)	Inconsistent spanned record.
144(90)	Invalid pointer (no associated base record) in an alternate index.
148(94)	The maximum number of pointers in the alternate index has been exceeded.
152(98)	Not enough buffers are available to process your request (shared resources only).
156(9C)	An invalid control interval was detected during keyed processing, or an addressed GET UPD request failed because the control interval flag was on. The RPL contains the invalid control interval's RBA.
192(C0)	Invalid relative record number.
196(C4)	You issued an addressed request to a relative record data set.
200(C8)	You attempted addressed or control-interval access through a path.
204(CC)	PUT insert requests are not allowed in backward mode.
208(D0)	The user has issued an ENDREQ macro instruction against an RPL that has an outstanding WAIT against the ECB associated with the RPL. This can occur when an ENDREQ is issued from a STAE or ESTAE routine against an RPL that was started before the ABEND. No ENDREQ processing has been done.

Figure 6 (Part 3 of 3). Logical-Error Return Codes in the Feedback Field of the Request Parameter List

VSAM is not able to maintain positioning after every logical error. If positioning is maintained following a POINT or a direct request that encountered a logical error, then it may be in one of four states: (1) no position, if no positioning had been established at the time the request in error was issued (No), (2) the position in effect before the request in error was issued (Yes), (3) a new position (New), or (4) an unpredictable position (U). The table below shows what cases; apply to a given logical code for sequential, direct, and skip-sequential processing. If case (3) applies, "New" appears in the appropriate column; otherwise, "Yes," "No," or "U."

Whenever positioning is not maintained following an error request, you must reestablish it before processing resumes.

FDBK Code When Register 15=8	Sequential	Direct	Skip-Sequential
4(4)	Yes	N/A	Yes
8(8)[1]	Yes	No	New
12(C)	Yes	N/A	Yes
16(10)	No	No	No
20(14)	U	No[2]	No[2]
24(18)	Yes	No	No
28(1C)	Yes	No	Yes
32(20)	No	No	N/A
36(24)	Yes	No	New
40(28)	Yes	No	No
44(2C)	Yes	New	Yes
64(40)	No	No	No
68(44)	Yes	Yes	Yes
72(48)	Yes	Yes	Yes
76(4C)	Yes	Yes	Yes
80(50)	Yes	Yes	Yes
84(54)	Yes	Yes	Yes
88(58)	Yes	Yes	Yes
92(5C)	Yes	Yes	Yes
96(60)	Yes	Yes	Yes
100(64)	Yes	Yes	Yes
104(68)	Yes	New	Yes
108(6C)	Yes	New	Yes
112(70)	Yes	Yes	Yes
116(74)	Yes	Yes	Yes
120(78)	Yes	No	No
132(84)	Yes	New	Yes
136(88)	No	No	N/A
140(8C)	Yes	New	Yes
144(90)	Yes	Yes	Yes
148(94)	Yes	Yes	Yes
152(98)	Yes	No	No
156(9C)	Yes	No	No
192(C0)	Yes	Yes	Yes
196(C4)	Yes	Yes	Yes
200(C8)	Yes	Yes	Yes
204(CC)	Yes	Yes	Yes
208(D0)	Yes	Yes	Yes

1 A subsequent GET SEQ will retrieve the duplicate record; however, a subsequent GET SKP for the same key will get a sequence error. In a relative record data set, a subsequent PUT SEQ positions to the next slot (whether the slot is empty or not).

2 PUT UPD, DIR or UPD, SKP retains positioning. The RPL contains an RBA that could not be obtained for exclusive control.

When the search argument you supply for a POINT or GET request is greater than the highest key in the data set, the return code in the feedback field depends on the request RPL's OPTCD options, as illustrated in the table below:

Request Type	RPLs OPTCD Options	FDBK Code When Register 15=8
POINT	GEN,KEQ	16(10)
POINT	GEN,KGE	4(4)
POINT	FKEY,KEQ	16(10)
POINT	FKEY,KGE	4(4)
GET	GEN,KEQ,DIR	16(10)
GET	GEN,KGE,DIR	16(10)
GET	FKEY,KEQ,DIR	16(10)
GET	FKEY,KGE,DIR	16(10)
GET	GEN,KEQ,SKP	16(10)
GET	GEN,KGE,SKP	4(4)
GET	FKEY,KEQ,SKP	16(10)
GET	FKEY,KGE,SKP	4(4)

FDBK Code (Physical Errors)

If a physical error occurs and you have no SYNAD routine (or the SYNAD exit is inactive), VSAM returns control to your program following the last executed instruction. Register 15 indicates a physical error (12), and the feedback field in the request parameter list contains a code identifying the error; the message area contains more details about the error. Register 1 points to the request parameter list. Figure 7 gives the physical-error return codes in the feedback field and explains what each one indicates.

FDBK Code When Register 15=12 (0C)	Condition
4(4)	Read error occurred for a data set.
8(8)	Read error occurred for an index set.
12(C)	Read error occurred for a sequence set.
16(10)	Write error occurred for a data set.
20(14)	Write error occurred for an index set.
24(18)	Write error occurred for a sequence set.

Figure 7. Physical-Error Return Codes in the Feedback Field of the Request Parameter List

Figure 8 gives the format of a physical-error message. The format and some of the contents of the message are purposely similar to the format and contents of the SYNADAF message, which is described in *OS/VS Data Management Macro Instructions*.

Field	Bytes	Length	Discussion
Message Length	0-1	2	Binary value of 128
	2-3	2	Unused (0)
Message Length - 4	4-5	2	Binary value of 124 (provided for compatibility with SYNADAF message)
	6-7	2	Unused (0)
Address of I/O Buffer	8-11	4	The I/O buffer associated with the data in relation to which the error occurred
			The rest of the message is in printable format:
Date	12-16	5	YYDDD (year and day)
	17	1	Comma (,)
Time	18-25	8	HHMMSSTH (hour, minute, second, and tenths and hundredths of a second)
	26	1	Comma (,)
RBA	27-34	8	Relative byte address of the record in relation to which the error occurred.
	35	1	Comma (,)
Component TYPE	36-41	6	"DATA" or "INDEX"
	42	1	Comma (,)
Volume Serial Number	43-48	6	Volume serial number of the volume in relation to which the error occurred
	49	1	Comma (,)

Figure 8 (Part 1 of 3). Physical-Error Message Format

Field	Bytes	Length	Discussion
Job Name	50-57	8	Name of the job in which error occurred
	58	1	Comma (,)
Step Name	59-66	8	Name of the job step in which error occurred
	67	1	Comma (,)
Unit	68-70	3	The unit, CUU (channel and unit), in relation to which the error occurred
	71	1	Comma (,)
Device Type	72-73	2	The type of device in relation to which the error occurred (always DA for direct access)
	74	1	Comma (,)
ddname	75-82	8	The ddname of the DD statement defining the data set in relation to which the error occurred
	83	1	Comma (,)
Channel Command	84-89	6	The channel command that caused the error in the first two bytes, followed by "- OP"
	90	1	Comma (,)
Message	91-105	15	Messages are divided according to ECB condition codes: X'41' "INCORR LENGTH" "UNIT EXCEPTION" "PROGRAM CHECK" "PROTECTION CHK" "CHAN DATA CHK" "CHAN CTRL CHK" "INTFCE CTRL CHK" "CHAINING CHK" "UNIT CHECK" *If the type of unit check can be determined, the 'UNIT CHECK' message is replaced by one of the following:* " CMD REJECT" " INT REQ" "BUS OUT CK" "EQP CHECK" "DATA CHECK" "OVER RUN" "TRACK COND CK" "SEEK CHECK" "COUNT DATA CHK" "TRACK OVERRUN" "CYLINDER END" "INVALID SEQ" "NO RECORD FOUND" "FILE PROTECT" "MISSING A.M." "OVERFL INCP" X'48'—"PURGED REQUEST" X'4F'—"R.HA.RO. ERROR" For any other ECB completion code—"UNKNOWN COND."
	106	1	Comma (,)

Figure 8 (Part 2 of 3). Physical-Error Message Format

Field	Bytes	Length	Discussion
Physical Direct-Access Address	107-120	14	BBCCHHR (bin, cylinder, head, and record)
	121	1	Comma (,)
Access Method	122-127	6	"VSAM"

Figure 8 (Part 3 of 3). Physical-Error Message Format

MACRO INSTRUCTION FORMATS AND EXAMPLES

ACB Macro (Generate an Access-Method Control Block)

Before you can open a data set for processing, you must create an access-method control block that identifies the data set to be opened, specifies the type of processing (for example, sequential processing) to be done, specifies basic options (for example, buffer size), and indicates whether exit routines are to be used while the data set is being processed.

The ACB macro can be used to build an access-method control block when the program is assembled. If you adopt a subsequent release of VSAM in which the format of a control block has changed and have generated access-method control blocks using the ACB macro, you will have to reassemble your program.

Values for ACB-macro operands can be specified as absolute numeric expressions, character strings, codes, and expressions that generate valid relocatable A-type address constants.

VSAM allows multiple access-method control blocks to gain access to the same data set and conserves resources by connecting those ACBs to the same control block structure. The ACBs must be in the same region, and they must be opening to the same base cluster. The connection occurs independently of the path selected to the base cluster. For OS/VS2 MVS, if the ATTACH macro is used to create a new task that will be processing a shared data set, the ATTACH keyword SZERO should either be allowed to default to YES or SZERO=YES should be coded. This will cause subpool 0 to be shared with the subtask(s). For more information on the ATTACH macro, refer to *OS/VS2 Supervisor Service and Macros.*

Through MACRF options, you specify whether sharing is to be based on DDNAME (MACRF=DDN) or data set name (MACRF=DSN). If the DDN option is selected (or taken as the default), two ACBs that specify the same DDNAME will share the same control block structure. If the DSN option is selected, the new ACB will be connected to an existing control block structure if:

- The existing control block structure also specifies DSN and
- The new and existing control block structures have compatible processing options

To be compatible, both the new ACB and the existing control block structure must be consistent in their specification of the following processing options:

- Structure of both is either an entry-sequenced data set or a key-sequenced data set.
- Structure is a relative record data set
- Structure has MACRF=DFR
- Structure has MACRF=UBF
- Structure has MACRF=ICI
- Structure has MACRF=LSR
- Structure has MACRF=GSR

Those options that apply to the existing structure must also be specified in the new ACB. Conversely, the options that apply to the new ACB must also be specified in the existing structure. For example, if the new ACB and the existing structure both specify MACRF=DFR, the connection will be made. If the new ACB specifies MACRF=DFR and the existing structure specifies MACRF=DFR,UBF, no connection will be made.

If compatibility cannot be established, OPEN tries (within the limitations of the share options specified when the data set was defined) to build a new control block structure. If it can't, the OPEN fails.

The other information that you specify enables Open to prepare for the kind of processing to be done by your program:

- The address of a list of exit routine addresses that you supply. You use the EXLST macro to construct the list.
- For processing concurrent requests, the number of requests that are defined for processing the data set. The control blocks for the set of concurrent strings you specify are allocated on contiguous virtual storage. If the number you specify is not sufficient, OS/VS dynamically extends the number of strings as needed by concurrent requests for the ACB. Strings allocated by dynamic extension are not necessarily on contiguous storage.
- The size of the virtual storage space for I/O buffers and the number of I/O buffers that you are supplying for VSAM to process data and index records.
- The password that is required for the type of processing desired.
- The processing options used: keyed, addressed, or control interval, or a combination; sequential, direct, or skip sequential access, or a combination; retrieval, storage, or update (including deletion), or a combination; shared or nonshared resources.
- Address and length of an area for error messages from VSAM.

The format of the ACB macro is:

[*label*]	**ACB**	**[AM=VSAM]** **[,BSTRNO=** *number* **]** **[,BUFND=** *number* **]** **[,BUFNI=** *number* **]** **[,BUFSP=** *number* **]** **[,CATALOG=YES | NO]** **[,CRA=SCRA | UCRA]** **[,DDNAME=** *ddname* **]** **[,EXLST=** *address* **]** **[,MACRF=([ADR][,CNV][,KEY]** **[,CFX | NFX]** **[,DDN | DSN]** **[,DFR | NDF]** **[,DIR][,SEQ][,SKP]** **[,ICI | NCI]** **[,IN][,OUT]** **[,NIS | SIS]** **[,NRM | AIX]** **[,NRS | RST]** **[,NSR | LSR | GSR]** **[,NUB | UBF])]** **[,MAREA=** *address* **]** **[,MLEN=** *number* **]** **[,PASSWD=** *address* **]** **[,STRNO=** *number* **]**

where:

label
is one to eight characters that provides a symbolic address for the access-method control block that is assembled and also, if you omit the DDNAME operand, serves as the ddname.

AM=VSAM
specifies that the access method using this control block is VSAM.

BSTRNO=*number*
specifies the number of strings initially allocated for access to the base cluster of a path. The default is STRNO. BSTRNO is ignored if the object being opened is not a path. If the number specified for BSTRNO is insufficient, VSAM will dynamically extend the number of strings as needed for the access to the base cluster. BSTRNO can influence performance. The VSAM control blocks for the set of strings specified by BSTRNO are allocated on contiguous virtual storage, whereas this is not guaranteed for the strings allocated by dynamic extension.

BUFND=*number*
specifies the number of I/O buffers VSAM is to use for transmitting data between virtual and auxiliary storage. A buffer is the size of a control interval in the data component. 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). The number can be supplied by way of the JCL DD AMP parameter as well as by way of the

macro. The default is the minimum number required. Note, however, that minimum buffer specification does not provide optimum sequential processing performance. Generally, more data buffers specified, the better the performance. Note also that additional data buffers will benefit direct inserts or updates during control area splits and will benefit spanned record accessing. See the chapter "Optimizing VSAM's Performance" for more information.

BUFNI=*number*
specifies the number of I/O buffers VSAM is to use for transmitting the contents of index entries between virtual and auxiliary storage for keyed access. A buffer is the size of a control interval in the index. The minimum number is the number specified for STRNO (if you omit STRNO, BUFNI must be at least 1, because the default for STRNO is 1). You can supply the number by way of the JCL DD AMP parameter as well as by way of the macro. The default is the minimum number required.

Additional index buffers will improve performance by providing for the residency of some or all of the high-level index, thereby minimizing the number of high-level index records to be retrieved from DASD for key-direct processing. See the chapter "Optimizing VSAM's Performance" for more information.

BUFSP=*number*
specifies the maximum number of bytes of virtual storage to be used for the data and index I/O buffers. VSAM gets the storage in your program's address space. If you specify less than the amount of space that was specified in the BUFFERSPACE parameter of the DEFINE command when the data set was defined, VSAM overrides your BUFSP specification upward to the value specified in BUFFERSPACE. (BUFFERSPACE, by definition, is the least amount of virtual storage that will ever be provided for I/O buffers.) You can supply BUFSP by way of the JCL DD AMP parameter as well as by way of the macro. If you don't specify BUFSP in either place, the amount of storage used for buffer allocation is the *largest* of:

- the amount specified in the catalog (BUFFERSPACE),
- the amount determined from BUFND and BUFNI, or
- the minimum storage required to process the data set with its specified processing options

If BUFSP is specified and the amount is less than the minimum amount of storage required to process the data set, VSAM cannot open the data set.

A valid BUFSP amount takes precedence over the amount called for by BUFND and BUFNI. If the BUFSP amount is greater than the amount called for by BUFND and BUFNI, the extra space is allocated as follows:

- When MACRF indicates direct access only, additional index buffers are allocated.
- When MACRF indicates sequential access, one additional index buffer and as many data buffers as possible are allocated.

If the BUFSP amount is less than the amount called for by BUFND and BUFNI, the number of data and index buffers is decreased as follows:

- When MACRF indicates direct access only, the number of data buffers is decreased to not less than the minimum number. Then, if required, the number of index buffers is decreased until the amount called for by BUFND and BUFNI complies with the BUFSP amount.
- When MACRF indicates sequential access, the number of index buffers is decreased to not less than 1 more than the minimum number. Then, if required, the number of data buffers is decreased to not less than the minimum number. If still required, 1 more is subtracted from the number of index buffers.
- Neither the number of data buffers nor the number of index buffers is decreased to less than the minimum number.

If the index doesn't exist or isn't being opened, only BUFND, and not BUFNI, enters into these calculations.

CATALOG=YES | NO

specifies whether a catalog is being opened as a catalog (YES) or as a data set (NO). When NO is coded (or taken as the default), you can process the catalog with request macros (GET, PUT, etc). In OS/VS2 MVS systems, your program must be APF-authorized to process a catalog as a data set. To open a password-protected catalog for processing with VSAM macros, you must supply its master password. When CATALOG=YES is coded, the catalog must be processed with an SVC designed for that purpose. Access Method Services, for example, processes catalogs with SVC 26. The request macros are invalid for processing a catalog "as a catalog." IBM recommends that VSAM users alter the contents of a VSAM catalog only by way of Access Method Services commands.

CRA=SCRA | UCRA

specifies that a catalog recovery area is to be opened and that the control blocks are to be built in either system storage (SCRA) or user storage (UCRA). If you specify SCRA and issue record management requests, you must operate in key 0. If you specify UCRA, you must be authorized by the system and you must supply the master password of the master catalog.

DDNAME=*ddname*

is one to eight characters that identifies the data set that you want to process by specifying the JCL DD statement for the data set. You may omit DDNAME and provide it by way of the *label* or by way of the MODCB macro before opening the data set. MODCB is described later in this chapter.

EXLST=*address*

specifies the address of a list of addresses of exit routines that you are providing. The list is established by the EXLST or GENCB macro. If you use the EXLST macro, you can specify its label here as the address of the exit list. If you use GENCB, you can specify the address returned by GENCB in register 1 or the label of an area you supplied to GENCB for the exit list. Omitting this operand indicates that you have no exit routines. Exit routines are described in the chapter "User-Written Exit Routines."

MACRF=([ADR][,CNV][,KEY]
[,CFX | NFX]
[,DDN | DSN]
[,DFR | NDF]
[,DIR][,SEQ][,SKP]
[,ICI | NCI]
[,IN][,OUT]
[,NIS | SIS]
[,NRM | AIX]
[,NRS | RST]
[,NSR | LSR | GSR]
[,NUB | UBF])

specifies the kind(s) of processing you will do with the data set. The options must be meaningful for the data set. For example, if you specify keyed access for an entry-sequenced data set, you cannot open the data set. You must specify all of the types of access you're going to use, whether you use them concurrently or by switching from one to the other. Figure 9 gives the options; they are arranged in groups, and each group has a default value (indicated by underlining). You may specify options in any order. You may specify both ADR and KEY to process a key-sequenced data set. You may specify both DIR and SEQ; with keyed access, you may specify SKP as well. If you specify OUT and want simply to retrieve some records as well as update, delete, or insert others, you need not also specify IN.

MAREA=*address*
specifies the address of an optional OPEN/CLOSE/TCLOSE message area. See "OPEN/CLOSE/TCLOSE Message Area" for more information.

MLEN=*number*
specifies the length of an optional OPEN/CLOSE/TCLOSE message area. Default=0; maximum=32K. See "OPEN/CLOSE/TCLOSE Message Area" for more information.

PASSWD=*address*
specifies the address of a field that contains the highest-level password required for the type(s) of access indicated by the MACRF operand. The first byte of the field pointed to contains the length (in binary) of the password (maximum of 8 bytes). Zero indicates that no password is supplied. If the data set is password-protected and you don't supply a required password in the access-method control block, VSAM gives the console operator the opportunity to supply it when you open the data set.

STRNO=*number*
specifies the number of requests requiring concurrent data-set positioning VSAM is to be prepared to handle. The default is 1. A request is defined by a given request parameter list or chain of request parameter lists. See "RPL Macro (Generate a Request Parameter List)" and "GENCB Macro (Generate a Request Parameter List)" later in this chapter for information on request parameter lists. When records are loaded into an empty data set, the STRNO value in the access-method control block must be 1.

OS/VS dynamically extends the number of strings as needed by concurrent requests for this ACB, and this automatic extension can influence performance. The VSAM control blocks for the set of strings specified by STRNO are allocated on contiguous virtual storage, but this is not guaranteed

Option	Meaning
ADR	Addressed access to a key-sequenced or an entry-sequenced data set; RBAs are used as search arguments and sequential access is by entry sequence
CNV	Access is to the entire contents of a control interval rather than to an individual data record[1]
KEY	Keyed access to a key-sequenced or relative record data set; keys and relative record numbers are used as search arguments and sequential access is by key or relative record number
CFX	Control blocks and I/O buffers are to be fixed in real storage; MACRF=ICI must also be specified[1]
NFX	Control blocks and I/O buffers are fixed in real storage only during I/O operations[1]
DDN	Subtask shared control block connection is based on common ddnames
DSN	Subtask shared control block connection is based on common data set names
DFR	With shared resources, writes for direct PUT requests are deferred until the WRTBFR macro is issued or until VSAM needs a buffer to satisfy a GET request; deferring writes saves I/O requests in cases where subsequent requests can be satisfied by the data already in the buffer pool[1]
NDF	Writes are not to be deferred for direct PUTs
DIR	Direct access to a key-sequenced, entry-sequenced, or a relative record data set
SEQ	Sequential access to a key-sequenced, entry-sequenced, or a relative record data set
SKP	Skip-sequential access to a key-sequenced or a relative record data set; used only with keyed access in a forward direction.
ICI	Processing is limited to improved control-interval processing; access is faster because fewer CPU instructions are executed[1]
NCI	Processing other than improved control-interval processing
IN	Retrieval of records of a key-sequenced, entry-sequenced, or a relative record data set; (not allowed for an empty data set)
OUT	Storage of new records in a key-sequenced, entry-sequenced, or relative record data set (not allowed with addressed access to a key-sequenced data set); update of records in a key-sequenced, entry-sequenced, or relative record data set; deletion of records from a key-sequenced data set
NIS	Normal insert strategy
SIS	Sequential insert strategy (split control intervals and control areas at the insert point rather than at the midpoint when doing direct PUTs); although positioning is lost and writes are done after each direct PUT request, SIS allows more efficient space usage when direct inserts are clustered around certain keys
NRM	The object to be processed is the one named in the specified ddname
AIX	The object to be processed is the alternate index of the path specified by ddname, rather than the base cluster via the alternate index
NRS	Data set is not reusable
RST	Data set is reusable (high-used RBA is reset to 0 during OPEN processing)
NSR	Nonshared resources
LSR	Local shared resources; each partition or address space may have one resource pool independently of other partitions or address spaces[1]
GSR	Global shared resources (OS/VS2 MVS only); all address spaces in the system share one resource pool; OS/VS2 MVS systems may have local and global resources pools, where tasks in an address space with a local resource pool may use either the local resource pool or the global resource pool.
NUB	Management of I/O buffers is left up to VSAM
UBF	Management of I/O buffers is left up to the user; 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; valid when MACRF=MVE and CNV are specified; when ICI is specified, UBF is assumed

[1] Described in *OS/VS Virtual Storage Access Method (VSAM) Options for Advanced Applications*

Figure 9. MACRF Options

for the strings allocated by dynamic extension. Exceptions to dynamic string expansion are:

- Create mode
- ICI
- LSR or GSR

You could specify for STRNO the total number of request parameter lists or chains of request parameter lists that you are using to define requests. (VSAM needs to remember only one position for a chain of request

parameter lists.) However, each position beyond the minimum number that VSAM needs to be able to remember requires additional virtual storage space for:

- A minimum of one data I/O buffer and, for keyed access, one index I/O buffer (the size of an I/O buffer is the control-interval size of a data set)
- Internal control blocks and other areas

Example: ACB Macro

In this example, the ACB macro is used to identify a data set to be opened and to specify the types of processing to be performed. The access-method control block generated by this example is built when the program is assembled.

```
BLOCK    ACB    AM=VSAM,BUFND=4,      BLOCK gives symbolic address of the
                BUFNI=3,              access-method control block.
                BUFSP=19456,
                DDNAME=DATASETS,
                EXLST=EXITS,
                MACRF=(KEY,DIR,
                SEQ,OUT),
                PASSWD=FIELD,
                STRNO=2

FIELD    DC     FL1'6',C'CHANGE'      The update password: CHANGE has 6
                                      characters.
```

The ACB macro's operands are:

- BUFND, BUFNI, and BUFSP, which specify four I/O buffers for data; three I/O buffers for index entries; and 19,456 bytes of buffer space, enough space to accommodate control intervals of data that are 4096 bytes and control intervals of index entries that are 1024 bytes.
- DDNAME, which specifies that this access-method control block is associated with a DD statement named DATASETS.
- EXLST, which specifies that the exit list associated with this access-method control block is named EXITS.
- MACRF, which specifies keyed-direct and keyed-sequential processing for both insertion and update.
- PASSWD, which specifies the location, FIELD, of the password provided. FIELD contains the length of the password as well as the password itself.
- STRNO, which specifies that two requests will require concurrent positioning.

CHECK Macro (Suspend Processing)

The CHECK macro is used to suspend processing to await the completion of VSAM's processing of the request.

The format of the CHECK macro is:

[*label*]	**CHECK**	**RPL=** *address*

where:

label
is one to eight characters that provides a symbolic address for the CHECK macro.

RPL= *address*
specifies the address of the request parameter list that defines the request. You may specify the address in register notation (using a register from 1 through 12, enclosed in parentheses) or specify it with an expression that generates a valid relocatable A-type address constant.

Example: Check Return Codes after an Asynchronous Request

In this example, return codes are checked after an asynchronous request. The CHECK macro is used to cause an exit to be taken if there is a logical or physical error or if the end of the data set is reached.

```
REQPARMS RPL     OPTCD=ASY
          .
          .
          .
         GET     RPL=REQPARMS
         LTR     15,15          Was the request completed
                                successfully?
         BNZ     REJECTED       Zero indicates the request was
                                accepted. If it wasn't accepted, register
                                15 contains 4: REQPARMS is active
                                for another request.
                                Continue to work on something that is
                                not dependent on the request.
         CHECK   RPL=REQPARMS   CHECK would cause one of the three
                                exits to be taken if there was a logical
                                or physical error or if the end of the
                                data set was reached and an active exit
                                list exists.
         LTR     15,15          Test return indication in register 15.
         BNZ     FAILURE        Zero indicates the request completed
                                successfully. If it failed, register 15
                                contains 8 or 12: there was a logical or
                                a physical error.
          .
          .
          .
REJECTED ...
FAILURE  ...
```

Always test register 15 after the CHECK unless you provide exit routines that terminate processing. If a routine returns to VSAM, register 15 is reset and control is passed back to your program immediately after the CHECK. An error-analysis routine normally issues SHOWCB or TESTCB to examine the feedback field in the request parameter list, so that when your processing program gets control back, it doesn't have to analyze the error—but it may alter its processing if there was an error. If you don't provide an error-analysis routine, your program can issue SHOWCB or TESTCB to analyze an error when it gets control back following the CHECK.

Example: Check Return Codes after a Synchronous Request

With synchronous processing, you should test register 15 after the request because the request may not have been accepted (register 15 contains 4) or because an error might have occurred (8 or 12):

```
          GET     RPL=REQPARMS
          LTR     15,15          Was the request completed
                                 successfully?
          BNZ     REJFAIL        If branch isn't taken, request was
                                 accepted and completed successfully.
          .
          .
          .
REJFAIL   ...
```

Example: Overlap Processing

In this example, the CHECK macro is used to await completion of a request before continuing to other processing. Access is asynchronous.

```
BLOCK     ACB
LIST      RPL     ACB=BLOCK,             Asynchronous access.
                  AREA=WORK,
                  AREALEN=50,
                  OPTCD=ASY
          .
          .
          .
LOOP      GET     RPL=LIST
          LTR     15,15
          BNZ     NOTACCEP
Do other processing.
          CHECK   RPL=LIST               Suspends your processing to await
                                         completion of GET if necessary and to
                                         cause VSAM to indicate return codes.
          LTR     15,15
          BNZ     ERROR
Process the record.
          B       LOOP
NOTACCEP  ...                            Request wasn't accepted.
ERROR     ...                            Request failed.
          .
          .
          .
WORK      DS      CL50                   Work area.
```

After issuing the request, make sure that VSAM accepted it before you go on to do other processing. When you have done as much other processing as you can, issue the CHECK macro. VSAM will not give you back control now until the request is complete. If you don't want to issue CHECK until you know the request is complete, use the ECB operand of the RPL macro or the IO=COMPLETE operand of the TESTCB macro. After you issue the

CHECK, VSAM immediately returns a code and takes an exit, if necessary. See "RPL Macro (Generate a Request Parameter List)" and "GENCB Macro (Generate a Request Parameter List)" in this chapter for information on the ECB operand.

Example: Suspend a Request for Many Records

In this example, a CHECK macro is issued for the first request parameter list in a chain of parameter lists. If an error occurred for one of the request parameter lists in the chain and you have supplied error-analysis routines, VSAM takes a LERAD or SYNAD exit before it returns control to your program after the CHECK.

```
FIRST    RPL    ACB=BLOCK,
                AREA=AREA1,
                AREALEN=50,
                NXTRPL=SECOND,
                OPTCD=ASY
SECOND   RPL    ACB=BLOCK,
                AREA=AREA2,
                AREALEN=50,
                NXTRPL=THIRD,
                OPTCD=ASY
THIRD    RPL    ACB=BLOCK,          Last list doesn't indicate a next list.
                AREA=AREA3,
                AREALEN=50,
                OPTCD=ASY
         .
         .
         .
LOOP     GET    RPL=FIRST           Request gives the address of the first
                                    request parameter list.
         LTR    15,15
         BNZ    NOTACCEP
Do other processing.
         CHECK  RPL=FIRST
         LTR    15,15
         BNZ    ERROR
Process the three records retrieved by the GET.
         B      LOOP
NOTACCEP ...                        Request wasn't accepted.
ERROR    ...                        Display the feedback field
                                    (FIELDS=FDBK) of each request
                                    parameter list to find out which one
                                    had an error.
AREA1    DS     CL50                A single GET request causes VSAM to
                                    put a record in each of AREA1,
                                    AREA2, and AREA3.
AREA2    DS     CL50
AREA3    DS     CL50
```

After the CHECK, register 15 is set to indicate the status of the request. A code of 0 indicates that no error was associated with any of the request parameter lists. Any other code indicates that an error occurred for one of the request parameter lists. You should issue a SHOWCB macro for each request parameter list in the chain to find out which one had an error. VSAM doesn't process any of the request parameter lists beyond the one with an error.

CLOSE Macro (Disconnect Program and Data)

The Close routine completes any operations that are outstanding when a processing program issues a CLOSE macro for a data set. For instance, the Close routine causes VSAM to write out any data or index buffers whose contents have been changed and which haven't already been written out.

The Close routine updates the catalog with any changes in the attributes of a data set. The addition of records to a data set may cause its end-of-file indicator to change, in which case the Close routine updates the end-of-file indicator in the catalog. End-of-file indicators help ensure that the entire data set is accessible. If an error prevents VSAM from updating the indicators, the data set is flagged as improperly closed. The Close routine also causes VSAM to write records to the SMF data set if you are using SMF.

The Close routine restores control blocks to the status that they had before the data set was opened and frees the virtual-storage space that Open used to construct VSAM control blocks.

There is another way in which VSAM data sets are closed. Whenever you enter an abnormal termination routine, any data sets remaining open are closed. The VSAM CLOSE invoked by ABEND does not update the data set's catalog information, it does not complete outstanding I/O operations, and it does not flush buffers. This means that the index may not reflect the status of the data component and the catalog may not properly reflect the status of the cluster. If this happens, you should issue an Access Method Services VERIFY command to store the data set's end-of-file values.

When you issue a temporary or a permanent CLOSE macro, the VSAM Close routine rewrites the AMDSB information in the data set's catalog records. The AMDSB information includes statistics about device and volume, high-used RBA, free space, record size, etc. All the fields that are updated are described in the appropriate *VSAM Logic* publication.

The format of the CLOSE macro is:

[*label*]	**CLOSE**	(*address* [,(*options*)],...) [,**TYPE=T**]

where:

label
is one to eight characters that provides a symbolic address for the CLOSE macro.

address
specifies the address of the access-method control block or DCB for each data set to be closed. You may specify the address in register notation (using a register from 2 through 12—in parentheses) or specify it with an expression that generates a valid relocatable A-type address constant. If you specify only one address with a register, you must enclose the expression identifying the register in two sets of parentheses: for example, CLOSE ((2)).

options
are options parameters for use only in closing nonVSAM data sets. If any options are specified with the address of an access-method control block, VSAM ignores them. Because the CLOSE operands are positional, include a comma for options (even if you don't specify options) before a subsequent operand.

TYPE=T
specifies that VSAM is to complete outstanding I/O operations and update the catalog, but not disconnect the program from the data.

You can issue a temporary CLOSE macro to cause VSAM to complete outstanding I/O operations, put back into the catalog the updated information that was brought into virtual storage when the data set was opened, and write records in the SMF data set if you are using SMF. A temporary CLOSE doesn't disconnect the program from the data set, so your program can continue to process the data set without issuing an OPEN macro again.

You must close and reopen a newly created VSAM data set before you can issue non-create requests. A temporary close is not adequate for this purpose.

Note: If you are subtask-sharing or if you have issued an asynchronous request for access to a data set, you must issue a CHECK or an ENDREQ on all RPLs before you issue a CLOSE or CLOSE TYPE=T; otherwise, concurrent data set I/O activity will cause unpredictable results during a temporary close.

ENDREQ Macro (Terminate a Request)

The ENDREQ macro is used to terminate a request. Issuing an ENDREQ has the effect of releasing exclusive control as well as releasing positioning. In addition, buffers are written out if required.

The format of the ENDREQ macro is:

[*label*]	**ENDREQ**	**RPL=** *address*

where:

label
is one to eight characters that provides a symbolic address for the ENDREQ macro.

RPL= *address*
specifies the address of the request parameter list that defines the request. You may specify the address in register notation (using a register from 1 through 12, enclosed in parentheses) or specify it with an expression that generates a valid relocatable A-type address constant.

Note: The ENDREQ macro must not be issued when records are being loaded into a VSAM data set (create mode). ENDREQs issued while in create mode are ignored.

Example: Release Positioning for Another Request

In this example, the ENDREQ macro is used to cause VSAM to release exclusive control of a control interval containing a record. There are two request parameter lists, both of which require VSAM to have the ability to remember its position until VSAM is explicitly requested to forget its position.

```
BLOCK    ACB     MACRF=(SEQ,
                 DIR),STRNO=2
SEQ      RPL     ACB=BLOCK,          VSAM must remember its position.
                 OPTCD=SEQ
DIRUPD   RPL     ACB=BLOCK,          VSAM must remember its position and
                 OPTCD=(DIR,UPD)     maintain exclusive control until
                                     explicitly requested to forget it by PUT
                                     or ENDREQ.
         .
         .
         .
LOOP     GET     RPL=SEQ             VSAM now remembers its position for
                                     this request only while it is processing
                                     the request.
         LTR     15,15
         BNZ     ERROR
         GET     RPL=DIRUPD          VSAM can remember its position for
                                     this request. The C.I. will be placed in
                                     exclusive control until either ENDREQ
                                     or PUT UPD is issued.
         LTR     15,15
         BNZ     ERROR

Decide whether to update the record.
         B       FORGET              No; do not update the record.

         PUT     RPL=DIRUPD          Yes, update the record, causing VSAM
                                     to forget its position for DIRUP.

         LTR     15,15
         BNZ     ERROR
         B       LOOP
FORGET   ENDREQ  RPL=DIRUPD          Cause VSAM to forget its position for
                                     DIRUPD. Release exclusive control.
         LTR     15,15
         BNZ     ERROR
         B       LOOP
ERROR    ...                         Request wasn't accepted or failed.
```

The use of ENDREQ illustrated here causes VSAM to release exclusive control of the C.I. for a record. When PUT is issued after a DIRUPD GET request, ENDREQ need not be issued, since PUT causes VSAM to release exclusive control (the next DIRUPD GET doesn't depend on VSAM's remembering its position). Another result of ENDREQ is that buffers are written out if required.

To cause VSAM to give up its position associated with a chain of request parameter lists, specify the first request parameter list in the chain in your ENDREQ macro.

ENDREQ can also be used to cancel an asynchronous request—rather than suspending processing with CHECK. But simply ignoring a request whose completion you are not interested in is adequate.

Because VSAM remembers its position after a direct GET with OPTCD=UPD, if no PUT or ENDREQ follows, you can switch to sequential access and use the positioning for a GET.

ERASE Macro (Delete a Record)

The ERASE macro is used with the GET macro to delete records in a key-sequenced or relative record data set.

The format of the ERASE macro is:

[*label*]	**ERASE**	**RPL=** *address*

where:

label
is one to eight characters that provides a symbolic address for the ERASE macro.

RPL= *address*
specifies the address of a request parameter list that defines the request. You may specify the address in register notation (using a register from 1 through 12, enclosed in parentheses) or specify it with an expression that generates a valid relocatable A-type address constant.

Example: Keyed-Direct Deletion

In this example, GET and ERASE macros are used to retrieve and delete records. Not every record retrieved for deletion is deleted. The search argument is a full key (5 bytes), compared equal.

```
DELETE   ACB    MACRF=(KEY,DIR,
                OUT)
LIST     RPL    ACB=DELETE,
                AREA=WORK,
                AREALEN=50,
                ARG=KEYFIELD,
                OPTCD=(KEY,DIR,
SYN,UPD,                              UPD indicates deletion.
MVE,FKS,
KEQ)
         .
         .
         .
LOOP     MVC    KEYFIELD,source       Search argument for retrieval, from a
                                      table or transaction record.
         GET    RPL=LIST
         LTR    15,15
         BNZ    ERROR
Decide whether to delete the record.
         B      LOOP                  No, retrieve the next record.
         ERASE  RPL=LIST              Yes, delete the record.
         LTR    15,15
         BNZ    ERROR
         B      LOOP
ERROR    ...                          Request wasn't accepted or failed.
WORK     DS     CL50                  Examine the data record here.
KEYFIELD DS     CL5                   Search argument.
```

When you retrieve a record for deletion (OPTCD=UPD, same as retrieval for update), VSAM is positioned at the record retrieved, in anticipation of a succeeding ERASE (or PUT) request for that record. You are not required to issue such a request, though. Another GET request nullifies any previous positioning for deletion or update.

Keyed-sequential retrieval for deletion varies from direct in not using a search argument (except for possible use of the POINT macro). Skip-sequential retrieval for deletion (OPTCD=(SKP,UPD)) has the same effect as direct, but it is faster or slower depending on the number of control intervals separating the records being retrieved.

Example: Addressed-Sequential Deletion

In this example, the ERASE macro is used to delete records from a key-sequenced data set. Not every record retrieved for deletion is deleted. Skipping is effected by POINT macro.

```
DELETE   ACB     MACRF=(ADR,SEQ,
                 OUT)
REQUEST  RPL     ACB=DELETE,
                 AREA=WORK,
                 AREALEN=100,
                 ARG=ADDR,
                 OPTCD=(ADR,SEQ,
                 ASY,
UPD,MVE)                          UPD indicates deletion.
         .
         .
         .
LOOP     ...                      Decide whether you need to skip to
                                  another position (forward or
                                  backward).
         B       RETRIEVE         No, bypass the POINT.
         MVC     ADDR,source      Yes, move search argument for POINT
                                  into search-argument field.
         POINT   RPL=REQUEST      Position VSAM to the record to be
                                  retrieved next.
         LTR     15,15
         BNZ     ERROR
         CHECK   RPL=REQUEST
         LTR     15,15
         BNZ     ERROR
RETRIEVE GET     RPL=REQUEST
         LTR     15,15
         BNZ     ERROR
         CHECK   RPL=REQUEST
         LTR     15,15
         BNZ     ERROR
Decide whether to delete the record.
         B       LOOP             No, skip ERASE and CHECK.
         ERASE   RPL=REQUEST      Yes, delete the record.
         LTR     15,15
         BNZ     ERROR
         CHECK   RPL=REQUEST
         LTR     15,15
         BNZ     ERROR
         B       LOOP
ERROR    ...                      Request wasn't accepted or failed.
         .
         .
         .
ADDR     DS      F                RBA search argument for POINT.
WORK     DS      CL100            Work area.
```

Addressed deletion is allowed only for a key-sequenced data set. The records of an entry-sequenced data set are fixed. When records are deleted using addressed deletion from a key-sequenced data set, the index is not updated.

EXLST Macro (Generate an Exit List)

You use the EXLST macro to specify the addresses of optional exit routines that you can supply for analyzing physical and logical errors, processing an end-of-data-set condition, noting RBA changes, and writing in a journal. Any number of ACB macros in a program can indicate the same exit list for the same exit routines to do all the special processing for them, or they can indicate different exit lists. You can use exit routines for:

Analyzing physical errors: When VSAM encounters an error in an I/O operation that OS/VS's error routine cannot correct, the error routine formats a message for your physical-error analysis routine (the SYNAD exit) to act on.

Analying logical errors: Errors not directly associated with an I/O operation, such as an invalid request, cause VSAM to exit to your logical-error analysis routine (the LERAD exit).

End-of-data-set processing: When your program requests a record beyond the last record in the data set, your end-of-data-set routine (the EODAD exit) is given control. The end of the data set is beyond either the highest addressed or the highest keyed record, depending on whether your program is using addressed or keyed access.

Writing a journal: To journalize the transactions against a data set, you might specify a journal routine (the JRNAD exit). To process a key-sequenced data set by way of addressed access, you need to know whether any RBAs changed during keyed processing. When you're processing by key, VSAM exits to your routine for noting RBA changes before writing this control interval in which there is an RBA change.

See the chapter "User-Written Exit Routines" for a description of the exit routines. The EXLST macro is coordinated with the EXLST operand of an ACB or GENCB macro used to generate an ACB: you must code the EXLST operand to make use of the exit list.

Values for EXLST macro operands can be specified as absolute numeric expressions, character strings, codes, and expressions that generate valid relocatable A-type address constants.

The format of the EXLST macro is:

[*label*]	**EXLST**	**[AM=VSAM]** **[,EODAD=(***address* **[,A | N][,L])]** **[,JRNAD=(***address* **[,A | N][,L])]** **[,LERAD=(***address* **[,A | N][,L])]** **[,SYNAD=(***address* **[,A | N][,L])]** **[,UPAD=(***address* **[,A | N][,L])]**

where:

label
is one to eight characters that provides a symbolic address for the exit list that is established.

AM=VSAM
specifies that the access method using the control block is VSAM.

EODAD=(*address* **[,A | N][,L])**
JRNAD=(*address* **[,A | N][,L])**
LERAD=(*address* **[,A | N][,L])**
SYNAD=(*address* **[,A | N][,L])**
UPAD=(*address* **[,A | N][,A])**

specify that you are supplying a routine for the exit specified. The exits and values that can be specified for them are:

EODAD
specifies that an exit is provided for special processing when the end of a data set is reached by sequential access.

JRNAD
specifies that an exit is provided for journalizing as you process data records.

LERAD
specifies that an exit is provided for analyzing logical errors.

SYNAD
specifies that an exit is provided for analyzing physical errors.

UPAD
specifies that an exit is provided for user processing during a VSAM request. The GENCB, MODCB, SHOWCB, and TESTCB macros do not support the UPAD user exit routine.

address
is the address of a user-supplied exit routine. The *address* must immediately follow the equal sign.

UPAD
specifies that an exit is provided for user processing during a VSAM request. The GENCB, MODCB, SHOWCB, and TESTCB macros do not support the UPAD user exit routine.

A | N
specifies that the exit routine is active (A) or not active (N). VSAM does not enter a routine whose exit is marked not active.

L
specifies that the *address* is the address of an eight-byte field that contains the name of an exit routine in a partitioned data set that is identified by a JOBLIB or STEPLIB DD statement or in SYS1.LINKLIB. VSAM is to load the exit routine for exit processing. If **L** is omitted, the address gives the entry point of the exit routine in virtual storage. **L** may precede or follow the **A** or **N** specification.

Example: EXLST Macro

In this example, an EXLST macro is used to identify exit routines that are provided for analyzing logical and physical errors. The label, EXITS, of the EXLST macro is used in an ACB or GENCB macro that generates an access-method control block to associate the exit list with an access-method control block. The exit list generated by this example is built when the program is assembled.

```
EXITS    EXLST  EODAD=(ENDUP,N),     EXITS gives symbolic address of the
                LERAD=LOGICAL,       exit list.
                SYNAD=(ROUTNAME,
                L)

ENDUP                                EODAD routine.
LOGICAL                              LERAD routine.
ROUTNAME DC     C'PHYSICAL'          Pad shorter names with blanks:
                                     C'SYN    ' or CL8'SYN'.
```

The EXLST macro's operands are:

- EODAD, which specifies that the end-of-data routine is located at ENDUP and is not active.
- LERAD, which specifies that the logical-error routine is located at LOGICAL and is active.
- SYNAD, which specifies that the physical-error routine's name is located at ROUTNAME.

GENCB Macro (Generate an Access-Method Control Block)

Before you can open a data set for processing, you must create an access-method control block that identifies the data set to be opened, specifies the type of processing (for example, sequential processing) to be done, specifies basic options (for example, buffer size), and indicates whether exit routines are to be used while the data set is being processed.

The GENCB macro is used to build an access-method control block when the program is executed. Generation at execution has the advantage of requiring no reassembly of a program when you adopt a new version of VSAM in which control block formats might have changed.

The operands of the GENCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. "Appendix C: Operand Notation for GENCB, MODCB, SHOWCB, and TESTCB" gives all the ways of coding each operand for the macros that work at execution.

See "Return Codes from the GENCB, MODCB, SHOWCB, and TESTCB Macros" for information on the return codes used to indicate whether the GENCB request was successful.

The format of the GENCB macro used to generate an access-method control block is:

[*label*]	**GENCB**	**BLK=ACB** **[,AM=VSAM]** **[,BSTRNO=** *number* **]** **[,BUFND=** *number* **]** **[,BUFNI=** *number* **]** **[,BUFSP=** *number* **]** **[,CATALOG=YES | NO]** **[,COPIES=** *number* **]** **[,CRA=SCRA | UCRA]** **[,DDNAME=** *ddname* **]** **[,EXLST=** *address* **]** **[,LENGTH=** *number* **]** **[,MACRF=([ADR][,CNV][,KEY]** **[,CFX | NFX]** **[,DDN | DSN]** **[,DFR | NDF]** **[,DIR][,SEQ][,SKP]** **[,ICI | NCI]** **[,IN][,OUT]** **[,NIS | SIS]** **[,NRM | AIX]** **[,NRS | RST]** **[,NSR | LSR | GSR]** **[,NUB | UBF])]** **[,MAREA=** *address* **]** **[,MLEN=** *number* **]** **[,PASSWD=** *address* **]** **[,STRNO=** *number* **]** **[,WAREA=** *address* **]**

where:

label
is one to eight characters that provides a symbolic address for the GENCB macro.

BLK=ACB
specifies that you are generating an access-method control block.

AM=VSAM
specifies that the access method using this control block is VSAM.

BSTRNO=*number*
specifies the number of strings initially allocated for access to the base cluster of a path. The default is STRNO. BSTRNO is ignored if the object being opened is not a path. If the number specified for BSTRNO is insufficient, VSAM will dynamically extend the number of strings as needed for the access to the base cluster. BSTRNO can also influence performance. The VSAM control blocks for the set of strings specified by BSTRNO are allocated on contiguous virtual storage, whereas this is not guaranteed for the strings allocated by dynamic extension.

BUFND=*number*

specifies the number of I/O buffers VSAM is to use for transmitting data between virtual and auxiliary storage. A buffer is the size of a control interval in the data component. 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). The number can be supplied by way of the JCL DD AMP parameter as well as by way of the macro. The default is the minimum number required. A larger number for BUFND can improve the performance of sequential access.

BUFNI=*number*

specifies the number of I/O buffers VSAM is to use for transmitting index entries between virtual and auxiliary storage for keyed access. A buffer is the size of a control interval in the index. The minimum number is the number specified for STRNO (if you omit STRNO, BUFNI must be at least 1, because the default for STRNO is 1). You can supply the number by way of the JCL DD AMP parameter as well as by way of the macro. The default is the minimum number required. A larger number for BUFNI can improve the performance of keyed-direct retrieval.

BUFSP=*number*

specifies the maximum number of bytes of virtual storage to be used for the data and index I/O buffers. VSAM gets the storage in your program's address space. If you specify less than the amount of space that was specified in the BUFFERSPACE parameter of the DEFINE command when the data set was defined, VSAM overrides your BUFSP specification upward to the value specified in BUFFERSPACE. (BUFFERSPACE, by definition, is the least amount of virtual storage that will ever be provided for I/O buffers.) You can supply BUFSP by way of the JCL DD AMP parameter as well as by way of the macro. If you don't specify BUFSP in either place, the amount of storage used for buffer allocation is the *largest* of:

- the amount specified in the catalog (BUFFERSPACE),
- the amount determined from BUFND and BUFNI, or
- the minimum storage required to process the data set with its specified processing options

If BUFSP is specified and the amount is less than the minimum amount of storage required to process the data set, VSAM cannot open the data set.

A valid BUFSP amount takes precedence over the amount called for by BUFND and BUFNI. If the BUFSP amount is greater than the amount called for by BUFND and BUFNI, the extra space is allocated as follows:

- When MACRF indicates direct access only, additional index buffers are allocated.
- When MACRF indicates sequential access, one additional index buffer and as many data buffers as possible are allocated.

If the BUFSP amount is less than the amount called for by BUFND and BUFNI, the number of data and index buffers is decreased as follows:

- When MACRF indicates direct access only, the number of data buffers is decreased to not less than the minimum number. Then if required, the number of index buffers is decreased until the amount called for by BUFND and BUFNI complies with the BUFSP amount.
- When MACRF indicates sequential access, the number of index buffers is decreased to not less than 1 more than the minimum number. Then, if required, the number of data buffers is decreased to not less than the minimum number. If still required, 1 more is subtracted from the number of index buffers.
- Neither the number of data buffers nor the number of index buffers is decreased to less than the minimum number.

If the index doesn't exist or isn't being opened, only BUFND, and not BUFNI, enters into these calculations.

CATALOG=YES | NO
specifies whether a catalog is being opened as a catalog (YES) or as a data set (NO). When NO is coded (or taken as the default), you can process the catalog with request macros. (GET,PUT,etc.) To open a password-protected catalog for processing with VSAM macros, you must supply its master password. When CATALOG=YES is coded, the catalog must be processed with an SVC designed for that purpose. Access Method Services, for example, processes catalogs with SVC 26. The request macros are invalid for processing a catalog "as a catalog." IBM recommends that VSAM users alter the contents of a VSAM catalog only by way of Access Method Services commands.

COPIES=*number*
specifies the number of copies of the access-method control block VSAM is to generate. All of the copies are identical. You can use MODCB to tailor each one for the data set and processing you want for it. MODCB is described in this chapter.

CRA=SCRA | UCRA
specifies that a catalog recovery area is to be opened and that the control blocks are to be built in either system storage (SCRA) or user storage (UCRA). If you specify SCRA and issue record management requests, you must operate in key 0. If you specify UCRA, you must be authorized by the system and you must supply the master password of the master catalog.

DDNAME=*ddname*
is one to eight characters that identifies the data set that you want to process by specifying the JCL DD statement for the data set. You may omit DDNAME and provide it by way of the MODCB macro before opening the data set. MODCB is described later in this chapter.

EXLST=*address*
specifies the address of a list of addresses of exit routines that you are providing. The list is established by the EXLST or GENCB macro. If you use the EXLST macro, you can specify its label here as the address of the exit list. If you use GENCB, you can specify the address returned by GENCB in register 1. Omitting this operand indicates that you have no exit routines. Exit routines are described in the chapter "User-Written Exit Routines."

LENGTH=*number*
specifies the length, in bytes, of the area, if any, that you are supplying for VSAM to generate the access-method control block(s). (See the WAREA operand.)

MACRF=([ADR][,CNV][,KEY]
[,CFX | NFX]
[,DDN | DSN]
[,DFR | NDF]
[,DIR][,SEQ][,SKP]
[,ICI | NCI]
[,IN][,OUT]
[,NIS | SIS]
[,NRM | AIX]
[,NRS | RST]
[,NSR | LSR | GSR]
[,NUB | UBF])

specifies the kind(s) of processing you will do with the data set. The options must be meaningful for the data set. For example, if you specify keyed access for an entry-sequenced data set, you cannot open the data set. You must specify all of the types of access you're going to use, whether you use them concurrently or by switching from one to the other. The options are shown earlier in Figure 9; they are arranged in groups, and each group has a default value (indicated by underlining). You may specify options in any order. You may specify both ADR and KEY to process a key-sequenced data set. You may specify both DIR and SEQ; with keyed access, you may specify SKP as well. If you specify OUT and want simply to retrieve some records as well as update, delete, or insert others, you need not also specify IN.

MAREA=*address*
specifies the address of an optional OPEN/CLOSE/TCLOSE message area.

MLEN=*number*
specifies the length of an optional OPEN/CLOSE/TCLOSE message area.

PASSWD=*address*
specifies the address of a field that contains the highest-level password required for the type(s) of access indicated by the MACRF operand. The first byte of the field contains the length (in binary) of the password (maximum of 8 bytes). Zero indicates that no password is supplied. If the data set is password-protected and you don't supply a required password in the access-method control block, VSAM gives the console operator the opportunity to supply it when you open the data set.

STRNO=*number*
specifies the number of requests requiring concurrent data-set positioning VSAM is to be prepared to handle. A request is defined by a given request parameter list or chain of request parameter lists. See "RPL Macro (Generate a Request Parameter List)" and "GENCB Macro (Generate a Request Parameter List)" in this chapter for information on request parameter lists.

WAREA=*address*
specifies the address of an area in which the access-method control block(s) is to be generated. (Otherwise VSAM obtains virtual-storage space for the area and returns its address to you in register 1 and its length in register 0.) The area must begin on a fullword boundary. This operand is paired with the LENGTH operand, which must be given if you specify an area address.

If you did not specify an area in which the access-method control block was to be generated, VSAM returns to your program the address of the area containing the control block(s) in register 1 and the length of the area in register 0. You can find out the length of each control block by dividing the length of the area by the number of copies. The address of each control block can then be calculated by this offset from the address in register 1. You can find the length of an access-method control block with the SHOWCB macro. SHOWCB is described later in this chapter.

If you are generating control blocks by issuing several GENCBs, specifying an area (WAREA and LENGTH parameters) for them enables you to address all of them with one base register and to avoid repetitive requests for virtual storage.

Example: GENCB Macro (Generate an Access-Method Control Block)

In this example, a GENCB macro is used to identify a data set to be opened and to specify the types of processing to be performed. The access-method control block generated by this example is built when the program is executed.

```
GENCB    GENCB  BLK=ACB,AM=VSAM,          1 copy generated; VSAM gets the
                BUFND=4,BUFNI=3,          storage for it, because the WAREA and
                BUFSP=19456,              LENGTH operands have been omitted.
                DDNAME=DATASETS,
                EXLST=EXITS,
                MACRF=(KEY,DIR,
                SEQ,OUT),
                PASSWD=FIELD,
                STRNO=2
         ST     1,ACBADDR                 Save the address of the access-method
                                          control block.
ACBADDR  DS     F                         The address of the access-method
                                          control block is saved in ACBADDR.
FIELD    DC     FL1'6',C'CHANGE'          CHANGE, the password, has 6
                                          characters.
```

The GENCB macro's operands are:

- BUFND, BUFNI, and BUFSP, which specify four I/O buffers for data; three I/O buffers for index entries; and 19,456 bytes of buffer space, enough space to accommodate control intervals of data that are 4096 bytes and of index entries that are 1024 bytes.
- DDNAME, which specifies that this access-method control block is associated with a DD statement named DATASETS.
- EXLST, which specifies that the exit list associated with this access-method control block is named EXITS.
- MACRF, which specifies keyed direct and keyed sequential processing for both insertion and update.
- PASSWD, which specifies the location, FIELD, of the password provided.

- STRNO, which specifies that two requests will require concurrent positioning.

GENCB Macro (Generate an Exit List)

The GENCB macro can be used to generate an exit list when the program is executed. The GENCB macro is coordinated with the EXLST operand of the ACB or GENCB macro used to generate an access-method control block to make use of the exit list. One or more access-method control blocks may use the same exit list—the exit routines indicated by the list would do all the exit processing for the data sets identified by the control blocks.

The operands of the GENCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. "Appendix C: Operand Notation for GENCB, MODCB, SHOWCB, and TESTCB" gives all the ways of coding each operand for the macros that work at execution.

See "Return Codes from the GENCB, MODCB, SHOWCB, and TESTCB Macros" for information on return codes used to indicate whether the GENCB request was successful.

The format of the GENCB macro used to generate an exit list is:

[*label*]	**GENCB**	**BLK=EXLST** **[,AM=VSAM]** **[,EODAD=(***address* **[,A | N][,L])]** **[,JRNAD=(***address* **[,A | N][,L])]** **[,LERAD=(***address* **[,A | N][,L])]** **[,SYNAD=(***address* **[,A | N][,L])]** **[,COPIES=** *number*] **[,LENGTH=** *number*] **[,WAREA=** *address*]

where:

label
is one to eight characters that provides a symbolic address for the GENCB macro.

BLK=EXLST
specifies that you are generating an exit list.

AM=VSAM
specifies that the access method using this control block is VSAM.

[,EODAD=(*address* **[,A | N][,L])]**
[,JRNAD=(*address* **[,A | N][,L])]**
[,LERAD=(*address* **[,A | N][,L])]**
[,SYNAD=(*address* **[,A | N][,L])]**
specify that you are supplying a routine for the exit named. If none of these is specified, VSAM generates an exit list with inactive entries for all of the exits. The exits and values that can be specified for them are:

EODAD
specifies that an exit is provided for special processing when the end of a data set is reached by sequential access.

JRNAD
specifies that an exit is provided for journalizing as you process data records.

LERAD
specifies that an exit is provided for analyzing logical errors.

SYNAD
specifies that an exit is provided for analyzing physical errors.

address
is the address of a user-supplied exit routine. The *address* must immediately follow the equal sign.

A | N
specifies that the exit routine is active (A) or not active (N). VSAM does not enter a routine whose exit is marked not active.

L
specifies that the *address* is the address of an eight-byte field that contains the name of an exit routine in a partitioned data set that is identified by a JOBLIB or STEPLIB DD statement or in SYS1.LINKLIB. VSAM is to load the exit routine for exit processing. If **L** is omitted, the address gives the entry point of the exit routine in virtual storage. **L** may precede or follow the **A** or **N** specification.

COPIES=*number*
specifies the number of copies of the exit list you want generated. GENCB generates as many copies as you specify (default is 1) when your program is executed. All of the copies are the same. You can use MODCB to change some or all of the addresses in a list. MODCB is described later in this chapter.

LENGTH=*number*
specifies the length, in bytes, of the area, if any, that you are supplying for VSAM to generate the exit list(s). (See the WAREA operand.)

WAREA=*address*
specifies the address of an area in which the exit list(s) is to be generated. (Otherwise VSAM obtains virtual-storage space for the area and returns its address in register 1 and its length in register 0.) The area must begin on a fullword boundary. This operand is paired with the LENGTH operand, which must be given if you specify an area address.

If you do not specify an area in which the exit list is to be generated, VSAM returns to your program the address of the area in which the exit list(s) is generated in register 1, and the length of the area in register 0. You can find out the length of each exit list by dividing the length of the area by the number of copies. The address of each exit list can then be calculated by this offset from the address in register 1. You can find the length of an exit list with the SHOWCB macro, described under "SHOWCB Macro (Display an Exit List)" later in this chapter.

If you are generating control blocks by issuing several GENCBs, specifying an area (WAREA and LENGTH) for them enables you to address all of them with one base register and to avoid repetitive requests for virtual storage.

Example: GENCB Macro (Generate an Exit List)

In this example, a GENCB macro is used to generate an exit list when the program is executed.

```
EXITS    GENCB  BLK=EXLST,
                EODAD=(EOD,N),
                LERAD=LOGICAL,
                SYNAD=(ERROR,
                A,L)
         LTR    R15,R15
         BNZ    ERROR
         ST     1,EXLSTADR        Address of the exit list is saved.
EOD                               EODAD routine.
LOGICAL                           LERAD routine.
ERROR    DC     C'PHYSICAL'       Name of the SYNAD module.
EXLSTADR DS     F                 Save area for exit-list address.
```

The GENCB macro's operands are:

- BLK, which specifies that an exit list is to be generated.
- EODAD, which specifies that the end-of-data routine is located at EOD and is not active.
- LERAD, which specifies that the logical-error routine is located at LOGICAL; because neither **A** nor **N** is specified, the LERAD routine is marked active by default.
- SYNAD, which specifies that the physical-error routine's name is located at ERROR.

Because no area was specified in which the exit list was to be generated, VSAM obtained virtual storage for the exit list and returned the address in register 1. Immediately after the GENCB macro, the address of the exit list, contained in register 1, is moved to EXLSTADR. EXLSTADR may be specified in a GENCB macro that generates an access-method control block or in a MODCB, SHOWCB, or TESTCB macro that modifies, displays, or tests fields in an exit list.

GENCB Macro (Generate a Request Parameter List)

After you have connected your program to the data set, you can issue requests for access to it. A *request parameter list* defines a request. Each request macro (GET, PUT, ERASE, POINT, CHECK, and ENDREQ) gives the address of a request parameter list that defines the request.

The GENCB macro can be used to generate the request parameter list when the program is executed. Using the GENCB macro gives you independence from possible changes in the format of the request parameter list in future releases of VSAM. It also gives you the ability to generate many copies of the list.

If you use GENCB to generate request parameter lists as you need them, and later free the space, you should first issue the ENDREQ macro for each request parameter list to free the VSAM resources used for keeping track of positions in the data set.

The operands of the GENCB macro to generate a request parameter list are optional in some cases, but required in others. It is not necessary to omit operands that are not required for a request; they are ignored. Thus, for example, if you switch from direct to sequential retrieval with a request parameter list, you don't have to zero out the address of the field containing the search argument (ARG=address).

The operands of the GENCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. "Appendix C: Operand Notation for GENCB, MODCB, SHOWCB, and TESTCB" gives all the ways of coding each operand for the macros that work at execution.

See "Return Codes from the GENCB, MODCB, SHOWCB, and TESTCB Macros" for information on the return codes used to indicate whether the GENCB request was successful.

The format of the GENCB macro used to generate a request parameter list is:

[*label*]	**GENCB**	**BLK=RPL** [,**ACB=** *address*] [,**AM=VSAM**] [,**AREA=** *address*] [,**AREALEN=** *number*] [,**ARG=** *address*] [,**COPIES=** *number*] [,**ECB=** *address*] [,**KEYLEN=** *number*] [,**LENGTH=** *number*] [,**MSGAREA=** *address*] [,**MSGLEN=** *number*] [,**NXTRPL=** *address*] [,**OPTCD=**([**ADR** | **CNV** | **KEY**] [,**DIR** | **SEQ** | **SKP**] [,**ARD** | **LRD**] [,**FWD** | **BWD**] [,**ASY** | **SYN**] [,**NSP** | **NUP** | **UPD**] [,**KEQ** | **KGE**] [,**FKS** | **GEN**] [,**LOC** | **MVE**])] [,**RECLEN=** *number*] [,**TRANSID=** *number*] [,**WAREA=** *address*]

where:

label
is one to eight characters that provides a symbolic address for the GENCB macro. See the discussion of the COPIES operand for addressing lists generated by GENCB.

BLK=RPL
specifies that you are generating a request parameter list.

ACB=*address*
specifies the address of the access-method control block that identifies the data set to which access will be requested. If you omit this operand, you must issue MODCB to specify the address of the access-method control block before you issue a request. MODCB is described later in this chapter.

AM=VSAM
specifies that the access method using this control block is VSAM.

AREA=*address*
specifies the address of a work area to and from which VSAM moves a data record if you request it to do so (with the RPL operand OPTCD=MVE). If you request to process records in the I/O buffer (OPTCD=LOC), VSAM puts into this work area the address of a data record within the I/O buffer.

AREALEN=*number*
specifies the length, in bytes, of the work area whose address is specified by the AREA operand. Its minimum for OPTCD=MVE is the size of a data record (or the largest data record, for a data set with records of variable length). For OPTCD=LOC the area should be 4 bytes to contain the address of a data record within the I/O buffer.

ARG=*address*
specifies the address of a field that contains the search argument for direct retrieval, skip-sequential retrieval, and positioning. For a relative record data set, the ARG field must be 4 bytes long. For direct or skip-sequential processing, this field contains your search argument, a relative record number. For sequential processing (OPTCD=(KEY,SEQ)), the 4 bytes are required for VSAM to return the feedback RRN. For keyed access (OPTCD=KEY), the search argument is a full or generic key; for addressed access (OPTCD=ADR), it is an RBA. If you specify a generic key (OPTCD=GEN), you must also specify in the KEYLEN operand how many of the bytes of the full key you are using for the generic key.

COPIES=*number*
specifies the number of copies of the request parameter list you want generated. GENCB generates as many copies as you specify (default is 1) when your program is executed.

The copies of a request parameter list can be used to:

- Chain lists together to gain access to many records with one request.
- Define many requests to gain access to many parts of a data set concurrently.

All of the copies generated are identical; you have to use MODCB to tailor them to specific requests. MODCB is described in this chapter.

ECB=*address*
specifies the address of an event control block (ECB) that you may supply. VSAM indicates in the ECB whether a request is complete or not (using standard OS/VS completion codes, which are described in *OS/VS1 System Data Areas,* and *OS/VS2 Data Areas).* This operand is always optional.

KEYLEN=*number*
specifies the length, in bytes, of the generic key (OPTCD=GEN) you are using for a search argument (given in the field addressed by the ARG operand). This operand is required with a search argument that is a generic key. The *number* can be 1 through 255. For full-key searches, VSAM knows the key length, which is taken from the catalog definition of the data set when you open the data set.

LENGTH= *number*
specifies the length, in bytes, of the area, if any, that you are supplying for VSAM to generate the request parameter list(s). (See the WAREA operand.) You can find out how long a request parameter list is with the SHOWCB macro, described later in this chapter.

MSGAREA=*address*

specifies the address of an area that you are supplying for VSAM to send you a message in case of a physical error. The format of a physical-error message is given under "Physical Errors" in the chapter "Request Macros." This operand is always optional.

MSGLEN=*number*

specifies the size, in bytes, of the message area indicated in the MSGAREA operand. The size of a message is 128 bytes; if you provide less than 128 bytes, no message is returned to your program. This operand is required when MSGAREA is coded.

NXTRPL=*address*

specifies the address of the next request parameter list in a chain. Omit this operand from the macro that generates the only or last list in the chain. When you issue a request that is defined by a chain of request parameter lists, indicate in the request macro the address of the first parameter list in the chain. A single request macro can be defined by multiple request parameter lists, such that a GET, for example, can cause VSAM to retrieve two or more records.

OPTCD=([ADR | CNV | KEY]
[,DIR | SEQ | SKP]
[,ARD | LRD]
[,FWD | BWD]
[,ASY | SYN]
[,NSP | NUP | UPD]
[,KEQ | KGE]
[,FKS | GEN]
[,NWAIT | WAITX)
[,LOC | MVE])

specifies the options that govern the request defined by the request parameter list. Each group of options has a default; options are shown in Figure 10 with defaults underlined. Only one option from each group is effective for a request. Some requests do not require an option from all of the groups to be specified. The groups that aren't required are ignored; thus, you can use the same request parameter list for a combination of requests (GET, PUT, POINT, for example) without zeroing out the inapplicable options each time you go from one request to another.

RECLEN=*number*

specifies the length, in bytes, of a data record being stored. With fixed-length records, set it and forget it. This operand is required for PUT requests. For GET requests, VSAM puts the length of the record retrieved in this field in the request parameter list. It will be there if you update and store the record.

TRANSID= *number*

specifies a number that relates modified buffers in a buffer pool. Used in shared resource applications and described in *OS/VS Virtual Storage Access Method (VSAM) Options for Advanced Applications.*

WAREA=*address*

specifies the address of an area in which the request parameter list(s) is to be generated. (Otherwise VSAM obtains virtual-storage space for the area and returns its address to you in register 1 and its length in register 0.) The area must begin on a fullword boundary. This operand is paired with the LENGTH operand, which must be given if you specify an area address.

If you do not specify an area in which the request parameter list is to be generated, VSAM returns to your program the address of the area in which the request parameter list(s) was generated in register 1, and the length of the area in register 0. You can find the length of each list by dividing the length of the area by the number of copies. You can then calculate the address of each list by using the length of each list as an offset.

If you are generating control blocks by issuing several GENCBs, specifying an area (WAREA and LENGTH parameters) for them enables you to address all of them with one base register and to avoid repetitive requests for virtual storage.

You can use the ECB to determine that an asynchronous request is complete before issuing a CHECK macro. (If you issue a CHECK before a request is complete, you give up control and must wait for completion.) You can also test for completion with the TESTCB I/O=COMPLETE operand.

When GENCB is used to build a chain of request parameter lists, the request parameter lists may be chained using only GENCB macros or using GENCB and MODCB macros together. When only GENCB is used, the request parameter lists are created in reverse order, as follows:

```
SECOND   GENCB BLK=RPL
         LR 2,1
FIRST    GENCB BLK=RPL,NXTRPL=(2)
```

SECOND GENCB creates the second request parameter list, which makes its address available for the first request parameter list. The address of the request parameter list is returned in register 1 and is loaded into register 2. FIRST GENCB creates the first request parameter list and supplies the address of the next request parameter list using register notation. GENCB and MODCB macros may be used together to create a chain of request parameter lists, as follows:

```
GENCB    BLK=RPL,COPIES=2
LR       2,0
SRL      2,1
LR       3,1
LA       4,0(2,3)
MODCB    RPL=(3),NXTRPL=(4)
```

The GENCB macro creates two request parameter lists. The length of the parameter lists is returned in register 0 and loaded into register 2. The address of the area in which the lists were created (and, therefore, the address of the first one) is returned in register 1 and loaded into register 3. The SRL statement divides the total length of the area (register 2) by 2. The LA statement loads the address of the second request parameter list into register 4. The MODCB macro modifies the first request parameter list (register 3) by supplying the address of the second request parameter list (register 4) in the NXTRPL operand.

Each request parameter list in a chain should have the same OPTCD options. Having different options may cause logical errors. You can't chain request parameter lists for updating or deleting records—only for retrieving records or storing new records. You can't process records in the I/O buffer with chained request parameter lists. (OPTCD=UPD and LOC are invalid for a chained request parameter list.)

With chained request parameter lists, a POINT, a sequential or skip-sequential GET, or a direct GET with positioning requested

(OPTCD=NSP) causes VSAM to position itself at the record following the record identified by the last request parameter list in the chain.

Example: GENCB Macro (Generate a Request Parameter List)

In this example, a GENCB macro is used to generate a request parameter list.

```
ACCESS   GENCB  BLK=RPL,
                ACB=ACCESS,
                AM=VSAM,
                AREA=WORK,
                AREALEN=125,
                ARG=SEARCH,
                MSGAREA=MESSAGE,
                MSGLEN=128,
                OPTCD=(SKP,UPD)
         .
         .
         .
ACCESS   ACB    MACRF=(SKP,OUT)
WORK     DS     CL125
SEARCH   DS     CL8
MESSAGE  DS     CL128
```

The GENCB macro's operands are:

- BLK, which specifies that a request parameter list is to be generated.
- ACB, which specifies that the request parameter list is associated with a data set and processing options identified by ACCESS.
- AREA and AREALEN, which specify a 125-byte work area to be used for processing records.
- ARG, which specifies the address of the search argument.
- MSGAREA and MSGLEN, which specify a 128-byte area to be used for physical-error messages.
- OPTCD, which specifies the options that govern the request defined by the request parameter list identified by SKP and UPD.

GET Macro (Retrieve a Record)

The GET macro is used to retrieve a record.

The GET macro is used with the PUT macro to update records. See "PUT Macro (Store a Record)" later in this chapter for examples that show the use of the GET macro to update records. The GET macro is used with the ERASE macro to delete records in a key-sequenced or relative record data set. See "ERASE Macro (Delete a Record)" in this chapter for examples that show the use of the GET macro to delete records.

You cannot update records in the I/O buffer. A direct GET for update positions VSAM at the record retrieved, in anticipation of storing back (or deleting) the record. This positioning allows you to switch to sequential access to retrieve another record.

You are not required to store back a record that you retrieve for update; however, another GET request nullifies any previous positioning for deletion or update.

The format of the GET macro is:

[*label*]	**GET**	**RPL=** *address*

where:

label
is one to eight characters that provides a symbolic address for the GET macro.

RPL= *address*
specifies the address of the request parameter list that defines this GET request. You may specify the address in register notation (using a register from 1 through 12, enclosed in parentheses) or specify it with an expression that generates a valid relocatable A-type address constant.

Example: Keyed-Sequential Retrieval (Forward)

In this example, a GET macro is used to sequentially retrieve records by key. Retrieval is in a forward direction. Fixed-length, 100-byte records are moved to a work area. Processing is synchronous.

```
INPUT    ACB     MACRF=(KEY,          All MACRF and OPTCD options
                 SEQ,IN)              specified are defaults and could have
                                      been omitted.
RETRVE   RPL     ACB=INPUT,
                 AREA=IN,
                 AREALEN=100,
                 OPTCD=(KEY,SEQ,
                 SYN,NUP,MVE)
                 .
                 .
                 .
LOOP     GET     RPL=RETRVE           This GET or identical GETs can be
                                      issued, with no change in the request
                                      parameter list, to retrieve subsequent
                                      records in key sequence.
         LTR     15,15
         BNZ     ERROR
                 .
                 .
                 .
         B       LOOP
ERROR    ...                          Request wasn't accepted or failed.
                 .
                 .
                 .
IN       DS      CL100                IN contains a data record after GET is
                                      completed.
```

The records are retrieved in key sequence in a forward direction. No search argument has to be specified; VSAM is positioned at the first record in key sequence when the data set is opened, and the next record is retrieved automatically as each GET is issued. The branch to ERROR could also be taken if the end of the data set is reached.

Example: Keyed-Sequential Retrieval (Backward)

This example is the same as the previous one except that a POINT macro instruction is issued to the last record in the data set and the records are retrieved in a backward direction.

```
INPUT    ACB     DDNAME=INPUT,
                 EXLST=EXLST1
RETRVE           RPL                    Define RPL for last record positioning
                 ACB=INPUT,             and backward processing
                 AREA=IN,
                 AREALEN=100,
                 OPTCD=(KEY,SEQ,
                 LRD,BWD)
EXLST1   EXLST   EODAD=EOD              Define end of data
         POINT   RPL=RETRVE             Position to last record (no argument is
                                        required)
         LTR     15,15
         BNZ     ERROR
LOOP     GET     RPL=RETRVE             Get previous record
         LTR     15,15
         .       LOOP
         .
         .
         B
EOD      EQU     *                      Come here for end of data
ERROR    ...                            Request failed
         .
         .
         .
IN       DS      CL100                  Area for retrieved record
```

Example: Skip-Sequential Retrieval

In this example, a GET macro is used to retrieve variable-length records synchronously. Records are to be processed in the I/O buffer. The search argument is full key, compared greater-than-or-equal; key length is eight bytes.

The records are retrieved in key sequence, but some records are skipped. Skip-sequential retrieval is very similar to keyed-direct retrieval, except that you must retrieve records in ascending sequence (with skips) rather than in a random sequence.

```
        GENCB  BLK=ACB,                 VSAM gets an area in virtual storage to
               DDNAME=INPUT,            generate the access-method control
               MACRF=(KEY,              block and returns the address in
               SKP,IN)                  register 1.
        LTR    15,15
        BNZ    CHECK0
        LR     2,1
        GENCB  BLK=RPL,
               ACB=(2),
               AREA=RCDADDR,
               AREALEN=4,
               ARG=SRCHKEY,
               OPTCD=(KEY,SKP,
               SYN,NUP,KGE,
               FKS,LOC)
        LTR    15,15
        BNZ    CHECK0
        LR     3,1                      Address of the request parameter list.
        .
        .
        .
LOOP    MVC    SRCHKEY,source           Search argument for retrieval, moved
                                        in from a table or a transaction record.
        GET    RPL=(3)
        LTR    15,15
        BNZ    ERROR
        SHOWCB AREA=RCDLEN,             Display the length of the record.
               FIELDS=RECLEN,
               LENGTH=4,
               RPL=(3)
        LTR    15,15
        BNZ    CHECK0
        .
        .
        .
        B      LOOP
ERROR   ...                             Request wasn't accepted or failed.
CHECK0  ...                             Generation or display failed.
        .
        .
        .
```

RCDADDR	DS	F	Work area into which VSAM puts the address of a data record within the I/O buffer (OPTCD=LOC).
SRCHKEY	DS	CL8	Search argument for retrieval.
RCDLEN	DS	F	For displaying variable record lengths.

The macros and instructions are described, as follows:

- The first GENCB generates an access-method control block, which specifies keyed, skip-sequential, and input processing. The address of the access-method control block is stored in register 2.
- The second GENCB generates a request parameter list. The address of the request parameter list is stored in register 3.
- MVC moves the search argument into SRCHKEY, the area defined for the search argument.
- GET specifies that the record pointed at by the request parameter list whose address is in register 3 is to be retrieved. Records are retrieved by a skip-sequential search through the sequence set of the index.

Example: Addressed-Sequential Retrieval

In this example, one GET macro is used to retrieve multiple fixed-length, 20-byte records. The records are moved to a work area (only option).

```
BLOCK    ACB     DDNAME=INPUT,
                 MACRF=(ADR,SEQ,
                 IN)
                 .
                 .
                 .
         GENCB   BLK=RPL,
                 COPIES=10,
                 ACB=BLOCK,
                 OPTCD=(ADR,SEQ,
                 SYN,NUP,MVE)
         LTR     15,15
         BNZ     CHECK0
         LA      3,10          Number of lists (10).
         LR      2,1           Address of the first list.
         LR      1,0           Length of all of the lists. Registers 0
                               and 1 contain length and address of the
                               generated control blocks when VSAM
                               returns control after GENCB.
         SR      0,0           Prepare for following division.
         DR      0,3           Divide number of lists into length of all
                               of the lists.
         LR      3,1           Save the resulting length of a single list
                               for an offset.
         LR      4,2           Save address of the first list.
         LA      5,RECAREA     Address of the first work area.
                 .             Do the following six instructions ten
                 .             times to set up all of the request
                 .             parameters lists. The tenth time,
                               register 4 must be set to 0 to indicate
                               the last request parameter list in the
                               chain.
         AR      4,3           Address the next list.
         MODCB   RPL=(2),      In each request parameter list, indicate
                 NXTRPL=(4),   the address of the next list and the
                 AREA=(5),     address and length of the work area.
                 AREALEN=20
         LTR     15,15
         BNZ     CHECK0
         AR      2,3           Address the next list.
         LA      5,20(5)       Address the next work area.
                 .             Restore register 2 to address the first
                 .             list before continuing to process.
                 .
LOOP     GET     RPL=(2)
         LTR     15,15
         BNZ     ERROR
                 .             Process the ten records that have been
                 .             retrieved by the GET.
                 .
         B       LOOP
```

```
CHECK0   ...
ERROR    ...                          Display the feedback field
                                      (FIELDS=FDBK) of each request
                                      parameter list to find out which one
                                      had an error.
RECAREA  DS       CL200               Space for a work area for each of the
                                      ten request parameter lists.
```

The GENCB macro generates ten request parameter lists; the lists are subsequently chained together by using the MODCB macro to modify the NXTRPL operand in each copy. Because SEQ is specified in each request parameter list and no previous request has been issued against the access-method control block since it was opened, retrieval begins at the beginning of the data set. Each time the GET macro is executed, VSAM is positioned at the next record in RBA sequence. VSAM moves each record into the work area provided for the request parameter list that identifies the record.

If an error occurred for one of the request parameter lists in the chain and you have supplied error-analysis routines, VSAM takes a LERAD or SYNAD exit before returning to your program. Register 15 is set to indicate the status of the request. A code of 0 indicates that no error was associated with any of the request parameter lists. Any other code indicates that an error occurred for one of the request parameter lists. You should issue a SHOWCB macro for each request parameter list in the chain to find out which one had an error. VSAM doesn't process any of the request parameter lists beyond the one with an error.

Example: Sequential Retrieval for a Relative Record Data Set

In this example, a GET macro is used to sequentially retrieve records by relative record number. Fixed-length, 100-byte records are moved to a work area. Processing is synchronous.

```
INPUT    ACB     MACRF=(KEY,SEQ,      All MACRF and OPTCD options
                 IN)                  specified are defaults and could have
                                      been omitted
RETRVE   RPL     ACB=INPUT,
                 AREA=IN,
                 AREALEN=100,
                 ARG=RCDNO,
                 OPTCD=(KEY,SEQ,
                 SYN,
NUP,MVE) .
         .
         .
LOOP     GET     RPL=RETRVE           This GET or identical GETs can be
                                      issued, with no change in the RPL, to
                                      retrieve subsequent records in relative
                                      record number sequence
         LTR     15,15
         BNZ     ERROR
         .       LOOP
         .
         .
         B
ERROR    ...                          Request was not accepted or it failed
         .
         .
         .
IN       DS      CL100                IN contains a data record after GET is
                                      completed.
RCDNO    DS      CL4                  VSAM returns relative record number
                                      of retrieved record in this field.
```

The records are retrieved in relative record number sequence. Empty records are bypassed for sequential retrieval. A four-byte search argument must be specified. The relative record number of each record retrieved is stored in the search argument. VSAM is positioned at the first relative record when the data set is opened, and the next nonempty record is retrieved automatically as each GET is issued. The branch to ERROR would also be taken if the end of the data set is reached.

Example: Keyed-Direct Retrieval

In this example, a GET macro is used to retrieve fixed-length, 100-byte records directly by key. The key length is 15 bytes; the search argument is a five-byte generic key, compared equal. The control blocks are generated at assembly.

```
INPUT    ACB    MACRF=(KEY,
                DIR,IN)
RETRVE   RPL    ACB=INPUT,          You specify all parameters for the
                AREA=IN,            request in the RPL macro.
                AREALEN=4,
                OPTCD=(KEY,DIR,
                SYN,NUP,KEQ,
                GEN,OPTCD=(KEY,
                DIR,SYN,NUP,
                KEQ,GEN,LOC),
                ARG=KEYAREA,
                KEYLEN=5
         .
         .
         .
LOOP     MVC    KEYAREA,source      Search argument for retrieval, moved
                                    in from a table or a transaction record.
         GET    RPL=RETRVE          This GET or identical GETs can be
                                    issued with no change in the RPL: just
                                    specify each new search argument in
                                    the field KEYAREA.
         LTR    15,15
         BNZ    ERROR
         .                          Process the record.
         .
         .
         B      LOOP
ERROR    ...                        Request wasn't accepted or failed.
         .
         .
         .
IN       DS     CL4                 VSAM puts here the address of the
                                    record within the I/O buffer.
KEYAREA  DS     CL5                 You specify the search argument here.
```

The generic key specifies a class of records. For example, if you search on the first third of employee number, VSAM positions at and retrieves the first of presumably several records that start with the specified characters. To retrieve all of the records in that class, either switch to sequential access or to a full-key search with greater-than-or-equal comparison.

Example: Addressed-Direct Retrieval

In this example, a GET macro is used to retrieve fixed-length, 20-byte records. The records are to be moved to a work area.

```
BLOCK    ACB     DDNAME=INPUT,              Access-method control block generated
                 MACRF=(ADR,DIR,            at assembly.
                 IN)
           .
           .
           .
         GENCB   BLK=RPL,                   ARG=SRCHADR,AREA=IN,
                 COPIES=1,                  AREALEN=20
                 ACB=BLOCK,                 Request parameter list generated at
                 OPTCD=(ADR,DIR,            execution.
                 SYN,NUP,MVE),
         LTR     15,15
         BNZ     CHECK0
         LR      2,1                        Address of the list.
           .
           .
           .
LOOP     MVC     SRCHADR,source             Search argument for retrieval,
                                            calculated or moved in from a table or
                                            a transaction record.
         GET     RPL=(2)
         LTR     15,15
         BNZ     ERROR
           .                                Process the record.
           .
           .
         B       LOOP
CHECK0   ...                                Generation failed.
ERROR    ...                                Request wasn't accepted or failed.
           .
           .
           .
IN       DS      CL20                       VSAM puts a record here for each GET
                                            request.
SRCHADR  DS      CL4                        You specify the RBA search argument
                                            here for each request.
```

The RBA provided for a search argument must match the RBA of a record. Keyed insertion and deletion of records in a key-sequenced data set will probably cause the RBAs of some records to change. Therefore, if you process a key-sequenced data set by addressed-direct access (or by addressed-sequential access using POINT), you need to keep track of changes. You can use the JRNAD exit for this purpose. See "EXLST Macro (Generate an Exit List)" in this chapter.

Example: Switch from Direct to Sequential Retrieval

In this example, GET macros are used to retrieve fixed-length, 100-byte records. The retrieval is via an alternate index path which is defined with the nonunique key option. Every time a nonunique key is retrieved, the program switches to sequential processing to retrieve the other records with the same key. The control blocks were generated at assembly, but the MODCB macro is used to modify the request parameter list to permit switching from keyed-direct to keyed-sequential retrieval. For the direct request preceding sequential requests, the search argument is an eight-byte, generic key, compared equal. Positioning is requested for direct requests.

```
INPUT    ACB     MACRF=(KEY,DIR,          Both direct and sequential access
                 SEQ,IN)                  specified.
RETRVE   RPL     ACB=INPUT,               NSP specifies that VSAM is to
                 AREA=IN,                 remember its position.
                 AREALEN=100,
                 OPTCD=(KEY,DIR,
                 SYN,NSP,KEQ,
                 GEN,MVE),
                 ARG=KEYAREA,
                 KEYLEN=8
                 .
                 .
                 .
LOOP     MVC     KEYAREA,source           Search argument for direct retrieval,
                                          moved in from a table or a transaction
                                          record.
LOOP1    GET     RPL=RETRVE
         LTR     15,15
         BNZ     ERROR
                 .                        Extract feedback information
                 .        RPL=RETRVE,
                 .        AREA=FDBAREA,
                          FIELDS=FDBK
         SHOWCB
         LTR     R15,R15
         BNZ     ERROR
         CLI     ERRCD,8                  Does a duplicate key follow
         BE      SEQ                      Yes; retrieve duplicates sequentially
         B       LOOP                     No; retrieve next record in direct mode
SEQ      MODCB   RPL=RETRVE,              Alter request parameter list for
                 OPTCD=SEQ                sequential access.
         LTR     15,15
         BNZ     CHECK0
SEQGET   GET     RPL=RETRVE               Do sequential retrieval
         LTR     15,15                    Test for error
         BNZ     ERROR
                 .
                 .
                 .
         SHOWCB  RPL=RETRVE,              Extract feedback information
                 AREA=FDBAREA,
                 FIELDS=FDBK
         LTR     R15,R15
```

```
         BNZ     ERROR
         CLI     ERRCD,8          Does a duplicate key follow
         BE      SEQGET           Yes; retrieve sequentially
DIR      MODCB   RPL=RETRVE,      Alter request parameter list for direct
                 OPTCD=DIR        access
         LTR     15,15
         BNZ     CHECK0
         B       LOOP             Prepare new search argument.
ERROR    ...                      Request wasn't accepted or failed
CHECK0           ...              Modification failed.
         .
         .
         .
IN       DS      CL100            VSAM puts retrieved records here.
KEYAREA  DS      CL8              Specify the generic key for a direct
                                  request here.
FDBAREA  DS      0F               Feedback area for SHOWCB
         DS      1C               Reserved
TYPECD   DS      1C               Error type code
CMPCD    DS      1C               Component Code
ERRCD    DS      1C               Error code
```

Positioning is associated with a request parameter list; the MODCB macro is used to modify a single request parameter list that alternately defines requests for both types of access rather than use a different request parameter list for each type.

With direct retrieval, VSAM doesn't remember its position for subsequent sequential retrieval unless you explicitly request it (OPTCD=NSP or UPD). After a direct GET for update, VSAM is positioned for a subsequent PUT, ERASE, or sequential GET. If you modify OPTCD=(DIR,NUP) to OPTCD=SEQ, you must issue POINT to get VSAM positioned for sequential retrieval, as NUP indicates that no positioning is desired with a direct GET.

If you have chained many request parameter lists together, one position is remembered for the whole chain. For example, if you issue a GET that gives the address of the first request parameter list in the chain, the position of VSAM when the GET request is complete is at the record following the record defined by the last request parameter list in the chain. Therefore, modifying OPTCD=(DIR,NSP) in each request parameter list in a chain to OPTCD=SEQ implies continuing with sequential access relative to the last of the direct request parameter lists.

MODCB Macro (Modify an Access-Method Control Block)

The MODCB macro can be used to modify the contents of an access-method control block. By using MODCB, you don't have to know the format of the control block.

MODCB allows you to tailor access-method control blocks generated with the GENCB macro for specific uses.

The operands of the MODCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. "Appendix C: Operand Notation for GENCB, MODCB, SHOWCB, and TESTCB" gives all the ways of coding each operand for the macros that work at execution.

See "Return Codes from the GENCB, MODCB, SHOWCB, and TESTCB Macros" for information on the return codes used to indicate whether the MODCB request was successful.

The format of the MODCB macro used to modify an access-method control block is:

[*label*]	**MODCB**	**ACB=***address* **[,BSTRNO=***number*] **[,BUFND=***number*] **[,BUFNI=***number*] **[,BUFSP=***number*] **[,CATALOG=YES | NO]** **[,CRA=SCRA | UCRA]** **[,DDNAME=***ddname*] **[,EXLST=***address*] **[,MACRF=([ADR][,CNV][,KEY]** **[,CFX | NFX]** **[,DDN | DSN]** **[,DFR | NDF]** **[,DIR][,SEQ][,SKP]** **[,ICI | NCI]** **[,IN][,OUT]** **[,NIS | SIS]** **[,NRM | AIX]** **[,NRS | RST]** **[,NSR | LSR | GSR]** **[,NUB | UBF])]** **[,MAREA=***address*] **[,MLEN=***number*] **[,PASSWD=***address*] **[,STRNO=***number*]

where:

label
is one to eight characters that provides a symbolic address for the MODCB macro.

ACB=*address*
specifies the address of the access-method control block to be modified. The data set identified by the access-method control block must not be opened. A request to modify the access-method control block of an open data set will fail.

The remaining operands represent operands of the ACB macro that can be modified. The value specified replaces the value, if any, presently in the access-method control block. There are no defaults. See "ACB Macro (Generate an Access-Method Control Block)" earlier in this chapter for an explanation of these operands.

If MODCB is used to modify a MACRF option, other options are unaffected, except when they are inconsistent. For example, if you specify MACRF=ADR in the MODCB and MACRF=KEY is already indicated in the control block, both ADR and KEY will now be indicated. But if you specify MACRF=UBF in the MODCB and NUB is indicated, only UBF will now be indicated.

If MODCB is used to change the address of an ACB, you must first issue an ENDREQ macro.

Example: MODCB Macro (Modify an Access-Method Control Block)

In this example, a MODCB macro is used to modify the name of the exit list in an access-method control block.

```
MODCB   ACB=BLOCK,          BLOCK was generated at assembly.
        EXLST=EGRESS
```

MODCB Macro (Modify an Exit List)

The MODCB macro can be used to modify an exit list.

The operands of the MODCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. "Appendix C: Operand Notation for GENCB, MODCB, SHOWCB, and TESTCB" gives all the ways of coding each operand for the macros that work at execution.

See "Return Codes from the GENCB, MODCB, SHOWCB, and TESTCB Macros" earlier in this chapter for information on the return codes used to indicate whether the MODCB request was successful.

The format of the MODCB macro used to modify an exit list is:

[*label*]	**MODCB**	**EXLST=***address* **[,EODAD=([***address***][,A | N][,L])]** **[,JRNAD=([***address***][,A | N][,L])]** **[,LERAD=([***address***][,A | N][,L])]** **[,SYNAD=([***address***][,A | N][,L])]**

where:

label
is one to eight characters that provides a symbolic address for the MODCB macro.

EXLST=*address*
specifies the address of the exit list to be modified. You can modify an exit list at any time—that is, before or after opening the data set(s) for which the list indicates exit routines. You cannot, however, add an entry to the exit list if it will change the exit list's length; the exit list must already be large enough to contain the new exit address. The order in which addresses are stored in the EXLST control block is: EODAD, SYNAD, LERAD, JRNAD, and UPAD. For example, if you generate an exit list with only the LERAD exit, you can add entries for EODAD and SYNAD later; you cannot add the JRNAD exit address because doing so would increase the size of the EXLST control block. The MODCB macro does not support the UPAD user exit.

The remaining operands represent operands of the EXLST macro that can be modified or added to an exit list. See "EXLST Macro (Generate an Exit List)" earlier in this chapter for an explanation of these operands.

Example: MODCB Macro (Modify an Exit List)

In this example, a MODCB macro is used to activate an exit in an exit list.

```
            MODCB   EXLST=( *,          Indirect notation is used to specify the
                    EXLSTADR),          address of the exit list, which was
                    EODAD=( EOD,L,A )   generated at execution.
                .
                .
                .
EOD         DC      C'ENDUP'
EXLSTADR    DS      F                   When the exit list was generated, its
                                        address was saved here.
```

The MODCB macro's operands are:

- EXLST, which specifies that the address of the exit list to be modified is located at EXLSTADR.
- EODAD, which specifies that the entry for the end-of-data routine is to be marked active in the exit list whose address resides at EXLSTADR. The name of the end-of-data routine, ENDUP, is located at EOD.

MODCB Macro (Modify a Request Parameter List)

The MODCB macro can be used to modify a request parameter list.

The operands of the MODCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. "Appendix C: Operand Notation for GENCB, MODCB, SHOWCB, and TESTCB" gives all the ways of coding each operand for the macros that work at execution.

See "Return Codes from the GENCB, MODCB, SHOWCB, and TESTCB Macros" for information on the return codes used to indicate whether the MODCB request was successful. Typical modifications to a request parameter list are to change the indication of length of a record (RECLEN) when you're processing a data set with variable-length records and to change the type of request (OPTCD), such as from direct to sequential access or from full-key search argument to generic-key search argument.

The format of a MODCB macro used to modify a request parameter list is:

[*label*]	**MODCB**	**RPL=** *address* [**,ACB=** *address*] [**,AREA=** *address*] [**,AREALEN=** *number*] [**,ARG=** *address*] [**,ECB=** *address*] [**,KEYLEN=** *number*] [**,MSGAREA=** *address*] [**,MSGLEN=** *number*] [**,NXTRPL=** *address*] [**,OPTCD=([ADR | CNV | KEY]** **[,DIR | SEQ | SKP]** **[,ARD | LRD]** **[,FWD | BWD]** **[,ASY | SYN]** **[,NSP | NUP | UPD]** **[,KEQ | KGE]** **[,FKS | GEN]** **[,LOC | MVE])]** [**,RECLEN=** *number*] [**,TRANSID=** *number*]

where:

label
is one to eight characters that provides a symbolic address for the MODCB macro.

RPL=*address*
specifies the address of the request parameter list to be modified. You may not modify an active request parameter list; that is, one that defines a request that has been issued but not completed. To modify such a request parameter list, you must first issue a CHECK or an ENDREQ macro.

The remaining operands represent operands of the RPL macro that can be modified. The value specified replaces the value, if any, presently in the request parameter list. There are no defaults. See "GENCB Macro (Generate a Request Parameter List)" earlier in this chapter for an explanation of these operands.

If MODCB is used to modify an OPTCD option within a group of options, the current option for that group is changed, because only one option in a group is effective at a time. Only the OPTCD option specified is changed; all other OPTCD options remain unchanged.

Example: MODCB Macro (Modify a Request Parameter List)

In this example, a MODCB macro is used to modify the record-length field in a request parameter list.

```
L       3,length          Load the new record length.
MODCB   RPL=(2),          Register 2 contains the address of the
                          request parameter list. Register 3
                          contains the record length.
        RECLEN=(3)
```

The MODCB macro's operands are:

- RPL, which specifies that register 2 contains the address of the request parameter list to be modified.
- RECLEN, which specifies that the record-length field is to be modified. The contents of register 3 will replace any current value in the RECLEN field.

OPEN Macro (Connect Program and Data)

Before your program can issue requests for access to a data set, it must open the data set for processing. Opening a data set causes VSAM to have the volume(s) on which it is stored mounted if necessary and to verify that the data set matches the description implied by the ACB or GENCB macro (for example, MACRF=KEY implies that the data set is a key-sequenced data set).

OPEN causes VSAM to construct control blocks (other than those you caused to be built by the ACB, EXLST, and GENCB macros) that it needs to process your requests for access to the data set. It determines what processing options are to be used by merging the information in the DD statement and the catalog definition of the data set with the information in the access-method control block and the exit list. The order of precedence is:

1. The DD-statement AMP parameters
2. The ACB, EXLST, or GENCB operands
3. The catalog entry for the data set

For example, if information about buffer space is specified both in the DD statement and in the ACB or GENCB macro, the values in the DD statement override those in the macro. Catalog information acts as a default when buffer space specified in the DD statement or in the macro is less than the minimum specified when the data set was defined or when buffer space is specified in neither the DD statement nor the macro. By examining the DD statement indicated by the ACB macro and the volume information in the catalog, VSAM calls for the necessary volumes to be mounted. If you are opening a key-sequenced data set, an alternate index, or a path. Open checks for consistency of updates of the prime index and data components. If a data set and its index have been updated separately. VSAM issues a warning message to indicate a time stamp discrepancy.

VSAM also checks the password that your program specified against the appropriate password (if any) in the catalog definition of the data set. The password required depends on the kind of access specified in the access-method control block (for example, access for retrieval or for update), as follows:

- Full access allows you to perform all operations (retrieving, updating, inserting, and deleting) on a data set and any index or catalog record associated with it. The master password allows you to delete or alter the catalog entry for the data set or catalog it protects.
- Control-interval access requires the control password. The control password allows you to use control-interval access and to retrieve, update, insert, or delete records in the data set it protects. See *OS/VS Virtual Storage Access Method (VSAM) Options for Advanced Applications,* for information on the use of control-interval access.
- Update access requires the update password. The update password allows you to retrieve, update, insert, or delete records in the data set it protects.

- Read access requires the read password. The read password allows you to examine records in the data set it protects; the read password does not allow you to add, change, or delete records.

A password of one level authorizes you to do everything that a password of a lower level authorizes you to do. Password protection is further described in the appropriate *Access Method Services* publication.

In addition to passwords, you can have the access control measures provided by the Resource Access Control Facility (RACF), an IBM program product for use with MVS systems only. When RACF protection and password protection are both applied to a data set, password protection is bypassed, and use is authorized solely through the RACF checking system. RACF is described in *OS/VS2 MVS Resource Access Control Facility (RACF) General Information Manual.*

The format of the OPEN macro is:

[*label*]	**OPEN**	(*address*,[(*options*)],...)

where:

label
is one to eight characters that provides a symbolic address for the OPEN macro.

address
specifies the address of the access-method control block or DCB for the data set(s) to be opened. You may specify the address in register notation (using a register from 2 through 12—in parentheses) or specify it with an expression that generates a valid relocatable A-type address constant. If you use register notation to open only one data set, you must enclose the expression identifying the register in two sets of parentheses: for example, OPEN ((2)).

options
are options parameters for use only in opening nonVSAM data sets. If any options are specified with the address of an access-method control block, VSAM ignores them.

Because the OPEN operands are positional, include a comma for options (even if you don't specify options) before a subsequent operand.

Example: OPEN Macro

In this example, an OPEN macro is used to open two data sets. The access-method control block for one data set was generated at execution; the other was generated at assembly.

```
        GENCB   BLK=ACB,               An access-method control block.
                DDNAME=DATA
        LTR     R15,R15
        BNZ     ERROR
        LR      2,1                     Address of the control block.
        OPEN    (BLOCK,,(2))            A label is used for the access-method
                                        control block generated by ACB;
                                        register notation is used for the one
                                        generated by GENCB. The two
                                        commas indicate the omission of
                                        options.
BLOCK   ACB                             Another access-method control block.
```

POINT Macro (Position for Access)

The POINT macro is used to position for access.

The format of the POINT macro is:

[*label*]	**POINT**	**RPL=** *address*

where:

label
is one to eight characters that provides a symbolic address for the POINT macro.

RPL= *address*
specifies the address of the request parameter list that defines the request. You may specify the address in register notation (using a register from 1 through 12, enclosed in parentheses) or specify it with an expression that generates a valid relocatable A-type address constant.

Example: Position with POINT

In this example, the POINT macro is used to position at a record identified by a full key (five-byte) search argument, compared equal.

```
BLOCK    ACB     DDNAME=IO              Default MACRF options sufficient.
POSITION RPL     ACB=BLOCK,             ARG operand and KEQ and FKS
                 AREA=WORK,             OPTCD options define the POINT
                 AREALEN=50,            request.
                 ARG=SRCHKEY,
                 OPTCD=(KEY,SEQ,
                 SYN,KEQ,FKS)
                 .
                 .
                 .
LOOP     MVC     SRCHKEY,source         Search argument for positioning,
                                        moved in from a table or a transaction
                                        record.
         POINT   RPL=POSITION
         LTR     15,15
         BNZ     ERROR
LOOP1    GET     RPL=POSITION
         LTR     15,15
         BNZ     ERROR
Process the record. Decide whether to skip to
another position (forward or backward).
         B       LOOP                   Yes, skip.
         B       LOOP1                  No, continue in consecutive sequence.
ERROR    ...                            Request wasn't accepted or failed.
         .
         .
         .
SRCHKEY  DS      CL50                   VSAM puts a record here for each GET
                                        request.
WORK     DS      CL50                   VSAM puts a record here for each GET
                                        request.
```

No access is gained to a record with POINT. POINT causes VSAM to be positioned ahead or back to the specified record for a subsequent sequential GET request, which retrieves the record. If, after positioning, you issue a direct request by way of the same request parameter list, VSAM doesn't remember the position established by the POINT. VSAM would then either be positioned somewhere else or not positioned at all, depending on whether OPTCD=NSP or UPD was specified or OPTCD=NUP.

Positioning by address is identical to positioning by key, except that the search argument is an RBA, which must be matched equal to the RBA of a record in the data set.

When a POINT is followed by a VSAM GET/PUT request, both the POINT and the subsequent request must be in the same processing mode. For example, a POINT with OPTCD=(KEY,SEQ,FWD) must be followed by GET/PUT with OPTCD(KEY,SEQ,FWD); otherwise, the GET/PUT request will be rejected.

PUT Macro (Store a Record)

The PUT macro is used to store a record.

The format of the PUT macro is:

[*label*]	**PUT**	**RPL=** *address*

where:

label
is one to eight characters that provides a symbolic address for the PUT macro.

RPL= *address*
specifies the address of the request parameter list that defines the request. You may specify the address in register notation (using a register from 1 through 12, enclosed in parentheses) or specify it with an expression that generates a valid relocatable A-type address constant.

Note: If the PUT macro is being used to load records into an empty data set, the STRNO value in the access-method control block must be 1, and RPL OPTCD=DIR must not be specified. However, for an empty RRDS, DIR is allowed.

Example: Keyed-Sequential Insertion

In this example, a PUT macro is used to perform keyed-sequential insertion. Variable-length records with a key length of 15 bytes are to be moved from a work area. Some records will be inserted between existing records; other records will be added at the end of the data set.

```
BLOCK     ACB     DDNAME=OUTPUT,
                  MACRF=(KEY,SEQ,
                  OUT)
LIST      RPL     ACB=BLOCK,
                  AREA=BUILDRCD,
                  AREALEN=250,
                  OPTCD=(KEY,SEQ,
                  SYN,NUP,MVE)
                .
                .
                .
LOOP      L       2,source          Put length of record to be inserted into
                                    register
                  2.
          MODCB   RPL=LIST,         Indicate record length in request
                  RECLEN=(2)        parameter list.
          LTR     15,15
          BNZ     CHECK0
          PUT     RPL=LIST
          LTR     15,15
          BNZ     ERROR
          B       LOOP
CHECK0    ...                       Modification failed.
ERROR     ...                       Request wasn't accepted or failed.
                .
                .
                .
BUILDRCD  DS      CL250             Work area for building records.
```

The request parameter list, LIST, is associated with the access-method control block, BLOCK. The length of each record to be inserted is put into register 2, which is subsequently used by MODCB to change the record length in the request parameter list. The record length is, therefore, correctly indicated in the request parameter list before the PUT macro is issued. The execution of the PUT macro causes VSAM to skip ahead (never back) to the next record.

Example: Record RBAs When Loading

In this example, a PUT macro is used to record the RBAs of records as they are loaded into a key-sequenced data set. The RBAs are recorded in a table with 20-byte entries (4 bytes for RBA, 15 bytes for associated key, and 1 byte of padding so the next entry begins on a fullword boundary).

```
          LA      3,RBATABLE          Address of the beginning of the table.
          .
          .
          .
LOOP      L       2,source            Put length of record to be inserted into
                                      register 2.
          MODCB   RPL=LIST,           Indicate record length in request
                  RECLEN=(2)          parameter list.
          LTR     15,15
          BNZ     CHECK0
          PUT     RPL=LIST
          LTR     15,15
          BNZ     ERROR
          SHOWCB  AREA=(3),           Each SHOWCB puts a record's RBA
                  FIELDS=RBA,         into the table.
                  LENGTH=4,
                  RPL=LIST
          LTR     15,15
          BNZ     CHECK0
          MVC     4(15,3),            Put the record's key field in the table.
                  keyfield
          LA      3,20(3)             Point to the next entry.
          B       LOOP
ERROR     ...                         Request wasn't accepted or failed.
CHECK0    ...                         Modification or display failed.
          .
          .
          .
          DSECT                       Get enough virtual storage for as many
                                      table entries as there are records in the
                                      data set.
RBATABLE  DS      0F
RBA       DS      CL4
KEY       DS      CL15
          DS      CL1                 Padding to keep each RBA entry on a
                                      fullword boundary: SHOWCB's display
                                      area must be on a fullword boundary.
```

The need to process a key-sequenced data set by address should be unusual, but by recording the RBA of each record in a key-sequenced data set, you have search arguments for possible processing of the data set by addressed-direct retrieval and by addressed-sequential retrieval using the POINT macro. (You don't need to know RBAs to process a key-sequenced data set by simple addressed-sequential retrieval, since you go from the beginning without any skips.)

You can display the RBA of a record after you issue a GET or a POINT, as well as after you issue a PUT.

Example: Load a Relative Record Data Set (Skip-Sequential and Direct Processing)

In this example, a PUT macro is used to store twenty 100-byte records in slots 5, 10, 15,...,100 of the data set. MODCB is used to switch to direct processing, and a PUT is used to store records in slots 26 and 51 of the data set.

```
OUTACB   ACB     MACRF=(SKP,OUT,
                 DIR,KEY)
                 .
                 .
                 .
         GENCB   BLK=RPL,                 Generate a request parameter list at
                 ACB=OUTACB,              execution time.
                 AREA=WORK,
                 AREALEN=100,
                 ARG=RCDNO,
                 OPTCD=(KEY,SKP)
         LTR     15,15
         BNZ     GENFAIL
         LR      5,0                      Save length of RPL
         LR      6,1                      Save address of RPL
         LA      7,5                      Initialize increment value
         ST      7,RCDNO                  Initialize argument to slot 5
         LA      10,20                    Initialize loop counter
LOOP     ...                              Move new record into work
         PUT     RPL=(6)                  Store record
         LTR     15,15
         BNZ     PUTERR                   Request was not accepted or it failed.
         L       1,RCDNO
         AR      1,7
         ST      1,RCDNO                  Increment argument by 5
         BCT     10,LOOP
         MODCB   RPL=(6),                 Switch to direct processing to store
                 OPTCD=(DIR,KEY)          records in slots 51 and 26
         LTR     15,15
         BNZ     GENFAIL
         LA      7,51
         ST      7,RCDNO                  Initialize argument to slot 51
         ...                              Move new record into WORK
         PUT     RPL=(6)                  Store record in slot 51
         LTR     15,15
         BNZ     PUTERR                   Request was not accepted or it failed.
         LA      7,26
         ST      7,RCDNO                  Initialize argument to slot 26
         ...                              Move new record into WORK
         PUT     RPL=(6)                  Store record in slot 26
         LTR     15,15
         BNZ     PUTERR                   Request was not accepted or it failed.
         B       RETURN
```

```
GENFAIL  ...                          Generation or modification failed.
PUTERR   ...                          PUT request was not accepted or it
                                      failed.
RETURN   ...                          Terminate program
WORK     DS      CL100                100-byte work area that contains record
                                      to be stored by PUT macro
RCDNO    DS      CL4                  4-byte relative record number
```

Both skip-sequential and direct processing can be used to create a relative record data set. The ACB is opened for output. The four-byte search argument (RCDNO) indicates the slot number where the record is to be stored.

Example: Keyed-Sequential Insertion (Relative Record Data Set)

In this example, a PUT macro is used to insert twenty 100-byte records into empty slots of a previously loaded relative record data set. If the slot is empty when the PUT is issued, the record is stored and the slot number (returned in the argument field) is stored in a table. If the slot is not empty when the PUT is issued, a duplicate record error indication is returned. When a duplicate record is indicated, the PUT is reissued until the record is successfully stored in an empty slot in the data set.

```
OUTACB   ACB     MACRF=(KEY,SEQ,
                 OUT)
                 .
                 .
                 .
         GENCB   BLK=RPL,             Generate a request parameter list
                 ACB=OUTACB,
                 AREA=WORK,
                 AREALEN=100,
                 ARG=RCDNO,
                 OPTCD=(KEY,SEQ)
         LTR     15,15
         BNZ     GENERR
         LR      6,1                  Save the address of the RPL
         LA      4,RRNTABLE+80        Initialize address of end of table
         LA      3,RRNTABLE           Initialize index to relative record
                                      number table
WRITERCD ...                          Move record into work area.
                 .
                 .
                 .
         PUT     RPL=(6)
         LTR     15,15
         BZ      STRCDNO              Branch, if PUT is successful
         LA      10,8
         CLR     10,15                Test for logical error
         BNE     PUTERR
         TESTCB  RPL=(6),FDBK=8,      Test for duplicate record
                 ERET=TESTERR
         BE      WRITERCD             Branch, if duplicate record, and try to
                                      store record in next slot
         B       PUTERR
```

```
STRCDNO         ...
          MVC   0(4,3),RCDNO    Store relative record number in
                                RRNTABLE
          LA    3,4(3)          Increment to next table entry
          CLR   3,4
          BE    RETURN          If table full, return to caller
          B     WRITERCD        Write next record
GENERR    ...                   Error routine for GENCB macro
TESTERR   ...                   Error routine for TESTCB macro
PUTERR    ...                   Error routine for PUT macro
RETURN    ...                   Return to caller or terminate program
RCDNO     DS    CL4             4-byte relative record number
                                (argument) field
RRNTABLE  DS    20F             Relative record number table
WORK      DS    100             100-byte work area that contains record
                                to be stored by PUT macro
```

Each record is stored in the next available slot in the data set. When a record is successfully stored, its relative record number is recorded in a table.

Example: Skip-Sequential Insertion

In this example, one PUT macro is used to insert multiple fixed-length, 100-byte records. Records are to be moved asynchronously from a work area.

```
OUTPUT    ACB     MACRF=(KEY,SKP,
                  OUT)
          .
          .
          .
          GENCB   BLK=RPL,              Generate 5 request parameter lists at
                  COPIES=5,             execution.
                  ACB=OUTPUT,
                  AREALEN=100,
                  OPTCD=(KEY,SKP,
                  ASY,NUP,MVE),
                  RECLEN=100
          LTR     15,15
          BNZ     CHECK0
```

Calculate length of each list and use register notation with the MODCB macro to complete each list.

```
          MODCB   RPL=(2),
                  AREA=(3),
                  NXTRPL=(4)
          LTR     15,15
          BNZ     CHECK0
```

Increase the value in each register and repeat the MODCB until all five request parameter lists have been completed. The last time, register 4 must be set to 0.

```
          .
          .
          .
LOOP      ...                         Restore address of first list in register 2.
```

```
                                                  Build 5 records in WORK.
          PUT     RPL=( 2 )                       Register 2 points to the first request
                                                  parameter list in the chain. The five
                                                  records in WORK are stored with this
                                                  one PUT request.
          LTR     15,15
          BNZ     NOTACCEP
          .
          .
          .
          CHECK   RPL=( 2 )
          LTR     15,15
          BNZ     ERROR
          B       LOOP
CHECK0    ...                                     Generation or modification failed.
NOTACCEP  ...
ERROR     ...                                     Display the feedback field in each
                                                  request parameter list to find out which
                                                  one had an error.
WORK      DS      CL500                           Contains five 100-byte work areas.
```

You give no search argument for storage: VSAM knows the position of the key field in each record and extracts the key from it. Skip sequential insertion differs from keyed-direct insertion in the sequence in which records may be inserted (ascending nonconsecutive sequence versus random sequence) and in performance.

With skip-sequential insertion, if you insert two or more records into a control interval, VSAM doesn't write the contents of the buffer to direct-access storage until you have inserted all of the records. With direct insertion, VSAM writes the contents of the buffer after you have inserted each record.

Example: Keyed-Direct Insertion

In this example, a PUT macro is used to move fixed-length, 100-byte records from a work area.

```
OUTPUT    ACB     MACRF=(KEY,DIR,
                  OUT)
DIRECT    RPL     ACB=OUTPUT,
                  AREA=WORK,
                  AREALEN=100,
                  OPTCD=(KEY,DIR,
                  ASY,NUP,MVE),
                  RECLEN=100
          .
          .
          .
LOOP      PUT     RPL=DIRECT
          LTR     15,15
          BNZ     NOTACCEP
          .
          .
          .
          CHECK   RPL=DIRECT
          LTR     15,15
          BNZ     ERROR
          B       LOOP
NOTACCEP  ...                       Request wasn't accepted.
ERROR     ...                       Request failed.
          .
          .
          .
WORK      DS      CL100             Work area.
```

The macros are described, as follows:

- ACB specifies that the data set, OUTPUT, into which records are to be inserted, is opened for keyed-direct, ouput processing.
- RPL specifies that the record to be inserted into the OUTPUT data set resides in a 100-byte area, WORK.

VSAM extracts the key from the key field of each record found at WORK. Using keyed-direct access is very similar to using skip sequential access.

Example: Addressed-Sequential Addition

In this example, a PUT macro is used to add variable-length records to a data set. The data set is assumed to be an entry-sequenced data set because records cannot be inserted into or added to a key-sequenced data set with addressed access.

```
BLOCK    ACB     MACRF=(ADR,SEQ,
                 OUT)
LIST     RPL     ACB=BLOCK,
                 AREA=NEWRCD,
                 AREALEN=100,
                 OPTCD=(ADR,SEQ,
                 SYN,MVE)
                 .
                 .
                 .
LOOP     ...                          Build the record.
         L       3,source             Put the length of the record into
                                      register 3.
         MODCB   RPL=LIST,            Indicate length of new record.
                 RECLEN=(3)
         LTR     15,15
         BNZ     CHECK0
         PUT     RPL=LIST
         LTR     15,15
         BNZ     ERROR
         B       LOOP
CHECK0   ...                          Modification failed.
ERROR    ...                          Request wasn't accepted or failed.
                 .
                 .
                 .
NEWRCD   DS      CL100                Build record in this work area.
```

Each record is stored in the next position after the last record in the data set. You do not have to specify an RBA or do any explicit positioning (with the POINT macro). Addressed addition of records is always identical to loading a data set: when additional space is required, VSAM extends the data set.

The only difference between addressed-sequential and addressed-direct addition is when the buffers are written to external storage. The buffer is written to external storage only when it is full for sequential addition; it is written after each record for direct addition. You cannot use direct storage to load records into a data set for the first time; you must use sequential storage.

Example: Keyed-Sequential Update

In this example, GET and PUT macros are used to retrieve and update fixed-length, 50-byte records. Records are updated synchronously in a work area. This example requires the use of a work area because you cannot update a record in the I/O buffer.

```
UPDATA   ACB    MACRF=(KEY,SEQ,
                OUT)
LIST     RPL    ACB=UPDATA,          UPD indicates the record may be
                AREA=WORK,           stored back (or deleted).
                AREALEN=50,
                OPTCD=(KEY,SEQ,
                SYN,UPD,MVE)
         .
         .
         .
LOOP     GET    RPL=LIST
         LTR    15,15
         BNZ    ERROR
Decide whether to update the record.
         B      LOOP                 Don't update it; retrieve another.
Do update the record.
         PUT    RPL=LIST             Store the record back.
         LTR    15,15
         BNZ    ERROR
         B      LOOP
ERROR    ...                         Request wasn't accepted or failed.
         .
         .
         .
WORK     DS     CL50                 VSAM puts the retrieved record here.
```

A GET for update (OPTCD=UPD) must precede a PUT for update. Besides retrieving the record to be updated, GET positions VSAM at the record retrieved, in anticipation of the succeeding update (or deletion). It is not necessary for you to store back (or delete) the record that you retrieved for update. VSAM's position at the record previously retrieved allows you to issue another GET to retrieve the following record. You cannot then, however, store back the previous record: the position for update has been forgotten because of the following GET.

Example: Keyed-Direct Update

In this example, GET and PUT macros are used to retrieve and update records. The MODCB macro is used to modify record length (RECLEN) in the request parameter list when an update causes the record length to change. The maximum record length is 120 bytes. The search argument is a full key (five bytes), compared equal.

```
INPUT    ACB     MACRF=(KEY,DIR,
                 OUT)
UPDTE    RPL     ACB=INPUT,          UPD indicates the record may be
                 AREA=IN,            stored back (or deleted).
                 AREALEN=120,
                 OPTCD=(KEY,DIR,
                 SYN,UPD,KEQ,
                 FKS,MVE),
                 ARG=KEYAREA,
                 KEYLEN=5
         .
         .
         .
```

Process input and get search argument into KEYAREA; proceed to retrieve a record.

```
LOOP     GET     RPL=UPDTE
         LTR     15,15
         BNZ     ERROR
         SHOWCB  RPL=UPDTE,          Display the length of the record.
                 AREA=RLNGTH,
                 FIELDS=RECLEN,
                 LENGTH=4
         LTR     15,15
         BNZ     CHECK0
```

Update the record. Does the update change the record's length?

```
         B       STORE               No, length not changed.
         L       5,length            Yes, load new length into register
5.       MODCB   RPL=UPDTE,          Modify length indication in the request
                 RECLEN=(5)          parameter
list.    LTR     15,15
         BNZ     CHECK0
STORE    PUT     RPL=UPDTE
         LTR     15,15
         BNZ     ERROR
         B       LOOP
ERROR    ...                         Request wasn't accepted or failed.
CHECK0   ...                         Display or modification failed.
         .
         .
         .
IN       DS      CL120               Work area for retrieving, updating, and
                                     storing a record.
KEYAREA  DS      CL5                 Search argument for retrieving a
                                     record.
RLNGTH   DS      F                   Area for displaying the length of a
                                     retrieved record.
```

You cannot update records in the I/O buffer. A direct GET for update positions VSAM at the record retrieved, in anticipation of storing back (or deleting) the record. This positioning also allows you to switch to sequential access to retrieve the next record.

You don't have to store back a record that you retrieve for update, but if you don't store it back before another retrieval, it's too late to do so.

Example: Addressed-Sequential Update

In this example, GET and PUT macros are used to retrieve and update records in an entry-sequenced data set. The records are variable in length, maximum 200 bytes. The lengths of the records are not changed by update (the length of a record can never be changed by addressed access).

```
ENTRY    ACB     MACRF=(ADR,SEQ,
                 OUT)
ADRUPD   RPL     ACB=ENTRY,          UPD indicates update (or deletion).
                 AREA=WORK,
                 AREALEN=200,
                 OPTCD=(ADR,SEQ,
                 SYN,UPD,MVE)
                 .
                 .
                 .
LOOP     GET     RPL=ADRUPD
         LTR     15,15
         BNZ     ERROR
         SHOWCB  RPL=ADRUPD,         Find out how long the record is.
                 AREA=RLNGTH,
                 FIELDS=RECLEN,
                 LENGTH=4
         LTR     15,15
         BNZ     CHECK0
                 .
                 .
                 .
         PUT     RPL=ADRUPD
         LTR     15,15
         BNZ     ERROR
         B       LOOP
ERROR    ...                         Request wasn't accepted or failed.
CHECK0   ...                         Display failed.
                 .
                 .
                 .
WORK     DS      CL200               Record-processing work area.
RLNGTH   DS      F                   Display area for length of records.
```

If you have inactive records in your entry-sequenced data set, you may reuse the space they occupy by retrieving the records for update and restoring a new record in their place.

With a key-sequenced data set, it is not possible to change the length of records by addressed update because the index is not used and VSAM could not split a control interval if required because of changing record length.

Addressed-direct update varies from sequential update in the specification of an RBA for a search argument.

Example: Mark Records Inactive

In this example, GET and PUT macros are used to retrieve a record from an entry-sequenced data set and to mark it as inactive. The record is marked as inactive by putting a hexadecimal 'FF' in first byte of a record. The inactive record will not be sequentially retrieved except for update.

```
ENTRYSEQ ACB     MACRF=(ADR,DIR,
                 OUT)
LIST     RPL     ACB=ENTRYSEQ,          UPD indicates update: storing the
                 AREA=RECORD,           record back marked inactive.
                 AREALEN=100,
                 OPTCD=(ADR,DIR,
                 SYN,UPD,MVE),
                 ARG=RBAAREA
         .
         .
         .
LOOP     GET     RPL=LIST
         LTR     15,15
         BNZ     ERROR
Decide whether you still want the data in the
record.
         B       LOOP                   Yes, retrieve the next record.
         MVI     RECORD,X'FF'           No, flag the record inactive.
         PUT     RPL=LIST               Storing the record with an inactive
                                        indicator is equivalent to deletion for
                                        an entry-sequenced data set.
         LTR     15,15
         BNZ     ERROR
         B       LOOP
ERROR    ...                            Request wasn't accepted or failed.
RECORD   DS      CL100                  Work area for marking records.
RBAAREA  DS      F                      Search argument for retrieving the
                                        record.
```

Records of an entry-sequenced data set can't be deleted. If a record loses its usefulness for your application, your program can mark it inactive by placing a unique flag in some conventional part of the record so that when your programs retrieve the record thereafter they can recognize and bypass it. You can use the space occupied by an inactive record by retrieving it for update and storing a new record in its place.

RPL Macro (Generate a Request Parameter List)

After you have connected your program to the data set, you can issue requests for access to it. A *request parameter list* defines a request. Each request macro (GET, PUT, ERASE, POINT, CHECK, and ENDREQ) gives the address of the request parameter list that defines it. See the chapter "Request Macros" for information on these macros.

The RPL macro can be used to generate a request parameter list when your program is assembled. It identifies the data set to which the request is directed by naming the ACB macro that defines the data set. The operands of the RPL macro are optional in some cases, but required in others. It is not necessary to omit operands that are not required for a request; they are ignored. Thus, for example, if you switch from direct to sequential retrieval with a request parameter list, you don't have to zero out the address of the field containing the search argument (ARG=address). You can use the MODCB macro to modify some of the parameters to change the type of processing. For example, you can change from direct to sequential processing or from update to nonupdate processing.

For concurrent requests that require VSAM to keep track of more than one position in a data set, you can use up to 255 RPL macros to specify requests that your processing program or its subtasks can issue concurrently. The requests can be sequential or direct or both, and they can be for records in the same or different parts of the data set.

You need specify only the RPL parameters appropriate to a given request:

Address of the exit request parameter list in the chain: You can chain request parameter lists together to define a series of actions for a single GET or PUT. For example, each parameter list in the chain could contain a unique search argument and point to a unique work area. A single GET macro would retrieve a record for each request parameter list in the chain. A chain of request parameter lists is processed serially as a single request (chaining request parameter lists is not the same as processing in parallel concurrent requests, requiring VSAM to keep track of many positions in a data set).

When using chained RPLs, the RPLs in the chain are processed serially. If an error occurs anywhere in the chain, the RPLs following the one in error are made available without being processed and are posted complete with a feedback code of zero.

Processing options for a request: A request gains access to a data record or a control interval. Access can be gained by address (RBA), key, or relative record number. Address access can be sequential or direct; keyed access can be sequential, skip sequential, or direct. Access can be forward (next sequential record) or backward (previous sequential record). Access can be for updating or not undating. A nonupdate direct request to retrieve a record can optionally cause VSAM to position to the following record for subsequent sequential access. Other characteristics that can be specified are:

- A request (including a request defined by a chain of request parameter lists is either synchronous, so that VSAM does not give control back to your program until the request completes, or asynchronous, so that your program can continue to process or issue other requests while the request is

active. Your program must later use the CHECK macro to suspend its processing until the asynchronous request completes.

- For a keyed request, you specify either a generic key or a full key to which the key field of the record is to be compared. A generic-key search argument is matched for either an equal or a greater-than-or-equal comparison to a user-defined subset of the key field; a full-key search argument is matched for either an equal or a greater-than-or-equal comparison to the key field.
- For retrieval, a request is either for a data record to be placed in a work area in the user's program or for the address of the record within VSAM's buffer to be passed to the user's program. For all other requests (requests that involve updating or inserting), the work area in the user's program contains the data record.
- For a request to directly access a control interval, you specify the RBA of the control interval. With control-interval access, you are responsible for maintaining the control information in the control interval. If VSAM's buffers are used, VSAM allows control-interval and stored record operation to be intermixed. If your program provides its own buffers, intermixing is not allowed.

Address and size of the work area to contain a data record: You must provide a work area large enough to contain a data record or the address of the data record within one of VSAM's buffers.

Length of the data record being processed: For storage, the user's program indicates the length to VSAM; for retrieval, VSAM indicates the length of the user's program.

Length of the key: This parameter is required only for processing by generic key. For full key access, the key length is available to VSAM.

Address of the area containing the search argument: The search argument is either a key value or an RBA (a relative record number is considered a key value).

Address and length of an area for error messages from VSAM: Your routine for analyzing physical errors (your SYNAD routine) receives messages in this area. The minimum length required for a message is 128 (X'80') bytes.

Values for RPL-macro operands can be specified as absolute numeric expressions, character strings, codes, and expressions that generate valid relocatable A-type address constants.

The format of the RPL macro is:

[*label*]	**RPL**	**[ACB=***address***]** **[,AM=VSAM]** **[,AREA=***address* **]** **[,AREALEN=***number* **]** **[,ARG=***address* **]** **[,ECB=***address* **]** **[,KEYLEN=***number* **]** **[,MSGAREA=***address***.]** **[,MSGLEN=***number* **]** **[,NXTRPL=***address* **]** **[,OPTCD=([ADR | CNV | KEY]** **[,DIR | SEQ | SKP]** **[,ARD | LRD]** **[,FWD | BWD]** **[,ASY | SYN]** **[,NSP | NUP | UPD]** **[,KEQ | KGE]** **[,FKS | GEN]** **[,LOC | MVE])]** **[,NWAITX) | WAITX]** **[,RECLEN=***number* **]** **[,TRANSID=***number* **]**

where:

label
is one to eight characters that provides a symbolic address for the request parameter list that is generated. You can use it in the request macros to give the address of the list. You can use it in the NXTRPL operand of the RPL macro, when you are chaining request parameter lists, to indicate the next list.

ACB=*address*
specifies the address of the access-method control block that identifies the data set to which access will be requested. If you used the ACB macro to generate the control block, you may specify the label of that macro for the address. If the ACB operand is not coded, you must specify the address before issuing the request.

AM=VSAM
specifies that the access method using the control block is VSAM.

AREA=*address*
specifies the address of a work area to and from which VSAM moves a data record if you request it to do so (with the RPL operand OPTCD=MVE). If your request is to process records in the I/O buffer (OPTCD=LOC), VSAM puts into this work area the address of a data record within the I/O buffer.

AREALEN=*number*
specifies the length, in bytes, of the work area whose address is specified by the AREA operand. Its minimum for OPTCD=MVE is the size of a data record (of the largest data record, for a data set with records of variable length). For OPTCD=LOC the area should be 4 bytes to contain the address of a data record within the I/O buffer.

ARG=*address*
specifies the address of a field that contains the search argument for direct retrieval, skip-sequential retrieval, and positioning. For a relative record data set, the ARG field must be 4 bytes long. For direct or skip-sequential processing, this field contains your search argument, a relative record number. For sequential processing (OPTCD=(KEY,SEQ)), the 4 bytes are required for VSAM to return the feedback RRN. For keyed access (OPTCD=KEY), the search argument is a full or generic key or relative record number; for addressed access (OPTCD=ADR), it is an RBA. If you specify a generic key (OPTCD=GEN), you must also specify in the KEYLEN operand how many of the bytes of the full key you are using for the generic key. ARG is also used with WRTBFR and MRKBFR. Its usage with these macros is described in *OS/VS Virtual Storage Access Method (VSAM) Options for Advanced Applications.*

ECB=*address*
specifies the address of an event control block (ECB) that you may supply. VSAM indicates in the ECB whether a request is complete or not (using standard OS/VS completion codes, which are described in *OS/VS1 System Data Areas* and *OS/VS2 Data Areas*). This operand is always optional.

KEYLEN=*number*
specifies the length, in bytes, of the generic key (OPTCD=GEN) you are using for a search argument (given in the field addressed by the ARG operand). This operand is specified as a number from 1 through 255; it is required when the search argument is a generic key. For full-key searches, VSAM knows the key length, which is taken from the catalog definition of the data set when you open the data set.

MSGAREA=*address*
specifies the address of an area that you may optionally supply for VSAM to send you a message in case of a physical error. The format of a physical-error message is given under "Physical Errors" in the chapter "Request Macros."

MSGLEN=*number*
specifies the size, in bytes, of the message area indicated in the MSGAREA operand. If MSGAREA is specified, MSGLEN is required. The size of a message is 128 bytes; if you provide less than 128 bytes, no message is returned to your program.

NXTRPL=*address*
specifies the address of the next request parameter list in a chain. Omit this operand from the macro that generates the last list in the chain. When you issue a request that is defined by a chain of request parameter lists, indicate in the request macro the address of the first parameter list in the chain.

OPTCD=([ADR | CNV | KEY]
[,DIR | SEQ | SKP]
[,ARD | LRD]
[,FWD | BWD]
[,ASY | SYN]
[,NSP | NUP | UPD]
[,KEQ | KGE]
[,FKS | GEN]
| [,NWAITX | WAITX)]
[,LOC | MVE])

specifies the options that govern the request defined by the request parameter list. Each group of options has a default; options are shown in Figure 10 with defaults underlined. Only one option from each group can be specified. Some requests do not require an option from all of the groups to be specified. The groups that aren't required are ignored; thus, you can use the same request parameter list for a combination of requests (GET, PUT, POINT, for example) without zeroing out the inapplicable options each time you go from one request to another.

RECLEN=*number*

specifies the length, in bytes, of a data record being stored. This parameter is required for a PUT request.

For GET requests, VSAM puts the length of the record retrieved in this field in the request parameter list. It will be there if you update and store the record.

TRANSID=*number*

specifies a number that relates modified buffers in a buffer pool. Used in shared resource applications and described in *OS/VS Virtual Storage Access Method (VSAM) Options for Advanced Applications.*

You can use the ECB to determine that an asynchronous request is complete before issuing a CHECK macro. (If you issue a CHECK before a request is complete, you give up control and must wait for completion.) You can also test for completion with the TESTCB I/O=COMPLETE operand. TESTCB is described later in this chapter.

Each request parameter list in a chain should have the same OPTCD options. Having different options may cause logical errors. You can't chain request parameter lists for updating or deleting records—only for retrieving records or storing new records. You can't process records in the I/O buffer with chained request parameter lists. (OPTCD=UPD and LOC are invalid for a chained request parameter list.)

With chained request parameter lists, a POINT, a sequential or skip-sequential GET, or a direct GET with positioning requested (OPTCD=NSP) causes VSAM to position itself at the record following the record identified by the last request parameter list in the chain.

Option	Meaning
ADR	Addressed access to a key-sequenced or an entry-sequenced data set: RBAs are used as search arguments and sequential access is done by entry sequence
CNV	Control-interval access (this type of access is described in *OS/VS Virtual Storage Access Method (VSAM) Options for Advanced Applications)*
KEY	Keyed access to a key-sequenced or relative record data set: keys or relative record numbers are used as search arguments and sequential access is done by key or relative record number sequence
DIR	Direct access to a key-sequenced, entry-sequenced, or relative record data set
SEQ	Sequential access to a key-sequenced, entry-sequenced, or relative record data set
SKP	Skip sequential access to a key-sequenced or a relative record data set: used with keyed access only
ARD	User's argument determines record to be located, retrieved, or stored
LRD	Last record in the data set is to be located (POINT) or retrieved (GET direct); requires OPTCD=BWD.
FWD	Processing to proceed in forward direction
BWD	Processing to proceed in backward direction; for keyed (KEY) or addressed (ADR) sequential (SEQ) or direct (DIR) requests; valid for POINT, GET, PUT, and ERASE operations; establish positioning by a POINT with OPTCD=BWD or by a GET direct with OPTCD=(NSP, BWD). When OPTCD=BWD is specified, options KGE and GEN are ignored; options KEQ and FKS are assumed.
ASY	Asynchronous access; VSAM returns to the processing program after scheduling a request so the program can do other processing while the request is being carried out
SYN	Synchronous access; VSAM returns to the processing program after completing a request
NSP	With OPTCD=DIR only, VSAM is to remember its position (for subsequent sequential access); that is, the position is not to be forgotten unless an ENDREQ macro is issued
NUP	A data record that is being retrieved will not be updated or deleted; a record that is being stored is a new record; VSAM doesn't remember its position for direct requests into a work area
UPD	A data record that is being retrieved may be updated or deleted; a record that is being stored or deleted was previously retrieved with OPTCD=UPD; VSAM remembers its position for sequential and direct GET requests
KEQ	For GET with OPTCD=(KEY,DIR) or (KEY,SKP) and for POINT with OPTCD=KEY, the key (full or generic) that you provide for a search argument must equal the key or relative record number of a record. For an RRDS, KEQ is assumed except for POINT.
KGE	For the same cases as KEQ, if the key (full or generic) that you provide for a search argument doesn't equal that of a record, the request applies to the record that has the next higher key For a relative record data set and POINT, KGE positions to the specified relative record number whether the slot is empty or not. If the relative record number is greater than the highest existing record, an EOD is returned. A subsequent PUT will insert the record at this position.

Figure 10 (Part 1 of 2) OPTCD Options

Option	Meaning
FKS	A full key is provided as a search argument
GEN	A generic key is provided as a search argument; give the length in the KEYLEN operand
NWAITX	Never take the user's UPAD exit
WAITX	If OPTCD=SYN and the ACB's MACRF=LSR GSR and UPDAD exit routine is specified, VSAM takes the UPAD exit at points when VSAM would normally issue a WAIT.
LOC	For retrieval, VSAM leaves the data record in the I/O buffer for processing; not valid for PUT or ERASE; valid for GET with OPTCD=UPD, but to update the record, you must build a new version of the record in a work area and modify the request parameter list OPTCD from LOC to MVE before issuing a PUT
MVE	For retrieval, VSAM moves the data record to a work area for processing, and for storage, VSAM moves it from the work area to the I/O buffer

Figure 10 (Part 2 of 2) OPTCD Options

Example: RPL Macro

In this example, an RPL macro is used to generate a request parameter list named PARMLIST.

```
ACCESS   ACB    MACRF=(SKP,OUT),
                DDNAME=PAYROLL

PARMLIST RPL    ACB=ACCESS,
                AM=VSAM,
                AREA=WORK,
                AREALEN=125,
                ARG=SEARCH,
                MSGAREA=MESSAGE,
                MSGLEN=128,
                OPTCD=(SKP,UPD)     Most OPTCD defaults are appropriate
                                    to assumptions.
WORK     DS     CL125
SEARCH   DS     CL8
MESSAGE  DS     CL128
```

The ACB macro, named ACCESS, specifies skip-sequential retrieval for update. Further details may be provided on a DD statement named PAYROLL.

The RPL macro's operands are:

- ACB, which associates the request parameter list with the access-method control block generated by ACCESS.
- AREA and AREALEN, which specify a work area, WORK, that is 125 bytes long.
- ARG, which specifies that the search argument is defined at SEARCH. The search argument is eight bytes long.
- MSGAREA and MSGLEN, which specify a message area, MESSAGE, that is 128 bytes long. The message area is provided for physical-error messages.
- OPTCD, which specifies skip-sequential processing and specifies that a retrieved record may be updated or deleted.

Because KEYLEN is not coded, a full-key search is assumed.

SHOWCB Macro (Display an Access-Method Control Block)

The SHOWCB macro can be used to cause VSAM to move the contents of various fields in an access-method control block into your work area. You might want to learn the reason for an error or to collect information about a data set in order to alter your program logic or print a message or report as a result of the examination.

The operands of the SHOWCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. "Appendix C: Operand Notation for GENCB, MODCB, SHOWCB, and TESTCB" gives all the ways of coding each operand for the macros that work at execution.

See "Return Codes from the GENCB, MODCB, SHOWCB, and TESTCB Macros" for information on the return codes used to indicate whether the SHOWCB request was successful.

The format of the SHOWCB macro used to display fields in an access-method control block is:

[*label*]	**SHOWCB**	**ACB=***address* **,AREA=***address* **,LENGTH=***number* **[,OBJECT=DATA | INDEX]** **,FIELDS=([ACBLEN][,AVSPAC][,BFRFND]** **[,BSTRNO][,BUFND][,BUFNI]** **[,BUFNO][,BUFRDS][,BUFSP]** **[,CINV][,DDNAME][,ENDRBA]** **[,ERROR][,EXLST][,FS]** **[,HALCRBA][,KEYLEN][,LRECL]** **[,MAREA][,MLEN][,NCIS][]** **[,NDELR][,NEXCP][,NEXT][,NINS]** **[,NIXL,NLOGR][NRETR][,NSSS]** **[,NUIW,NUPDR][,PASSWD][,RKP]** **[,STMST][,STRMAX][,STRNO]** **[,UIW])**

where:

label
is one to eight characters that provides a symbolic address for the SHOWCB macro.

ACB=*address*
specifies the address of the access-method control block whose fields are to be displayed. If you used the ACB macro with a label, you can specify the label here. The ACB operand is optional when you wish to display the length of an access-method control block (FIELDS=ACBLEN). (All access-method control blocks have the same length, so you need not specify the address of a particular one.)

AREA=*address*

specifies the address of a work area that you are supplying for VSAM to display the contents of the fields you specify in the FIELDS operand. The contents of the fields are displayed in the order you specify them. The area must begin on a fullword boundary.

LENGTH=*number*

specifies the length, in bytes, of the work area that you are providing for VSAM to display the indicated fields in. See the FIELDS operand for the fields that can be displayed and for the length of each field. If the area is not large enough for all of the fields, VSAM doesn't display any of their contents and returns an error code indicating it (see "Return Codes from the GENCB, MODCB, SHOWCB, and TESTCB Macros" earlier in this chapter).

OBJECT=DATA | INDEX

specifies whether fields are to be displayed for the data or for the index.

FIELDS= ([ACBLEN,][AVSPAC,][BFRFND,][,BSTRNO][,BUFND]
[,BUFNI][,BUFNO][,BUFRDS][,BUFSP]
[,CINV][,DDNAME][,ENDRBA]
| [,ERROR][,EXLST][,FS][,HALCRBA]
[,KEYLEN][,LRECL][,MAREA][,MLEN][,NCIS]
[,NDELR][,NEXCP][,NEXT]
[,NINSR][,NIXL][,NLOGR]
[,NRETR][,NSSS][,NUIW][,NUPDR]
[,PASSWD][,RKP][,STMST][,STRMAX]
[,STRNO][,UIW])

specifies the fields whose contents are to be displayed. Some of the fields can be displayed at any time; others only after a data set is opened. The ones that can be displayed only after a data set is opened can, in the case of a key-sequenced data set that has been opened for keyed access, pertain either to the data or to the index. See the OBJECT operand. Figure 11 explains the keywords you can code in the FIELDS operand for an access-method control block.

Key-word	Full-words	Description of the Field
The following fields can be displayed at any time		
ACBLEN	1	Length of an access-method control block (displaying the length of an access-method control block gives your program independence from changes in the length that may occur from release to release of VSAM)
BSTRNO	1	Number of strings initially allocated for access to the base cluster by a path
BUFND	1	Number of I/O buffers to be used for data, as specified in the ACB (or GENCB)
BUFNI	1	Number of I/O buffers to be used for index entries, as specified in the ACB (or GENCB)
BUFSP	1	Amount of space specified in the ACB (or GENCB) for I/O buffers
DDNAME	2	Name of the DD statement that identifies the data set
ERROR	1	The code returned by VSAM after the opening or closing of the data set (see "OPEN Macro" and "CLOSE Macro")
EXLST	1	Address of the exit list, if any; 0 if none
MAREA	1	Address of the message area, if any; 0 if none
MLEN	1	Length of the message area, if any; 0 if none
PASSWD	1	Address of the field containing the password; the first byte of the field contains the length of the password (in binary)
STRMAX	1	Maximum number of strings concurrently active
STRNO	1	Number of requests for which VSAM is prepared to remember its position in the data set
The following fields can be displayed only after the data set is opened		
AVSPAC	1	Amount of available space in the data component or index component, in bytes
BFRFND	1	Number of successful looks-aside[1]
BUFNO	1	Number of I/O buffers actually in use for the data component or index component
BUFRDS	1	Number of buffer reads[1]
CINV	1	Control-interval size for the data component or index component
ENDRBA	1	Ending RBA of the space used by the data component or index component; not the RBA of any record in the data set, but of the last used byte in the data set
FS	1	Number of free control intervals per control area in the data component (0 for OBJECT=INDEX)
HALCRBA	1	High-allocated RBA; the relative byte address of the end of the data component (OBJECT=DATA) or the index component (OBJECT=INDEX)
KEYLEN	1	Length of the key of reference of the key field of data records in the data component (whether OBJECT=DATA or INDEX)
LRECL	1	Length of data records in the data component (maximum length for variable-length data records) or of index records in the index component (control-interval length minus 7)

Figure 11 (Part 1 of 2). FIELDS Operand Keywords for an Access-Method Control Block

Key-word	Full-words	Description of the Field
NCIS	1	Number of control intervals that have been split in the data component (0 for OBJECT=INDEX)
NDELR	1	Number of records that have been deleted from the data component (0 for OBJECT=INDEX)
NEXCP	1	Number of EXCP macros that VSAM has issued for access to the data component or index component since it was opened
NEXT	1	Number of extents now allocated to the data component or index component (the maximum that can be allocated is 123)
NINSR	1	Number of records that have been inserted into (or added to) the data component (0 for OBJECT=INDEX)
NIXL	1	Number of levels in the index of the data component (0 for OBJECT=INDEX)
NLOGR	1	Number of data records in the data component (0 for OBJECT=INDEX)
NRETR	1	Number of records that have ever been retrieved from the data component (0 for OBJECT=INDEX)
NSSS	1	Number of control areas that have been split in the data component (0 for OBJECT=INDEX)
NUIW	1	Number of writes not initiated by user[1]
NUPDR	1	Number of records in the data component that have ever been updated (0 for OBJECT=INDEX)
RKP	1	Displacement of the key of reference of the key field from the beginning of a data record (whether OBJECT=DATA or INDEX)
STMST	2	System time stamp, which gives the time and day of the last time the data component or index component was closed, with bit 51 (counting from 0 at the left) equivalent to one microsecond and bits 52 through 63 unused
UIW	1	Number of user-initiated writes[1]

[1] Described in *OS/VS Virtual Storage Access Method (VSAM) Options for Advanced Applications*

Figure 11 (Part 2 of 2). FIELDS Operand Keywords for an Access-Method Control Block

Example: SHOWCB Macro (Display an Access-Method Control Block)

In this example, a SHOWCB macro is used to display fields in an access-method control block. The fields displayed (KEYLEN, LRECL, and RKP) permit the program to modify variables to process any one of a number of data sets that have different-sized key fields and records and different placements of key field in a record.

```
         SHOWCB ACB=CONTROL,
                AREA=DISPLAY,
                FIELDS=(KEYLEN,
                LRECL,RKP),
                LENGTH=12
DISPLAY  DS     0F                  Align on fullword boundary.
KEYLEN   DS     F
LRECL    DS     F
RKP      DS     F
```

The SHOWCB macro's operands are:

- ACB, which specifies the address of the access-method control block to be displayed.
- AREA, which specifies that the area to be used to display access-method control block fields is to begin on a fullword boundary.
- FIELDS, which specifies that the KEYLEN, LRECL, and RKP fields are to be displayed.
- LENGTH, which specifies that the length of the area to be used for the display is 12 bytes, enough to accommodate the specified fields.

This display enables the program to set up its variables for the particular data set it has opened.

Example: SHOWCB Macro (Display an Exit List Address)

In this example, a SHOWCB macro is used to get the address of an exit list by displaying the address in an access-method control block that uses the exit list.

```
SHOWCB ACB=address,
       AREA=address,
       FIELDS=EXLST,
       LENGTH=4
```

The SHOWCB macro's operands are:

- ACB, which specifies the address of an access-method control block from which the address of an exit list is to be displayed.
- AREA and LENGTH, which specify an area and length, four bytes, to be used to display the address of the exit list.
- FIELDS, which specifies that the EXLST field in an access-method control block is to be displayed.

SHOWCB Macro (Display an Exit List)

The SHOWCB macro can be used to display fields in an exit list.

The operands of the SHOWCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. "Appendix C: Operand Notation for GENCB, MODCB, SHOWCB, and TESTCB" gives all the ways of coding each operand for the macros that work at execution.

See "Return Codes from the GENCB, MODCB, SHOWCB, and TESTCB Macros" for information on the return codes used to indicate whether the SHOWCB request was successful.

The format of the SHOWCB macro used to display fields in an exit list is:

[*label*]	**SHOWCB**	**EXLST=***address*, **AREA=***address* **,LENGTH=***number* **,FIELDS=([EODAD][,EXLLEN][,JRNAD]** **[,LERAD][,SYNAD])**

where:

label
is one to eight characters that provides a symbolic address for the SHOWCB macro.

EXLST=*address*
specifies the address of the exit list whose fields are to be displayed. If you used the EXLST macro with a label, you can specify the label here. The EXLST operand is optional only when you wish to display the length that an exit list can have (see FIELDS=EXLLEN below). The SHOWCB macro does not support the UPAD user exit.

AREA=*address*
specifies the address of a work area that you are supplying for VSAM to display the contents of the fields you specify in the FIELDS operand. The contents of the fields are displayed in the order you specify them. The area must begin on a fullword boundary.

LENGTH=*number*
specifies the length, in bytes, of the work area that you are providing for VSAM to display the indicated fields in. Each exit-list field requires a fullword. If the area is not large enough for all of the fields, VSAM doesn't display any of their contents and returns an error code indicating it (see "Return Codes from the GENCB, MODCB, SHOWCB, and TESTCB Macros" earlier in this chapter).

FIELDS=([EODAD][,EXLLEN][,JRNAD]
[,LERAD][,SYNAD])
specifies the values to be displayed, as follows:

EODAD
specifies that the address of the end-of-data-set routine is to be displayed.

EXLLEN
specifies that the length of the exit list indicated in the EXLST operand or if EXLST is omitted, the maximum length an exit length can have, is to be displayed.

JRNAD
specifies that the address of the journaling routine is to be displayed.

LERAD
specifies that the address of the logical-error analysis routine is to be displayed.

SYNAD
specifies that the address of the physical-error analysis routine is to be displayed.

You can use SHOWCB to display the address of an exit routine only if the exit routine is indicated in the exit list. If it isn't, the SHOWCB request will fail. Use TESTCB to test whether an entry for a given exit type is present in the exit list and to find out whether the exit is active and whether the routine is to be loaded.

Example: SHOWCB Macro (Display the Length of an Exit List)

In this example, a SHOWCB macro is used to display the maximum length of an exit list. The maximum length of an exit list is subsequently used in a GENCB macro to get virtual storage for an exit list.

```
         SHOWCB AREA=LENGTH,
                FIELDS=EXLLEN,
                LENGTH=4
         L      0,LENGTH          Amount of storage for GETMAIN.
         GETMAIN R,LV=(0)
         LR     2,1               Address of storage for GENCB.
         GENCB  BLK=EXLST,        Indirect notation for length of work
                LENGTH=(*,        area.
                LENGTH),
                WAREA=(2)
                .
                .
                .
LENGTH   DS     F                 Contains the length of GENCB's work
                                  area.
```

The SHOWCB macro's operands are:

- AREA and LENGTH, which specify the area, which begins on a fullword boundary, and its length, four bytes, that is to be used for the display.
- FIELDS, which specifies that the maximum length of an exit list is to be displayed. Because only EXLLEN is specified, the EXLST operand is omitted.

The GENCB macro specifies a work area in which an exit list is to be generated. The length of the work area is located at LENGTH, where the maximum length of an exit list was put as a result of the SHOWCB macro.

SHOWCB Macro (Display a Request Parameter List)

The SHOWCB macro can be used to display fields in a request parameter list.

The operands of the SHOWCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. "Appendix C: Operand Notation for GENCB, MODCB, SHOWCB, and TESTCB" gives all the ways of coding each operand for the macros that work at execution. See "Return Codes from the GENCB, MODCB, SHOWCB, and TESTCB Macros" for information on the return codes used to indicate whether the SHOWCB request was successful.

The format of the SHOWCB macro used to display fields in a request parameter list is:

[*label*]	**SHOWCB**	**RPL=***address* **,AREA=***address* **,LENGTH=***number* **,FIELDS=([ACB][,AIXPC][,AREA][,AREALEN]** **[,ARG][,ECB][,FDBK][,FTNCD]** **[,KEYLEN][,MSGAREA]** **[,MSGLEN]** **[,NXTRPL][,RBA]** **[,RECLEN]** **[,RPLLEN]** **[,TRANSID])**

where:

label
is one to eight characters that provides a symbolic address for the SHOWCB macro.

AREA=*address*
specifies the address of a work area that you are supplying for VSAM to display the contents of the fields you specify in the FIELDS operand. The contents of the fields are displayed in the order you specify them. The area must begin on a fullword boundary.

LENGTH=*number*
specifies the length, in bytes, of the work area that you are providing for VSAM to display the indicated fields in. Each request parameter list field requires a fullword. If the area is not large enough for all of the fields, VSAM doesn't display any of their contents and returns an error code indicating it (see "Return Codes from the GENCB, MODCB, SHOWCB, and TESTCB Macros" earlier in this chapter).

RPL=*address*
specifies the address of the request parameter list whose fields are to be displayed. If you used the RPL macro with a label, you can specify the label here. The RPL operand is optional when you wish to display the length of a request parameter list (FIELDS=RPLLEN). (All VSAM request parameter lists have the same length, so you need not specify the address of a particular one.)

FIELDS=([ACB][,AIXPC] [,AREA] [,AREALEN] [,ARG]
[,ECB][,FDBK][,FTNCD][,KEYLEN]
[,MSGAREA][,MSGLEN]
[,NXTRPL][,RBA][,RECLEN]
[,RPLLEN][,TRANSID])

specifies the fields whose contents are to be displayed. Figure 12 explains the keywords you can code in the FIELDS operand for a request parameter list.

Key-word	Full-words	Description of the Field
ACB	1	Address of the access-method control block that relates the request parameter list to the data
AIXPC	1	Number of alternate-index pointers
AREA	1	Address of the work area which the program uses to process a data record to which access is defined by the request parameter list
AREALEN	1	Length of the work area whose address is given in AREA
ARG	1	Address of the field containing a search argument, if search arguments are being used
ECB	1	Address of an event control block, if any, in which VSAM indicates the completion of requests defined by the request parameter list
FDBK	1	The feedback field into which VSAM puts a return code upon completion of a request (for asynchronous requests, you must issue a CHECK to cause VSAM to put a return code into the feedback field; the meaning of the code in this field depends on the contents of register 15, which indicates whether the request was successful or failed because of a logical or physical error—see "Return Codes from the Request Macros" in the "Macro Instruction Descriptions and Return Codes" chapter)
FTNCD	1	Code that describes the function in which a logical or physical error occurred; indicates whether the upgrade set may have been modified incorrectly by the request
KEYLEN	1	Length of the search argument, if a generic key is used for a search argument
MSGAREA	1	Address of the area, if any, into which VSAM puts physical-error messages
MSGLEN	1	Length of the message area, if any
NXTRPL	1	Address of the next request parameter list, if another one is chained to this one
RBA	1	Relative byte address of the most recently processed record; you could use it to record the RBAs of records that you are retrieving or storing sequentially or by key
RECLEN	1	Length of the data record, access to which is defined by the request parameter list
RPLLEN	1	Length of a request parameter list
TRANSID	1	Number that relates modified buffers in a buffer pool; described in *OS/VS Virtual Storage Access Method (VSAM) Options for Advanced Applications*

Figure 12. FIELDS Operand Keywords for a Request Parameter List

Example: SHOWCB Macro (Display a Physical-Error Message)

In this example, a SHOWCB macro is used to display a physical-error message. This example assumes that there is no SYNAD routine (or the SYNAD exit is inactive), in which case, VSAM returns control to your program following the last executable instruction if a physical error occurs. Register 15 indicates a physical error (12), and the feedback field in the request parameter list contains a code identifying the error; the message area contains more details about the error. Register 1 points to the request parameter list.

```
REQUEST  RPL     MSGAREA=
                 MESSAGES,
                 MSGLEN=128
         .
         .
         .
         SHOWCB  AREA=MSGADDR,
                 FIELDS=MSGAREA,
                 LENGTH=4,
                 RPL=REQUEST
         LTR     15,15
         BNZ     CHECK0
         .
         .
         .
CHECK0   ...                          Display failed.
         .
         .
         .
MESSAGES DS      CL128                For VSAM to give you a detailed
                                      message about a physical error.
MSGADDR  DS      F                    For displaying the address of the
                                      message area with SHOWCB.
```

The RPL macro in this example provides for a message area, MESSAGES, of 128 bytes to be used for any physical-error message.

The SHOWCB macro's operands are:

- AREA and LENGTH, which specify a four-byte area, MSGADDR, to be used for displaying the address of the message area for the associated request parameter list.
- FIELDS, which specifies that the address of the message area is to be displayed.
- RPL, which specifies the name, REQUEST, of the request parameter list for which the message-area address is to be displayed.

TESTCB Macro (Test an Access-Method Control Block)

With the TESTCB macro, you can cause VSAM to set the condition code in the PSW (program status word) as a result of a comparison between the contents of a field that you specify and a value that you specify. Only one keyword can be specified each time TESTCB is issued. You might want to do this to:

- Find out whether an action (for example, opening a data set or activating an exit) has been done by VSAM or your program
- Find out what kind of a data set is being processed in order to alter your program logic as a result of the test.

You examine the PSW condition code after issuing a TESTCB macro (and examining the return code in register 15). For keywords specified as an option or a name, you test for an equal or unequal comparison; for keywords specified as an address or a number, you test for an equal, unequal, high, low, not-high, or not-low condition.

VSAM compares A to B, where A is the contents of the field and B is the value to which it is to be compared. A low condition means, for example, that A is lower than B—that is, that the value in the control block is lower than the value you specified. You may specify only one keyword to be tested. These keywords are the same as those that can be specified in the SHOWCB macro to display fields in an ACB. Fields can be tested at the same time they are displayed. If you specify a list of option codes for a keyword (for example, MACRF=(ADR,DIR)), each of them must equal the corresponding value in the control block for you to get an equal condition.

Some of the fields can be tested at any time; others only after a data set is opened. The ones that can be tested only after a data set is opened can, in the case of a key-sequenced data set, pertain either to the data or to the index. See the OBJECT operand.

The operands of the TESTCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. "Appendix C: Operand Notation for GENCB, MODCB, SHOWCB, and TESTCB" gives all the ways of coding each operand for the macros that work at execution.

See "Return Codes from the GENCB, MODCB, SHOWCB, and TESTCB Macros" for information on the return codes used to indicate whether the TESTCB request was successful.

Only one keyword can be specified each time you issue the macro. The format of the TESTCB macro used to test a field in an access-method control block is:

[*label*]	**TESTCB**	**ACB=***address* **[,ERET=***address* **]** **[,OBJECT=DATA | INDEX]** **,{ATRB=([ESDS][,KSDS][,REPL]** **[,RRDS][,SPAN][,SSWD]** **[,WCK] |** **ATRB=UNQ |** **CATALOG=YES | NO |** **CRA=SCRA | UCRA |** **MACRF=([ADR][,AIX][,CFX][,CNV]** **[,DDN][,DFR][,DIR][,DSN]** **[,GSR][,ICI][,IN][,KEY]** **[,LSR][,NCI][,NDF][,NFX]** **[,NIS][,NRM][,NRS][,NSR]** **[,NUB][,OUT][,RST][,SEQ]** **[,SIS][,SKP][,UBF]) |** **OFLAGS=OPEN |** **OPENOBJ=PATH | BASE | AIX |** **ACBLEN=***number* | **AVSPAC=***number* | **BSTRNO=***number* | **BUFND=***number* | **BUFNI=***number* | **BUFNO=***number* | **BUFSP=***number* | **CINV=***number* | **DDNAME=***ddname* | **ENDRBA=***number* | **ERROR=***number* | **EXLST=***address* | **FS=***number* | **KEYLEN=***number* | **LRECL=***number* | **MAREA=***address* | **MLEN=***number* | **NCIS=***number* | **NDELR=***number* | **NEXCP=***number* | **NEXT=***number* | **NINSR=***number* | **NIXL=***number* | **NLOGR=***number* | **NRETR=***number* | **NSSS=***number* | **NUPDR=***number* | **PASSWD=***address* | **RKP=***number* | **STMST=***address* | **STRNO=***number***}**

where:

ACB=*address*
specifies the address of the access-method control block whose information you want to test. You may omit it only if you're testing the length of an access-method control block (ACBLEN=number). (All VSAM access-method control blocks have the same length.)

ERET=*address*
specifies the address of a routine that VSAM is to give control if, because of an error, it is unable to test for the condition you specify. For example, testing AVSPAC in an access-method control block for an unopened data set would fail. VSAM indicates in register 15 whether it could do the test and, if not, indicates in register 0 the reason it couldn't. (The reasons are discussed under "Return Codes from the GENCB, MODCB, SHOWCB, and TESTCB Macros.") A failure trying to execute TESTCB indicates a basic logical problem in the processing program, so the error routine would probably issue an ABEND. If it lets the program continue, it must branch to the continuation point itself—and not return to VSAM.

OBJECT={DATA | INDEX}
specifies whether you want to test a field for data or for index.

ATRB=([ESDS][,KSDS][,REPL][,RRDS][,SPAN][,SSWD][,WCK])
specifies, for an open data set, the attribute that is to be tested for, as follows:

ESDS
entry-sequenced data set

KSDS
key-sequenced data set

REPL
some portion of the index is replicated

RRDS
relative record data set

SPAN
data set contains spanned records

SSWD
sequence set is adjacent to the data

WCK
write operations for the data set are being verified

ATRB=UNQ
specifies, for an open alternate index or path, that the alternate index requires unique keys. The test for ATRB=UNQ must be made with a separate TESTCB macro. VSAM examines the path control blocks for the UNQ attribute; VSAM examines the base cluster's control blocks for the other attributes. If other attributes are tested for, VSAM examines the base cluster's control blocks for all attributes: the test for ATRB=UNQ would give inaccurate results when applied to the base cluster's control blocks.

CATALOG=YES | NO
specifies that a test is to be made to determine, anytime, whether or not the access-method control block specifies a catalog data set.

CRA=SCRA | UCRA
specifies that a test is to be made to determine, anytime, whether catalog recovery area control blocks are to be built in system storage or user storage.

MACRF=([ADR][,AIX][,CFX][,CNV][,DDN][,DFR]
[,DIR][,DSN][,GSR][,ICI][,IN][,KEY][,LSR][,NCI]
[,NDF][,NFX][,NIS][,NRM][,NRS][,NSR][,NUB][,OUT][,RST]
[,SEQ][,SIS][,SKP][,UBF])

specifies that a test is to be made to determine, anytime, what option or combination of options is being used for processing.

OFLAGS=OPEN
specifies that a test is to be made to determine, after open, whether the data set identified by the control block has been opened.

OPENOBJ=PATH | BASE | AIX
specifies that a test is to be made to determine, after open, whether an opened object is a path, a base cluster, or an alternate index.

The remaining operands represent fields in an access-method control block that can be compared with the value specified. These fields are the same as those that can be displayed by using the SHOWCB macro. See Figure 12 for an explanation of these fields.

If you omit a routine to handle error conditions, you can examine register 15 following TESTCB by using a branch table, for example, but don't alter the PSW condition code that VSAM set to indicate the result of a test until you've had a chance to test it.

Example: TESTCB Macro (Test for Data-Set Attributes)

In this example, a TESTCB macro is used to determine whether a data set is a key-sequenced or an entry-sequenced data set.

```
LIST     RPL
         .
         .
         .
         SHOWCB  AREA=DATAFACT,
                 FIELDS=ACB,
                 LENGTH=4,
                 RPL=LIST
         LTR     15,15
         BNZ     CHECK0
         TESTCB  ACB=(*,                  Is the data set key-sequenced?
                 DATAFACT),
                 ATRB=KSDS,
                 ERET=CHECK0
         BE      KEYSEQ                   YES.
         .
         .
         .
KEYSEQ   . . .                            Data set is key sequenced.
CHECK0   . . .                            Display or test failed.
         .
         .
         .

DATAFACT DS      F                        For displaying address of
                                          access-method control block.
```

The SHOWCB macro's operands are:

- AREA and LENGTH, which specify a four-byte area, DATAFACT, aligned on a fullword boundary, to be used for the display.
- FIELDS and RPL, which specify that the address of the access-method control block in the LIST request parameter list is to be displayed.

The TESTCB macro's operands are:

- ACB, which specifies that a field in the access-method control block, the address of which is located at DATAFACT, is to be tested. The SHOWCB macro put the address of the access-method control block at DATAFACT.
- ATRB, which specifies that the access-method control block is to be tested to determine whether it is a key-sequenced data set.
- ERET, which specifiifies that a routine named CHECK0 is to be given control if an error occurs that makes it impossible to make the test.

There is no need to examine the feedback field in an EODAD routine because it can be assumed to contain the end-of-data-set indication.

TESTCB Macro (Test an Exit List)

The TESTCB macro can be used to test fields in an exit list.

The operands of the TESTCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. "Appendix C: Operand Notation for GENCB, MODCB, SHOWCB, and TESTCB" gives all the ways of coding each operand for the macros that work at execution.

The format of the TESTCB macro used to test fields in an exit list is:

[*label*]	**TESTCB**	**EXLST=***address* [**,ERET=***address*] **,EODAD={0 | ([***address***][,A | N][,L])} |** **JRNAD={0 | ([***address***][,A | N][,L])} |** **LERAD={0 | ([***address***][,A | N][,L])} |** **SYNAD={0 | ([***address***][,A | N][,L])}** [**,EXLLEN=***number*]

where:

label
is one to eight characters that provides a symbolic address for the TESTCB macro.

EXLST=*address*
specifies the address of the exit list whose information you want to test. You may omit it only if you're testing the maximum length of an exit list (EXLLEN=number). The TESTCB macro does not support the UPAD user exit.

ERET=*address*
specifies the address of a routine that VSAM is to give control if, because of an error, it is unable to test for the condition you specify. For example, testing AVSPAC in an access-method control block for an unopened data set would fail. VSAM indicates in register 15 whether it could do the test and, if not, indicates in register 0 the reason it couldn't. (The reasons are discussed under "Return Codes from the GENCB, MODCB, SHOWCB, and TESTCB Macros.") A failure trying to execute TESTCB indicates a basic logical problem in the processing program, so the error routine would probably issue an ABEND. If it lets the program continue, it must branch to the continuation point itself—and not return to VSAM.

EODAD={0 | ([*address*][,A | N][,L])} |
JRNAD={0 | ([*address*][,A | N][,L])} |
LERAD={0 | ([*address*][,A | N][,L])} |
SYNAD={0 | ([*address*][,A | N][,L])}

specifies the exit about which you are asking a yes-no question. If you code more than one operand for an exitname, each of them must equal the corresponding value in the control block for you to get an equal condition. The values that can be tested are:

0

specifies that a test is to be made to determine whether an entry is provided for the exit in the exit list.

address

specifies that a test is to be made to determine whether this is the address of the exit. Tests for an address result in an equal, unequal, high, low, not-high, or not-low condition. Tests for a combination of an address and A, N, or L result in an equal or unequal condition.

A | N

specifies that a test is to be made to determine whether an exit is active (A) or not active (N). Tests for A or N result in an equal or unequal condition.

L

specifies that a test is to be made to determine whether the *address* is the location of an 8-byte field containing the name of a module to be loaded rather than the entry point of the routine. Tests for L result in an equal or unequal condition.

EXLLEN=*number*

specifies either the maximum length that an exit list can have (if you don't code the EXLST operand) or the actual length of the exit list indicated by the EXLST operand. If you specify an exit, you may not also specify EXLLEN; if you specify EXLLEN, you may not also specify an exit.

If you omit a routine to handle error conditions, you can examine register 15 following TESTCB by using a branch table, for example, but don't alter the PSW condition code that VSAM set to indicate the result of a test until you've had a chance to test it.

Example: TESTCB Macro (Use a Branch Table)

In this example, a TESTCB macro is used to test whether ENDPROC is the routine supplied for the EODAD exit in the exit list EXITS, and whether the EODAD exit is active. A branch table is used to determine whether the test is successful.

```
          TESTCB EODAD=(ENDPROC,          Is ENDPROC supplied and is the exit
                 A),EXLST=EXITS           active?
          B      *+4(15)
```

If the test was made successfully, register 15 contains 0 and the next instruction is executed.

```
          B      TEST1
```

If it was unsuccessful, register 15 contains 4 and the next instruction is executed.

```
          ABEND  2,DUMP
TEST1     BNE    NO
YES       ...                             Yes, ENDPROC is supplied and active.
NO        ...                             ENDPROC isn't supplied, or the exit
                                          isn't active.
```

TESTCB Macro (Test a Request Parameter List)

The TESTCB macro can be used to test fields in a request parameter list.

The operands of the TESTCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. "Appendix C: Operand Notation for GENCB, MODCB, SHOWCB, and TESTCB" gives all the ways of coding each operand for the macros that work at execution.

See "Return Codes from the GENCB, MODCB, SHOWCB, and TESTCB Macros" for information on the return codes used to indicate whether the TESTCB request was successful.

The format of the TESTCB macro to test fields in a request parameter list is:

[*label*]	**TESTCB**	**RPL**=*address* [,**ERET**=*address*] {**AIXFLAG**=**AIXPKP** | **AIXPC**=*number* | **FTNCD**=*number* | **I/O**=**COMPLETE** | **OPTCD**=([**ADR**][,**ARD**][**ASY**][,**BWD**] [,**CNV**][,**DIR**][,**FKS**][,**FWD**] [,**GEN**][,**KEQ**][,**KEY**][,**KGE**][,**LOC**] [,**LRD**][,**MVE**][,**NSP**][,**NUP**][,**SEQ**] [,**SKP**][,**SYN**][,**UPD**]) | **ACB**=*address* | **AREA**=*address* | **AREALEN**=*number* | **ARG**=*address* | **ECB**=*address* | **FDBK**=*number* | **KEYLEN**=*number* | **MSGAREA**=*address* | **MSGLEN**=*number* | **NXTRPL**=*address* | **RBA**=*number* | **RECLEN**=*number* | **RPLLEN**=*number* | **TRANSID**=*number*}

where:

label
is one to eight characters that provides a symbolic address for the TESTCB macro.

RPL=*address*
specifies the address of the request parameter list whose information you want to test. You may omit it only if you're testing the length of a request parameter list (RPLLEN=number). (All request parameter lists have the same length.)

ERET=*address*
specifies the address of a routine that VSAM is to give control if, because of an error, it is unable to test for the condition you specify. For example, testing AVSPAC in an access-method control block for an unopened data set would fail. VSAM indicates in register 15 whether it could do the test and, if not, indicates in register 0 the reason it couldn't. (The reasons are discussed under "Return Codes from the GENCB, MODCB, SHOWCB, and TESTCB Macros.") A failure trying to execute TESTCB indicates a basic logical problem in the processing program, so the error routine would probably issue an ABEND. If it lets the program continue, it must branch to the continuation point itself—and not return to VSAM.

AIXFLAG=AIXPKP
specifies that prime-key pointers are used rather than RBAs.

AIXPC= *number*
specifies the pointer count.

FTNCD= *number*
specifies whether the upgrade set is correct or may have been modified by a request. These codes are described under "Function Codes" in the chapter "Macro Instruction Descriptions and Return Codes."

IO=COMPLETE
specifies that a test is to be made to determine whether an asynchronous request has been completed. (When you issue a CHECK macro, you suspend processing until a request has been completed if it hasn't yet been completed.)

OPTCD=([ADR][,ARD][,ASY][,BWD][,CNV][,DIR][,FKS][,FWD]
[,GEN][,KEQ][,KEY][,KGE][,LOC][,LRD][,MVE][,NSP]
[,NUP][,SEQ][,SKP][,SYN][,UPD])

specifies that a test is to be made to determine what option or combination of options is being used for the request. See Figure 10 for a description of these options.

The remaining operands specify fields in a request parameter list and values; the contents of a field are to be compared to the specified value. These fields are the same as those that can be displayed by using a SHOWCB macro. See Figure 12 in chapter "SHOWCB Macro (Display a Request Parameter List)" for an explanation of these fields. Fields can be tested at the same time they are displayed.

You may specify only one keyword. If you code a list of option codes (for example, OPTCD=(KEY,DIR)), each of them must equal the corresponding value in the control block for you to get an equal condition.

If you omit a routine to handle error conditions, you can examine register 15 following TESTCB by using a branch table, for example, but don't alter the PSW condition code that VSAM set to indicate the result of a test until you've had a chance to test it. Examples of using an ERET routine and using a branch table are given at the end of the section.

Example: TESTCB Macro (Test a Request Parameter List)

```
          TESTCB RPL=(3),
                 RECLEN=80
          BE     NOCHNGE
CHANGE    ...                 Because the record length in the request
                              parameter list was not 80, the length
                              indicator must be modified so that it is
                              80.
NOCHNGE   ...                 Because the record length in the request
                              parameter list was 80, no change is
                              required.
```

The TESTCB macro's operands are:

- RPL, which specifies that the address of the request parameter list to be tested is contained in register 3.
- RECLEN, which specifies that the record length indicated in the request parameter list is to be tested to determine whether it is 80.

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 in the Use of the ISAM Interface" later in this chapter.

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 15 shows the relationship between ISAM programs processing VSAM data with the ISAM interface and VSAM programs processing the data.

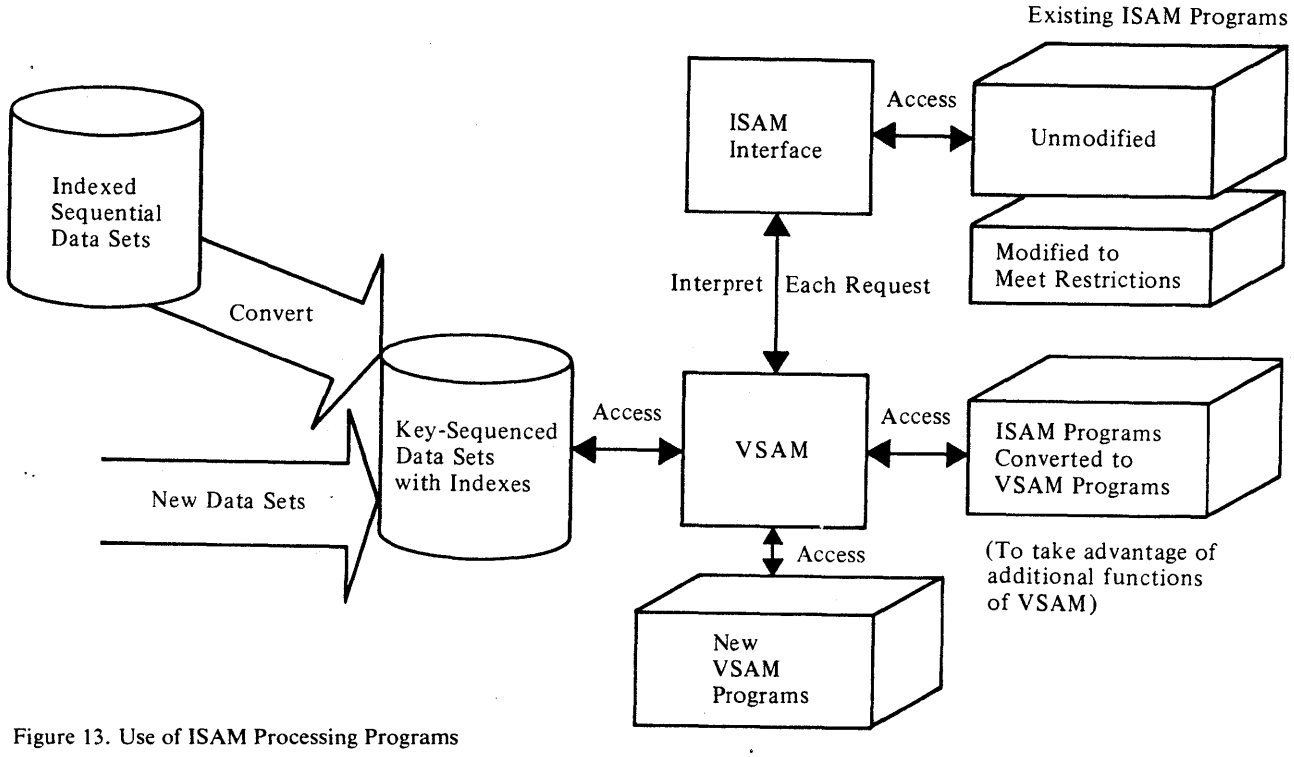


Figure 13. Use of ISAM Processing Programs

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 14 shows QISAM error conditions and the meaning they have when the ISAM interface is being used.

Figure 15 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.

Figure 16 gives the contents of registers 0 and 1 when a SYNAD routine specified in a DCB gets control.

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 17 gives the contents of registers 0 and 1 when a SYNAD routine specified by way of AMP gets control.

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 non-mountable volume
Bit 1	Invalid device address			Always zero
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
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
Bit 3	Overflow record	Interface		Always one

Figure 14 (Part 1 of 2). QISAM Error Conditions

Byte and Offset	QISAM Meaning	Error Detected By	Request Parameter List Error Code	Interface/VSAM Meaning
Bit 4	Length of logical recor	Interface		Length of Logical record is greater than
		VSAM	108	Invalid record length
Bits 5 - 7	Reserved			Always zero

Figure 14 (Part 2 of 2). QISAM Error Conditions

Byte and Offset	BISAM Meaning	Error Detected By	Request Parameter List Error Code	Interface/VSAM Meaning
DECBEXC1				
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
		VSAM	96	Key change attempted
Bit 4	Uncorrectable I/O error	VSAM		Physical error (register 15 will contain a
Bit 5	Unreachable block	VSAM		Logical error not covered by any other e
Bit 6	Overflow record	Interface		Always one for READ requests
Bit 7	Duplicate record	VSAM	8	Duplicate record
DECBEXC2				
Bits 0 - 5	Reserved			Always zero
Bit 6	Channel program initia			Always zero
Bit 7	Previous macro was R	Interface		Previous macro was READ KU

Figure 15. BISAM Error Conditions

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 16. Register Contents for DCB-Specified ISAM SYNAD Routine

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 17. 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 *OS/VS Data Management Macro Instructions.*

Figure 18 shows abnormal-end (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

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 18. ABEND Codes Issued by the ISAM Interface

- Checkpoint/restart
- Abnormal end

Figure 19 shows the DEB fields that are supported by the ISAM interface; field meanings are the same as in ISAM, except as noted.

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 19. 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 the appropriate *Access Method Services* publication 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 the appropriate *Access Method Services* publication 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. In an OS/VS2 MVS system, data sets can also be allocated dynamically. See *OS/VS2 JCL* and *OS/VS2 System Programming Library: Job Management* for a description of dynamic allocation.

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) in an MVS system or is defined in a DD statement in VS1 or SVS that includes DISP=OLD, volume and unit information, the AMP parameter, and DSNAME=dsname, where dsname is the name of the key-sequenced data set. In either system, use a STEPCAT or JOBCAT DD statement as described in the chapter "Job Control Language" to make user catalogs available; in an OS/VS2 MVS system, you may also use dynamic allocation.

With ISAM, deleted records are flagged as deleted, but are not actually removed from the data set. 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. Certain DCB-type information may be specified in the AMP parameter, which is described later in this chapter.

Figure 20 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
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), 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

Figure 20 (Part 1 of 2).DCB Fields Supported by ISAM Interface

Field Name	Meaning
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 20 (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 the chapter "Job Control Language." 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 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 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 like 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 along 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 tie up concurrently. Neither VSAM nor the ISAM interface can anticipate the number, so you must 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 the number of such requests for each subtask together, plus an additional one for each subtask that sequentially processes the same data set.) In a create step, STRNO cannot be greater than 1.

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 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 ties up 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 tied up 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.

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 merely examines DCB or DECB exception codes.

Restrictions in 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 VS 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. Not all of the fields in a DCB can be counted 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 take it out. 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 which 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 ties 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.

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).

For details on the use of the REPRO command and its parameters, see the *Access Method Services* publication appropriate for your system.

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.

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

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." If the error cannot be corrected, close the data set and either terminate processing or return to VSAM.

Figure 21 gives the contents of the registers when VSAM exits to the 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. See "Logical

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 21. Contents of Registers at Entry to LERAD Exit Routine

Errors" in the chapter "Request Macros" for a description of these return codes.

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."

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 22 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 request macro; if you issued a CLOSE macro, the request parameter list was built by VSAM to process the close request. Register 1 must contain this address if 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. The register doesn't contain the physical-error indicator.

Figure 22. 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 "Physical Errors" in the chapter "Request Macros" 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 "Physical Errors" in the chapter "Request Macros" 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. The name of your routine is specified 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."

See the appropriate *Access Method Services* publication for information about how exception exits are established, changed, or nullified.

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."

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 23 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 23. 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 ATRB characteristics.

JRNAD Exit Routine to Journalize Transactions

A JRNAD 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.

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.

Figure 24 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) 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' through X'FC' Reserved 1 byte — Reserved.
2-13	Unpredictable.
14	Return address to VSAM.
15	Entry address to the JRNAD routine.

[1] Described in *OS/VS Virtual Storage Access Method (VSAM Options for Advanced Applications*

Figure 24. 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 journalizing 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 one. 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

The user 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.

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
2-13	Unpredictable
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 is 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)	8	Reserved
20 (14)	1	Reason code: X'00' VSAM is about to wait X'04'–'FC' Reserved

If the UPAD exit routine modifies register 14 (for example, by issuing a TESTCB, the routine must restore register 14 before returing 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.

When VSAM regains control from a UPAD exit that was taken for reason code zero, VSAM tests the ECB for completion. If the I/O request has not completed, VSAM issues a WAIT. Once the ECB has been posted complete, VSAM clears it in preparation for the next WAIT.

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 requester'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.

You may, through the Access Method Services DEFINE command, identify your user security-verification routine (USVR) and associate up to 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 USVR 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 his ID when he defines the data set and then allow only the owner to gain access to the data set.

Figure 25 gives the contents of the registers when VSAM gives control to the USVR.

If the USVR is being used by more than one task at a time, you must code the USVR 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 0s)
	8 bytes	Owner identification (or 0s)
	8 bytes	The password that the requester gave (it has been verified by VSAM)
	2 bytes	Length of the USAR (in binary)
	—	The USAR
2-13	Unpredictable.	
14	Return address to VSAM.	
15	Entry address to the USVR. 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 25. Communication with User-Security-Verification 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. (The exception exit routine is described in the *Access Method Services* publication appropriate for your system.) When your exit routine

completes its processing, it can return to your main program in one of two ways:

A. 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.

B. 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 A provides the easier way to return to your program. However, there is a special situation that requires you to return via method B; 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 non-reentrant,
*                              Register 15 is entry address
         .
         .                     Save caller's registers (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: SUMMARY OF MACROS

For easy reference, the formats of all of the macros described in this book are repeated in this one place in alphabetic order.

BLDVRP, DLVRP, GETIX, MRKBFR, PUTIX, SCHBFR, SHOWCAT, VERIFY, and WRTBFR are described in *OS/VS Virtual Storage Access Method (VSAM) Options for Advanced Applications.*

ACB (Generate an Access-Method Control Block)

[*label*]	**ACB**	**[AM=VSAM]** **[,BSTRNO=** *number* **]** **[,BUFND=** *number* **]** **[,BUFNI=** *number* **]** **[,BUFSP=** *number* **]** **[,CATALOG={YES | NO}]** **[,CRA={SCRA | UCRA}]** **[,DDNAME=** *ddname* **]** **[,EXLST=** *address* **]** **[,MACRF=([ADR][,CNV][,KEY]** **[,CFX | NFX]** **[,DDN | DSN]** **[,DFR | NDF]** **[,DIR][,SEQ][,SKP]** **[,ICI | NCI]** **[,IN][,OUT]** **[NIS | SIS]** **[,NRM | AIX]** **[NRS | RST]** **[NSR | LSR | GSR]** **[,NUB | UBF])]** **[,MAREA=** *address* **]** **[,MLEN=** *number* **]** **[,PASSWD=** *address* **]** **[,STRNO=** *number* **]**

CHECK (Suspend Processing)

[*label*]	**CHECK**	**RPL=** *address*

CLOSE (Disconnect Program and Data)

[*label*]	**CLOSE**	(*address* [,(*options*)]...] **[,TYPE=T]**

ENDREQ (Terminate a Request)

[*label*]	**ENDREQ**	**RPL=** *address*

ERASE (Delete a Record)

[*label*]	**ERASE**	**RPL=** *address*

EXLST (Generate an Exit List)

[*label*]	**EXLST**	**[AM=VSAM]** **[,EODAD=(***address* **[,A | N][,L])]** **[,JRNAD=(***address* **[,A | N][,L])]** **[,LERAD=(***address* **[,A | N][,L])]** **[,SYNAD=(***address* **[,A | N][,L])]** **[UPAD=(***address***[,A | N][,L])]**

GENCB (Generate an Access-Method Control Block)

[*label*]	**GENCB**	**BLK=ACB** **[,AM=VSAM]** **[,BSTRNO=** *number* **]** **[,BUFND=** *number* **]** **[,BUFNI=** *number* **]** **[,BUFSP=** *number* **]** **[,CATALOG={YES | NO}]** **[,COPIES=** *number* **]** **[,CRA={SCRA | UCRA}]** **[,DDNAME=***ddname* **]** **[,EXLST=** *address* **]** **[,LENGTH=** *number* **]** **[,MACRF=([ADR][,CNV][,KEY]** **[,CFX | NFX]** **[,DDN | DSN]** **[,DFR | NDF]** **[,DIR][,SEQ][,SKP]** **[,ICI | NCI]** **[,IN][,OUT]** **[,NIS | SIS]** **[,NRM | AIX]** **[,NRS | RST]** **[,NSR | LSR]** **[,NUB | UBF])]** **[,MAREA=** *address* **]** **[,MLEN=** *number* **]** **[,PASSWD=** *address* **]** **[,STRNO=** *address* **]** **[,WAREA=** *address* **]**

GENCB (Generate an Exit List)

[*label*]	**GENCB**	**BLK=EXLST** **[,AM=VSAM]** **[,COPIES=** *number* **]** **[,EODAD=(***address* **[,A | N][,L])]** **[,JRNAD=(***address* **[,A | N][,L])]** **[,LENGTH=** *number* **]** **[,LERAD=(***address* **[,A | N][,L])]** **[,SYNAD=(***address* **[,A | N][,L])]** **[,WAREA=** *address* **]**

GENCB (Generate a Request Parameter List)

[*label*]	GENCB	**BLK=RPL** [,**ACB=***address*] [,**AM=VSAM**] [,**AREA=***address*] [,**AREALEN=***number*] [,**ARG=***address*] [,**COPIES=***number*] [,**ECB=***address*] [,**KEYLEN=***number*] [,**LENGTH=***number*] [,**MSGAREA=***address*] [,**MSGLEN=***number*] [,**NXTRPL=***address*] [,**OPTCD=([ADR** | **CNV** | **KEY]** [,**DIR** | **SEQ** | **SKP]** [**ARD** | **LRD]** [**FWD** | **BWD]** [,**ASY** | **SYN]** [,**NSP** | **NUP** | **UPD]** [,**KEQ** | **KGE]** [,**FKS** | **GEN]** [,**LOC** | **MVE])]** [,**RECLEN=***number*] [,**TRANSID=***number*] [,**WAREA=***address*]

GET (Retrieve a Record)

[*label*]	**GET**	**RPL=***address*

MODCB (Modify an Access-Method Control Block)

[*label*]	**MODCB**	**ACB=***address* **[,BSTRNO=***number*] **[,BUFND=***number*] **[,BUFNI=***number*] **[,BUFSP=***number*] **[,CATALOG={YES | NO}]** **[,CRA={SCRA | UCRA}]** **[,DDNAME=***ddname*] **[,EXLST=***address*] **[,MACRF=([ADR][,CNV][,KEY]** **[,CFX | NFX]** **[,DDN | DSN]** **[,DFR | NDF]** **[,DIR][,SEQ][,SKP]** **[ICI | NCI]** **[,IN][,OUT]** **[,NIS | SIS]** **[,NRM | AIX]** **[,NRS | RST]** **[,NSR | LSR** **[,NUB | UBF])]** **[,MAREA=***address*] **[,MLEN=***number*] **[,PASSWD=***address*] **[,STRNO=***number*]

MODCB (Modify an Exit List)

[*label*]	**MODCB**	**EXLST=***address* **[,EODAD=(***address* **[,A | N][,L])]** **[,JRNAD=(***address* **[,A | N][,L])]** **[,LERAD=(***address* **[,A | N][,L])]** **[,SYNAD=(***address* **[,A | N][,L])]**

MODCB (Modify a Request Parameter List)

[*label*]	**MODCB**	**RPL**=*address* [,**ACB**=*address*] [,**AREA**=*address*] [,**AREALEN**=*number*] [,**ARG**=*address*] [,**ECB**=*address*] [,**KEYLEN**=*number*] [,**MSGAREA**=*address*] [,**MSGLEN**=*number*] [,**NXTRPL**=*address*] [,**OPTCD**=([**ADR** | **CNV** | **KEY**] [,**ARD** | **LRD**] [,**FWD** | **BWD**] [,**DIR** | **SEQ** | **SKP**] [,**ASY** | **SYN**] [,**NSP** | **NUP** | **UPD**] [,**KEQ** | **KGE**] [,**FKS** | **GEN**] [,**LOC** | **MVE**]) [,**RECLEN**=*number*] [,**TRANSID**=*number*]

OPEN (Connect Program and Data)

[*label*]	**OPEN**	(*address* [,(*options*)]...)

POINT (Position for Access)

[*label*]	**POINT**	**RPL=** *address*

PUT (Store a Record)

[*label*]	**PUT**	**RPL=** *address*

RPL (Generate a Request Parameter List)

[*label*]	**RPL**	**ACB=** *address* [,**AM=VSAM**] [,**AREA=** *address*] [,**AREALEN=** *number*] [,**ARG=** *address*] [,**ECB=** *address*] [,**KEYLEN=** *number*] [,**MSGAREA=** *address*] [,**MSGLEN=** *number*] [,**NXTRPL=** *address*] [,**OPTCD=**([**ADR** | **CNV** | **KEY**] [,**DIR** | **SEQ** | **SKP**] [,**ARD** | **LRD**] [,**FWD** | **BWD**] [,**ASY** | **SYN**] [,**NSP** | **NUP** | **UPD**] [,**KEQ** | **KGE**] [,**FKS** | **GEN**] [,**NWAITX** | **WAITX**] [,**LOC** | **MVE**])] [,**RECLEN=** *number*] [,**TRANSID=** *number*]

SHOWCB (Display Fields of an Access-Method Control Block)

[*label*]	**SHOWCB**	**ACB=***address* **,AREA=***address* **,LENGTH=***number* **[,OBJECT={DATA | INDEX}]** **,FIELDS=([,ACBLEN][,AVSPAC][,BFRFND]** **[,BSTRNO][,BUFND][,BUFNI]** **[,BUFNO][,BUFRDS][,BUFSP]** **[,CINV][,DDNAME][,ENDRBA]** **[,ERROR][,EXLST][,FS]** **[,HALCRBA][,KEYLEN][,LRECL]** **[,MAREA][,MLEN][,NCIS]** **[,NDELR][,NEXCP][,NEXT]** **[,NINSR][,NIXL][,NLOGR]** **[,NRETR][NSSS][,NUIW]** **[,NUPDR][,PASSWD][,RKP]** **[,STMST][,STRMAX][,STRNO]** **[UIW])**

SHOWCB (Display Fields of an Exit List)

[*label*]	**SHOWCB**	**AREA=***address* **,EXLST=***address* **,FIELDS=([EODAD][,EXLLEN][,JRNAD]** **[,LERAD][,SYNAD])** **,LENGTH=***number*

SHOWCB (Display Fields of a Request Parameter List)

[*label*]	**SHOWCB**	**AREA=***address* **,FIELDS=([ACB][,AIXPC][,AREA][,AREALEN]** **[,ARG][,ECB][,FDBK][,FTNCD]** **[,KEYLEN][,MSGAREA]** **[,MSGLEN][,NXTRPL][,RBA]** **[,RECLEN][,RPLLEN][,TRANSID]** **,LENGTH=***number* **,RPL=***address*

TESTCB (Test a Field of an Access-Method Control Block)

[*label*]	TESTCB	ACB=*address* [,ERET=*address*] [,OBJECT=DATA | INDEX] ,{ATRB=([ESDS][,KSDS][,REPL][,RRDS] [,SPAN][,SSWD][,WCK]) | ATRB=UNQ CATALOG={YES | NO} | MACRF=([ADR][,AIX][,CFX][,CNV][,DDN] [,DFR][,DIR][,DSN][,GSR][,ICI] [,IN][,KEY][,LSR][,NCI][,NDF] [,NFX][,NIS],NRM][,NRS][,NSR] [,NUB][,OUT][,RST][,SEQ][SIS] [,SKP][,UBF],) | OFLAGS=OPEN | OPENOBJ={PATH | BASE | AIX} | ACBLEN=*number* | AVSPAC=*number* | BSTRNO=*number* | BUFND=*number* | BUFNI=*number* | BUFNO=*number* | BUFSP=*number* | CINV=*number* | DDNAME=*ddname* | ENDRBA=*number* | ERROR=*number* | EXLST=*address* | FS=*number* | KEYLEN=*number* | LRECL=*number* | MAREA=*address* | MLEN=*number* | NCIS=*number* | NDELR=*number* | NEXCP=*number* | NEXT=*number* | NINSR=*number* | NIXL=*number* | NLOGR=*number* | NRETR=*number* | NSSS=*number* | NUPDR=*number* | PASSWD=*address* | RKP=*number* | STMST=*address* | STRNO=*number*}

TESTCB (Test a Field of an Exit List)

[*label*]	**TESTCB**	**,EXLST=***address* **[,ERET=***address* **]** **{EODAD={0 | ([***address* **][,A | N][,L])} |** **JRNAD={0 | ([***address* **][,A | N][,L])} |** **LERAD={0 | ([***address* **][,A | N][,L])} |** **SYNAD={0 | ([***address* **][,A | N][,L])}}** **[,EXLLEN=***number***]**

TESTCB (Test a Field of a Request Parameter List)

[*label*]	**TESTCB**	**RPL=***address* **[,ERET=***address* **]** **{IO=COMPLETE |** **OPTCD=([ADR][,ARD][,ASY][,BWD][,CNV]** **[,DIR][,FKS][,FWD][,GEN][,KEQ]** **[,KEY][,KGE],LOC][,LRD][,MVE]** **[,NSP][,NUP][,SEQ][,SKP][,SYN]** **[,UPD]) |** **RBA=***number* **|** **RECLEN=***number* **|** **RPLLEN=***number* **|** **ACB=***address* **|** **AIXFLAG=AIXPKP |** **AIXPC=***number* **|** **AREA=***address* **|** **AREALEN=***number* **|** **ARG=***address* **|** **ECB=***address* **|** **FDBK=***number* **|** **FTNCD=***number* **|** **KEYLEN=***number* **|** **MSGAREA=***address* **|** **MSGLEN=***number* **|** **NXTRPL=***address* **|** **TRANSID=***number***}**

APPENDIX B: LIST, EXECUTE, AND GENERATE FORMS OF GENCB, MODCB, SHOWCB, AND TESTCB

The standard forms of the GENCB, MODCB, SHOWCB, and TESTCB macros build a parameter list describing in codes the actions indicated by the operands you specify and pass the list to VSAM to take the indicated action. The list, execute, and generate forms of GENCB, MODCB, SHOWCB, and TESTCB allow you to write reentrant programs, share parameter lists, and to modify a parameter list before using it.

Following is a brief description of the list, execute, and generate forms:

- The list form is used to build the parameter list either inline (referred to as *simple list*) or in an area remote from the macro expansion (referred to as *remote list*). Both the simple- and the remote-list forms allow you to build a single parameter list that can be shared.
- The execute form is used to modify a parameter list and to pass it to VSAM for action.
- The generate form is used to build the parameter list in a remote area and to pass it to VSAM for action.

The list, execute, and generate forms of the GENCB, MODCB, SHOWCB, and TESTCB macros have the same format as the standard forms, with the exception of:

- An additional keyword, MF
- Some operands' being optional or not allowed

The sections that follow describe the format of the MF keyword and the use of list, execute, and generate forms. They indicate the optional and required operands.

List-Form Keyword

The format of the MF keyword for the list form is:

MF={L | (L,*address* **[,***label***])}**

where:

L
specifies that this is the list form of the macro.

address
specifies the address of a remote area in which the parameter list is to be built. The area must begin on a fullword boundary. You can specify the address in register notation or as an expression valid for a relocatable A-type address constant or a direct or indirect S-type address constant.

label
is a unique name that is used in an EQU instruction in the expansion of the macro; label is equated to the length of the parameter list. You do not have to know the length of the parameter list if you code label; the expansion of the macro determines the amount of storage required.

Because the MF=L expansion does not include executable code, register notation and expressions that generate S-type address constants cannot be used.

If you code MF=L, the parameter list is built inline, which means that the program is not reentrant if the parameter list is modified at execution.

If you code MF=(L,address), the parameter list is built in the remote area specified, and the area must be large enough for the parameter list.

The size, in fullwords, of a parameter list is:

- For GENCB, 4, plus 3 times the number of ACB, EXLST, or RPL keywords specified (plus 1 for DDNAME, EODAD, JRNAD, LERAD, or SYNAD)
- For MODCB, 3, plus 3 times the number of ACB, EXLST, or RPL keywords specified (plus 1 for DDNAME, EODAD, JRNAD, LERAD, or SYNAD)
- For SHOWCB, 5, plus 2 times the number of fields specified in the FIELDS operand
- For TESTCB, 8 (plus 1 for DDNAME, STMST, EODAD, JRNAD, LERAD, or SYNAD)

If you code MF=(L,address,label), the parameter list is built in the remote area specified. The expansion of the macro equates label with the length of the parameter list.

Execute-Form Keyword

The format of the MF keyword for the execute form is:

MF=(E,*address***)**

where:

E
specifies that this is the execute form of the macro.

address
is the address of the parameter list.

The expansion of the execute form of the macro results in executable code that causes:

1. A parameter list to be modified if requested
2. Control to be passed to a routine that satisfies the request

You may not use the execute form to add an entry to a parameter list. If you try to add an entry, an error code of 8 is returned to you in register 15.

Generate-Form Keyword

The format of the MF keyword for the generate form is:

MF=(G,*address* **[,***label* **])**

where:

G
specifies that this is the generate form of the macro.

address
specifies the address of a remote area in which the parameter list is to be built. The area must begin on a fullword boundary.

label
is a unique name that is used in an EQU instruction in the expansion of the macro; label is equated to the length of the parameter list. You do not have to know the length of the parameter list if you code label; the expansion of the macro determines the amount of storage required.

If you code MF=(G,address), the parameter list is built in the remote area specified.

If you code MF=(G,address,label), the parameter list is built in the remote area specified. The expansion of the macro equates the length of the parameter list to label.

Optional and Required Operands

Keywords that are required in the standard forms of the GENCB, MODCB, SHOWCB, and TESTCB macros may be optional in the list, execute, and generate forms or may not be allowed in the execute form. The meaning of the keywords, however, and the notation that may be used to express addresses, names, numbers, and option codes are the same. See the chapter "Control Block Macros" for the meaning of keywords. See "Appendix C: Operand Notation for GENCB, MODCB, SHOWCB, and TESTCB" for an explanation of how operands may be coded.

List Form of GENCB

The format of the list form of GENCB is:

[*label*]	**GENCB**	**BLK={ACB | EXLST | RPL}** **[,AM=VSAM]** **[,COPIES=***number***]** **[,***keyword***={***address* |*name* | *number* | *option* **},...]** **[,LENGTH=***number***]** **,MF={L | (L,***address* **[,***label* **])}** **[,WAREA=***address***]**

Execute Form of GENCB

The format of the execute form of GENCB is:

[*label*]	**GENCB**	**BLK={ACB | EXLST | RPL}** **[,AM=VSAM]** **[COPIES=** *number*] [,*keyword* ={ *address* | *name* | *number* | *option* },...] **[,LENGTH=** *number*] **,MF=(E,** *address*) **[,WAREA=** *address*]

Generate Form of GENCB

The format of the generate form of the GENCB macro is:

[*label*]	**GENCB**	**BLK={ACB | EXLST | RPL}** **[,AM=VSAM]** **[,COPIES=** *number*] [,*keyword* ={ *address* | *name* | *number* | *option* },...] **[,LENGTH=** *number*] **,MF=(G,** *address* [, *label*]) **[,WAREA=** *address*]

List Form of MODCB

The format of the list form of MODCB is:

[*label*]	**MODCB**	{**ACB | EXLST | RPL**{=*address* ,*keyword* ={ *address* | *name* | *number* | *option* },... **,MF={L | (L,** *address* [, *label*])}

Execute Form of MODCB

Note: If the execute form of MODCB is used and EXLST is used as a keyword to be processed, the block must be identified by ACB=.

The format of the execute form of MODCB is:

[*label*]	**MODCB**	[{**ACB | EXLST | RPL**}=*address*] [,*keyword* ={ *address* | *name* | *number* | *option* },...] **,MF=(E,** *address*)

Generate Form of MODCB

The format of the generate form of MODCB is:

[*label*]	**MODCB**	{**ACB | EXLST | RPL**{=*address* ,*keyword* ={ *address* | *name* | *number* | *option* },.... **,MF=(G,** *address* [, *label*])

List Form of SHOWCB

The format of the list form of SHOWCB is:

[*label*]	**SHOWCB**	[{**ACB** | **EXLST** | **RPL**}=*address*] ,**AREA**=*address* ,**FIELDS**=(*keyword*[,*keyword*,...]) ,**LENGTH**=*number* ,**MF**={**L** | (**L**,*address*[,*label*])} [,**OBJECT**={**DATA** | **INDEX**}]

Execute Form of SHOWCB

The format of the execute form of SHOWCB is:

[*label*]	**SHOWCB**	[{**ACB** | **EXLST** | **RPL**}=*address*] ,**AREA**=*address* ,**MF**=(**E**,*address*) [,**OBJECT**={**DATA** | **INDEX**}]

Generate Form of SHOWCB

The format of the generate form of SHOWCB is:

[*label*]	**SHOWCB**	[{**ACB** | **EXLST** | **RPL**}=*address*] ,**AREA**=*address* ,**FIELDS**=(*keyword*[,*keyword*,...]) ,**LENGTH**=*number* ,**MF**=(**G**,*address*[,*label*]) [,**OBJECT**={**DATA** | **INDEX**}]

List Form of TESTCB

Note: If the execute form of TESTCB is used and EXLST is used as a keyword to be processed, the block must be identified by ACB=.

The format of the list form of TESTCB is:

[*label*]	**TESTCB**	[{**ACB** | **EXLST** | **RPL**}=*address*] [,**ERET**=*address*] ,*keyword*={*address* | *name* | *number* | *option*} ,**MF**={**L** | (**L**,*address*[,*label*])} [,**OBJECT**={**DATA** | **INDEX**}]

Execute Form of TESTCB

Note: If the execute form of TESTCB is used and EXLST is used as a keyword to be processed, the block must be identified by ACB=.

The format of the execute form of TESTCB is:

[*label*]	**TESTCB**	[{**ACB** | **EXLST** | **RPL**}=*address*] [,**ERET**=*address*] [,*keyword*={*address* | *name* | *number* | *option*}] ,**MF**=(**E**,*address*) [,**OBJECT**={**DATA** | **INDEX**}]

Generate Form of TESTCB

The format of the generate form of TESTCB is:

[*label*]	**TESTCB**	[{**ACB** | **EXLST** | **RPL**}=*address*] [,**ERET**=*address*] ,*keyword*={*address* | *name* | *number* | *option*} ,**MF**=(**G**,*address*[,*label*]) [,**OBJECT**={**DATA** | **INDEX**}]

Use of List, Execute, and Generate Forms

Again, the list, execute, and generate forms allow you to use GENCB, MODCB, SHOWCB, and TESTCB in reentrant programs and allow you to share parameter lists. Figure 26 indicates which forms of these macros should be used in reentrant/nonreentrant and shared/nonshared environments.

	Reentrant	**Nonreentrant**
Shared	**MF**=(**L**,*address*[,*label*]) **MF**=(**E**,*address*)	**MF**=**L** **MF**=(**E**,*address*)
Nonshared	**MF**=(**G**,*address*[,*label*])	Standard Form

Figure 26. Reentrant Programming

The figure shows that:

- To share parameter lists in a reentrant program, the remote-list form should be used in conjunction with the execute form.
- To share parameter lists in a nonreentrant program, the simple-list form should be used in conjunction with the execute form.
- If you do not intend to share parameter lists, the generate form should be used in reentrant programs and the standard form should be used for nonreentrant programs.

The examples that follow illustrate how the list, execute, and generate forms work.

Example: Generate Form (Reentrant)

In this example, the generate form of GENCB is used to create a default request parameter list (RPL) in a reeentrant environment.

```
LA       10,LEN1              Get length of the parameter list.
GETMAIN  R,LV=(10)            Get storage for the area in which the
                              parameter list is to be built.
LR       2,1                  Save address of parameter-list area.
GENCB    BLK=RPL,
         MF=(G,(2),LEN1)
```

The macro expansion equates LEN1 to the length of the parameter list, as follows:

```
+LEN1 EQU 16
```

The parameter list will be built in the area acquired by the GETMAIN macro and pointed to by register 2. This list is used by VSAM to build the RPL. VSAM returns the RPL address in register 1 and the RPL length in register 0. If the WAREA and LENGTH parameters are used, the RPL will be built at the WAREA address.

Example: Remote-List Form (Reentrant)

In this example, the remote-list form of MODCB is used to build a parameter list that will later be used to modify the MACRF bits in the access-method control block ANYACB.

```
LA       8,LEN2               Get length of the parameter list.
GETMAIN  R,LV=(8)             Get storage for the area in which the
                              parameter list is to be built.
LR       3,1                  Save address of the parameter-list area.
MODCB    ACB=ANYACB,
         MACRMF=(L,(3),LEN2)
```

The macro expansion equates the length of the parameter list to LEN2, as follows:

```
+LEN2 EQU 24
```

This parameter list is built in the remote area pointed to by register 3. The list will be used by VSAM to modify the ACB when an execute form of MODCB is issued (see next example). The list form only creates a parameter list; it does not modify the ACB.

Example: Execute Form (Reentrant)

In this example, the execute form of MODCB is used to modify the address of the access-method control block and MACRF codes in the parameter list created by the remote-list form of MODCB in the previous example.

```
MODCB    ACB=MYACB,MACRF=(ADR,SEQ,OUT),MF=(E,(3))
```

The parameter list pointed to by register 3 is changed so that the ACB and MACRF parameter values in the execute form override the ones in the list form. The access-method control block MYACB is then modified to MACRF=ADR,SEQ,OUT). The access-method control block at ANYACB is not changed by either of these examples.

APPENDIX C: OPERAND NOTATION FOR GENCB, MODCB, SHOWCB, AND TESTCB

The addresses, names, numbers, and options required with operands in GENCB, MODCB, SHOWCB, and TESTCB can be expressed in a variety of ways:

- An absolute numeric expression, for example, STRNO=3 and COPIES=10
- A character string, for example, DDNAME=DATASET
- A code or a list of codes separated by commas and enclosed in parentheses, for example, OPTCD=KEY or OPTCD=(KEY,DIR,IN)
- An expression valid for a relocatable A-type address constant, for example, AREA=MYAREA+4
- A register from 2 through 12 that contains an address or numeric value, for example, SYNAD=(3); equated labels can be used to designate a register, for example, SYNAD=(ERR), where the following equate statement has been included in the program: ERR EQU 3
- An expression of the form (S,scon), where scon is an expression valid for an S-type address constant, including the base-displacement form. The contents of the base register will be added to the displacement to obtain the value of the keyword. For example, if the value of the keyword being represented is a numeric value (that is, COPIES, LENGTH, RECLEN), the contents of the base register will be added to the displacement to determine the numeric value. If the value of the keyword being represented is an address constant (that is, WAREA, EXLST, EODAD, ACB), the contents of the base register will be added to the displacement to determine the value of the address constant.
- An expression of the form (*,scon), where scon is an expression valid for an S-type address constant, including the base-displacement form; the address specified by scon is indirect, that is, it is the address of an area that contains the value of the keyword. The contents of the base register will be added to the displacement to determine the address of the fullword of storage that contains the value of the keyword.

If an indirect S-type address constant is used, the value it points to must meet the following criteria:

- If it is a numeric quantity or an address, it must occupy a fullword of storage.
- If it is an alphameric character string, it must occupy two words of storage, be left aligned, and be filled on the right with blanks.

The expressions that can be used depend on the keyword specified. Additionally, register and S-type address constants cannot be used when MF=L is specified. See "Appendix B: List, Execute, and Generate forms of GENCB, MODCB, SHOWCB, and TESTCB."

Operands with GENCB

Figure 27 shows the expressions that can be used in the GENCB macro.

	Absolute Numeric	Code	Character String	Register	S-Type Address	Indirect S-Type Address	A-Type Address
GENCB Keywords							
AM		X					
BLK		X					
COPIES	X			X	X	X	
LENGTH	X			X	X	X	
WAREA				X	X	X	X
ACB Keywords (BLK=ACB)							
BSTRNO	X			X	X	X	
BUFND	X			X	X	X	
BUFNI	X			X	X	X	
BUFSP	X			X	X	X	
CATALOG		X					
CRA		X					
DDNAME			X			X	
EXLST				X	X	X	X
MACRF		X					
MAREA				X	X	X	X
MLEN				X	X	X	
PASSWD				X	X	X	X
STRNO	X			X	X	X	
EXLST Keywords (BLK=EXLST)							
EODAD				X	X	X	X
JRNAD				X	X	X	X
LERAD				X	X	X	X
SYNAD				X	X	X	X
A		X					
N		X					
L		X					
RPL Keywords (BLK=RPL)							
ACB				X	X	X	X
AREA				X	X	X	X
AREALEN	X			X	X	X	
ARG				X	X	X	X
ECB				X	X	X	X
KEYLEN	X			X	X	X	
MSGAREA				X	X	X	X
MSGLEN	X			X	X	X	
NXTRPL				X	X	X	X
OPTCD		X					
RECLEN	X			X	X	X	
TRANSID	X			X	X	X	

Figure 27. GENCB Operands

Operands with MODCB

Figure 28 shows the expressions that can be used in the MODCB macro.

	Absolute Numeric	Code	Character String	Register	S-Type Address	Indirect S-Type Address	A-Type Address
MODCB Keyword							
{ACB | EXLST | RPL}				X	X	X	X
ACB Keywords							
BSTRNO	X			X	X	X	
BUFND	X			X	X	X	
BUFNI	X			X	X	X	
BUFSP	X			X	X	X	
CATALOG		X					
CRA		X					
DDNAME			X			X	
EXLST				X	X	X	X
MACRF		X					
MAREA				X	X	X	X
MLEN	X			X	X	X	
PASSWD				X	X	X	X
STRNO	X			X	X	X	
EXLST Keywords							
EODAD				X	X	X	X
JRNAD				X	X	X	X
LERAD				X	X	X	X
SYNAD				X	X	X	X
A	X						
N	X						
L	X						
RPL Keywords							
ACB				X	X	X	X
AREA				X	X	X	X
AREALEN	X			X	X	X	
ARG				X	X	X	X
ECB				X	X	X	X
KEYLEN	X			X	X	X	
MSGAREA				X	X	X	X
MSGLEN	X			X	X	X	
NXTRPL				X	X	X	X
OPTCD		X					
RECLEN	X			X	X	X	
TRANSID	X			X	X	X	

Figure 28. MODCB Operands

Operands with SHOWCB

Figure 29 shows the expressions that can be used in the SHOWCB macro.

	Absolute Numeric	Code	Character String	Register	S-Type Address	Indirect S-Type Address	A-Type Address
SHOWCB Keywords							
{ACB | EXLST | RPL}				X	X	X	X
AREA				X	X	X	X
FIELDS		X					
LENGTH	X			X	X	X	
OBJECT		X					

Figure 29. SHOWCB Operands

Operands with TESTCB

Figure 30 shows the expressions that can be used in the TESTCB macro.

	Absolute Numeric	Code	Character String	Register	S-Type Address	Indirect S-Type Address	A-Type Address
TESTCB Keywords							
{ACB | EXLST | RPL}				X	X	X	X
ERET				X	X	X	X
OBJECT		X					
ACB Keywords							
ACBLEN	X			X	X	X	
ATRB		X					
AVSPAC	X			X	X	X	
BSTRNO	X			X	X	X	
BUFND	X			X	X	X	
BUFNI	X			X	X	X	
BUFNO	X			X	X	X	
BUFSP	X			X	X	X	
CATALOG		X					
CRA		X					
CINV	X			X	X	X	
DDNAME			X			X	
ENDRBA	X			X	X	X	
ERROR	X			X	X	X	
EXLST				X	X	X	X
FS	X			X	X	X	
KEYLEN	X			X	X	X	
LRECL	X			X	X	X	
MACRF		X					
MAREA				X	X	X	X
MLEN	X			X	X	X	
NCIS	X			X	X	X	
NDELR	X			X	X	X	
NEXCP	X			X	X	X	
NEXT	X			X	X	X	
NINSR	X			X	X	X	
NIXL	X			X	X	X	
NLOGR	X			X	X	X	
NRETR	X			X	X	X	
NSSS	X			X	X	X	
NUPDR	X			X	X	X	
OFLAGS		X					
OPENOBJ		X					
PASSWD				X	X	X	X
RKP	X			X	X	X	
STMST						X	
STRNO	X			X	X	X	
EXLST Keywords							
EODAD				X	X	X	X
EXLLEN	X			X	X	X	
JRNAD				X	X	X	X
LERAD				X	X	X	X
SYNAD				X	X	X	X
A		X					
N		X					
L		X					

Figure 30 (Part 1 of 2). TESTCB Operands

	Absolute Numeric	Code	Character String	Register	S-Type Address	Indirect S-Type Address	A-Type Address
RPL Keywords							
ACB				X	X	X	X
AIXFLAG		X					
AIXPC	X			X	X	X	
AREA				X	X	X	X
AREALEN	X			X	X	X	
ARG				X	X	X	X
ECB				X	X	X	X
FDBK	X	X		X	X	X	
FTNCD		X					
IO		X					
KEYLEN	X			X	X	X	
MSGAREA				X	X	X	X
MSGLEN	X			X	X	X	
NXTRPL				X	X	X	X
OPTCD		X					
RBA	X			X	X	X	
RECLEN	X			X	X	X	
RPLLEN	X			X	X	X	
TRANSID	X			X	X	X	

Figure 30 (Part 2 of 2). TESTCB Operands

GLOSSARY

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 Data Processing Glossary,* 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 catalogs to OS/VS2 catalogs.

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, key field, and prime key.)

alternate index cluster: The data and index components of an alternate index.

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.

base cluster: A key-sequenced or entry-sequenced data set over which one or more alternate indexes are built.

catalog: (*See* master catalog and user catalog.)

catalog recovery area: (*See* CRA.)

chained RPL: (*see* RPL string.)

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 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.

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.

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.

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.)

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.

entry sequence: The order in which data records are physically arranged (according to ascending RBA) in auxiliary storage, without respect to their contents. (Contrast to 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.

field: In a record or a control block, a specified area used for a particular category of data or control information.

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.

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 record: A collection of index entries that are retrieved and stored as a group. (Contrast to data 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.

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.

job catalog: A catalog made available for a job by means of the JOBCAT 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 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.)

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.

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 of 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.

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.)

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.

record: (*See* index record, data record, stored record.)

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.)

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.

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.

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.)

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.

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.

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 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.

INDEX

For additional information about any subject listed in this index, refer to the publications that are listed under the same subject in either *OS/VS1 Master Index*, GC24-5104, or *OS/VS2 Master Index*, GC28-0693.

This index makes no page references to the glossary.

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

1 2 3

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

OS/VS Virtual Storage Access Method
(VSAM) Programmer's Guide
GC26-3838-3

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. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM shall have the nonexclusive right, in its discretion, to use and distribute all submitted information, in any form, for any and all purposes, without obligation of any kind to the submitter. Your interest is appreciated.

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.

Fold on two lines, tape, and mail. No postage necessary if mailed in the U.S.A. (Elsewhere, any IBM representative will be happy to forward your comments.) Thank you for your cooperation.

GC26-3838-3

Reader's Comment Form

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

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

OS/VS Virtual Storage Access Method (VSAM) Programmer's Guide (File No. S370-30) Printed in U.S.A. GC26-3838-3