SY26-3825-1
File No. S370-30

Systems

OS/VS2 Virtual Storage Access Method (VSAM) Logic

Release 3.7

IBM

Second Edition (January 1976)

This edition replaces the previous edition (numbered SY26-3825-0) and its technical newsletter (numbered SN26-0782) and makes them both obsolete. This edition also makes obsolete *OS/VS2 Independent Component: Virtual Storage Access Method (VSAM) Logic,* SY26-3846-0, and its technical newsletters (numbered SN26-0803, SN26-0796).

This edition applies to Release 3.7 of OS/VS2 and to all subsequent releases of that system unless otherwise indicated in new editions or technical newsletters.

Significant system changes are summarized under "Summary of Amendments" following the list of illustrations. In addition, miscellaneous editorial and technical changes have been made throughout the publication. Because the technical changes in this edition are extensive and difficult to localize, they are not marked by vertical lines in the left margin; the entire edition should be reviewed carefully.

Information in this publication is subject to significant change. Any such changes will be published in new editions or technical newsletters. Before using the publication, consult the latest *IBM System/370 Bibliography,* GC20-0001, and the technical newsletters that amend the bibliography, to learn which editions and technical newsletters are applicable and current.

Requests for copies of IBM publications should be made to the IBM branch office that serves you.

Forms for readers' comments are provided at the back of the publication. If the forms have been removed, comments may be addressed to IBM Corporation, General Products Division, Programming Publishing—Department J57, 1501 California Avenue, Palo Alto, California 94304. All comments and suggestions become the property of IBM.

PREFACE

This book describes the internal logic of the Virtual Storage Access Method (VSAM) and contains diagnostic information. It is directed to maintenance personnel and development programmers who require an in-depth knowledge of VSAM's design, organization, and data areas.

Organization of This Book

This book has the following major divisions:

- "Introduction," which describes the use of VSAM, how VSAM fits into the operating system, how VSAM interacts with the operating system and the user's program, and the major components of VSAM.
- "Method of Operation," which describes the functions performed by VSAM.
- "Program Organization," which describes the information contained in VSAM program listings and the flow of control between modules.
- "Directory," which lists VSAM modules and the method of operation diagrams related to each module.
- "Data Areas," which describes control blocks used by VSAM and describes the format of VSAM data and index records.
- "Diagnostic Aids," which contains useful information for locating the cause of problems in the VSAM procedures.
- "Glossary," which defines terms relevant to VSAM, and lists abbreviations and acronyms used in this book and in the VSAM program listings.
- "Index," which is a subject index to the book.

Required Reading

The following book should be read and understood before using this one:

- *OS/VS Virtual Storage Access Method (VSAM) Programmer's Guide,* GC26-3838, which introduces VSAM concepts and contains definitive explanations of VSAM macros.

Related IBM Publications

- *Introduction to the IBM 3850 Mass Storage System (MSS),* GA32-0028
- *OS/VS Data Management Macro Instructions,* GC26-3793
- *OS/VS Mass Storage System (MSS) Planning Guide,* GC35-0011
- *OS/VS Message Library: VS2 System Messages,* GC38-1002
- *OS/VS Virtual Storage Access Method (VSAM) Options for Advanced Applications,* GC26-3819
- *OS/VS2 Access Method Services,* GC26-3841
- *OS/VS2 Catalog Management Cross Reference,* SYB6-3843
- *OS/VS2 Catalog Management Logic,* SY26-3826
- *OS/VS2 Checkpoint/Restart Logic,* SY26-3820
- *OS/VS2 DADSM Logic,* SY26-3828
- *OS/VS2 Data Areas,* SYB8-0606
- *OS/VS2 I/O Supervisor Logic,* SY26-3823
- *OS/VS2 JCL,* GC28-0692
- *OS/VS2 Open/Close/EOV Logic,* SY26-3827
- *OS/VS2 Supervisor Services and Macro Instructions,* GC28-0683
- *OS/VS2 System Logic Library, Volumes 1-7,* SY28-0713 through SY28-0719 (All seven volumes can be ordered as SBOF-8210.)
- *OS/VS2 System Programming Library: Debugging Handbook, Volume 1,* GC28-0708, and *Volume 2,* GC28-0709 (Both volumes can be ordered as GBOF-8211.)
- *OS/VS2 System Programming Library: Service Aids,* GC28-0674
- *OS/VS2 System Programming Library: System Management Facilities (SMF),* GC28-0706
- *OS/VS2 VSAM Cross Reference,* SYB6-3842

Using This Book

This book is designed to be used with the VSAM program listings in the microfiche for VSAM and with *OS/VS2 VSAM Cross Reference,* SYB6-3842, also on microfiche cards. Cross-reference reports are described in "Microfiche Cross-Reference Aids" in "Diagnostic Aids."

The diagrams in "Method of Operation" describe the major functions performed by VSAM; these diagrams are intended to be your key to a module name (and procedure name, as appropriate) in the listing. See "Reading Method of Operation Diagrams" in "Method of Operation" for a description of how to read these diagrams. For information on what is available in the program listings, see "Module Prologues" in "Program Organization."

CONTENTS

ILLUSTRATIONS

Figures

Diagrams

SUMMARY OF AMENDMENTS

Release 3.7

VSAM SNAP Dump Facility

To increase the serviceability of VSAM, the VSAM SNAP dump facility has been added to provide hexadecimal dumps of VSAM-owned control blocks in CSA. Included in the dump are:

- The JSCBSHR field of the JSCB (used by VSAM to locate the VAT)
- The control blocks for open VSAM data sets processed with the global shared resources (GSR) option
- The control blocks making up the GSR pool
- The VGTT chain for the ASCB associated with the TCB being dumped and any PSBs associated with these VGTTs

The dump facility is described in "Diagnostic Aids."

Control Block Manipulation Macros

Changes to support improved control block manipulation macro processing were made in

- Diagrams CA and CB
- "Data Areas," where KEYWDTAB, a branch table that controls execution of IDA019C1 and supports processing of the control block macros, is described
- "Diagnostic Aids," where a new return code, issued when a block to be displayed or tested does not exist because the data set is a dummy data set, has been added

Enhanced VSAM

VSAM has several new functions and data structues for the independent component release of VS2 Release 3: alternate indexes, Checkpoint/Restart processing, spanned records, relative record data sets, processing the index of a key-sequenced data set, shared resources among data sets, improved control-interval processing, backward sequential processing, catalog recovery, and virtual-storage management. These additions to VSAM change this logic manual in all its sections: method of operation diagrams (HIPOs), program organization figures (compendiums), directories, data areas, and diagnostic aids. The directories identify all the new modules and external procedures and indicate which HIPOs and compendiums refer to them.

Method of operation diagrams have been added for Data-Set Management to document recovery-termination processing.

The detailed descriptions of some control blocks, for which you were previously referred to *OS/VS2 Data Areas,* are included in this book.

The index has been expanded to include more proper names:

- Modules and external procedures are included with a page reference to the "Directory."

- Internal procedures and program instruction labels are included with page references to the pages where may appear.

Alternate Indexes

Alternate indexes for key-sequenced and entry-sequenced data sets add control blocks and complicate control block interrelationships. Opening and closing a path (a base cluster and the alternate index through which access is gained to it) more than double the number of HIPOs for Data-Set-Management. Access by way of a path and alternate-index upgrading change and add HIPOs to Record Management.

Checkpoint/Restart Processing

Checkpoint/Restart processing changes four HIPOs, and adds five HIPOs, two program organization figures, and four control blocks.

Spanned Records

Having data records longer than one control interval changes a number of HIPOs in Record Management. It changes the contents of control information in the RDFs in a control interval.

Relative Record Data Set

The relative record data set brings to three the number of types of VSAM data sets. It changes the contents of control information in the RDFs in a control interval. It changes HIPOs and adds a HIPO to Record Management.

Processing the Index of a Key-Sequenced Data Set

User access to the control intervals of a prime index changes HIPOs in Record Management to include the GETIX and PUTIX macros.

Shared Resources among Data Sets

Shared buffers, I/O-related control blocks, and channel programs among data sets for processing add control blocks and change control block interrelationships. Building and deleting a VSAM resource pool add a HIPO to Data-Set Management for the BLDVRP and DLVRP macros add a section to "Diagnostic Aids" to describe recovery with global shared resources. Managing I/O buffers adds HIPOs to Record Management for the MRKBFR, WRTBFR, and SCHBFR macros.

Improved Control-Interval Processing

Improved control-interval processing changes HIPOs in Record Management to show the bypassing of certain functions for faster processing.

Backward Sequential Processing

Backward sequential processing changes HIPOs in Record Management to include processing data records in descending sequence by RBA or key.

Catalog Recovery

The optional recovery function that enables users to recover or restore data sets changes HIPOs slightly in Data-Set Management and Record Management.

Virtual-Storage Management

The management of virtual storage has been centralized in Virtual-Storage Management, which controls most requests for storage. It adds control blocks, which are described in "Virtual-Storage Management" in "Diagnostic Aids."

Release 3

Staging and destaging of data between mass storage and direct-access storage is added for the IBM 3850 Mass Storage System.

Release 2

I/O Management

For communication with the VS2 I/O Supervisor, the IOB control block is replaced by a set of three control blocks: IOMB, IOSM, and SRB.

Interface Between VSAM Record Management and VS2 I/O Supervisor. The function of putting together a channel program for issuing STARTIO has been separated from Record Management to stand logically as an interface between Record Management and the VS2 I/O Supervisor.

Interface Between VS2 Auxiliary Storage Management and I/O Supervisor. VSAM I/O Management serves the same function for paging I/O between real storage and external page storage. It is the programming interface between the VS2 Auxiliary Storage Manager and I/O Supervisor.

Security and Integrity

OS/VS2 multiprocessing requires changes in the way serially reusable resources are shared. A scheme of software locks that programs must obtain and free in order to use certain resources replaces hardware disabling. The CS instruction (compare and swap) is also used to ensure the integrity of serially reusable resources. ENQ/DEQ and the TS instruction (test and set) are still used in OS/VS2, Release 2.

I/O Management, Data-Set Management (Open and Close, for both VSAM and the ISAM Interface), and End of Volume (considered logically as part of Record Management) use the local memory lock to protect VSAM control blocks when chaining them. They obtain or release the local memory lock with the SETLOCK macro. Data-Set Management also uses the CS instruction when chaining DEB control blocks or modifying UCB information.

Most of the processing of I/O Management is in supervisor mode. It uses the MODESET macro to swap storage-protection keys. Data-Set Management runs primarily in storage-protection key 0 (as does End of Volume). ISAM-Interface Open and Close run primarily in the user's key. These modules use the MODESET macro to swap keys when transferring control to and from OS/VS Open and Close. Record Management (with the exception of End of Volume) continues to run in the user's key, as in OS/VS1 and Release 1 of OS/VS2.

VSAM protects crucial I/O control blocks by placing them in protected subpools. A user of VSAM cannot modify these control blocks and cannot, therefore, interfere with the operation of the system. VSAM Open and Close work with copies of ACB control blocks to prevent a user from interfering with the system. ISAM-Interface Open and Close don't work with copies since they run in the user's storage-protection key and thus provide no system processing to be interfered with.

Recovery and Termination

OS/VS2's philosophy of recovery is to avoid reIPLing the system and to free up resources claimed by a failing task in order to be able to continue processing without contending with fragmented resources. The system must be able to reclaim actual resources and slots in control tables.

The modules of I/O Management have functional recovery routines that get control from VS2 Recovery Termination Manager when an error occurs in I/O Management. A recovery routine releases various resources (such as the local memory lock, if it has been obtained) and may cause information to be recorded in SYS1.DUMP (with the SDUMP macro) or in SYS1.LOGREC (with the SETRP macro).

Data-Set Management and End of Volume share a recovery routine that gets control from the VS2 I/O Support Recovery Routine (which is an ESTAE routine) when an error occurs during the processing of these modules, and they share a Task Close Executor that gets control from VS2 Task Close for task or memory termination.

The recovery routine causes information to be recorded in SYS1.DUMP and SYS1.LOGREC concerning the processing that preceded an error. Open, Close, and End of Volume have been altered to leave various audit information in the Open/Close/End-of-Volume Work Area for this purpose.

The Task Close Executor frees up storage in the system area.

ISAM-Interface Open and Close also have a recovery routine that runs under control of the ESTAE routine mentioned above. This routine frees up ISAM Interface work areas following errors from which recovery cannot be made.

Page-Space Preformatting

VSAM in Release 2 of OS/VS2 recognizes the special case of a page-space data set being opened for output. When it occurs, VSAM makes the calculations required for preformatting, then transfers control to the Control-Area Preformat routine of Record Management to preformat all of the control areas that comprise the page-space data set.

INTRODUCTION

Virtual Storage Access Method (VSAM) is an access method for use with OS/VS1 and OS/VS2. VSAM is used with direct-access storage to provide fast storage and retrieval of data.

VSAM's record format is different from that of other access methods. All VSAM records are stored in *control intervals.* A control interval is a continuous segment of auxiliary storage. The records are ordered according to values in a key field or according to when they were stored. With key-sequenced data sets, the user can gain access to a record by specifying its key or its relative byte address (RBA). With entry-sequenced data sets, the user can gain access to a record only by specifying its RBA. For additional information on VSAM records and how they are stored, see "Data Areas."

User programs that contain Indexed Sequential Access Method (ISAM) macros can be used to process records in a VSAM data set. The ISAM interface program that allows the use of ISAM macros builds the necessary VSAM control blocks when an OPEN macro is issued and ensures that VSAM control blocks are properly initialized when subsequent requests are made for reading or writing records.

Most of VSAM resides in the pageable link pack area in the common area of virtual storage. Figure 1 illustrates VSAM's relationship to OS/VS2, to the processing program, and to the data stored on a direct-access storage device and in mass storage. The subpools indicated in the figure (230, 231, 239, 241, 245, 250, 252) contain VSAM control blocks. For more information see "Virtual-Storage Management" in "Diagnostic Aids."

VSAM is controlled by user macros. These macros are expanded into calling sequences to VSAM functions. For additional information on user macros, see *OS/VS Virtual Storage Access Method (VSAM) Programmer's Guide* and *OS/VS Access Method Services.*

VSAM communicates with other parts of the operating system through the SVC processor and through VS2 control blocks used by VSAM. In addition to the VS2 control blocks used by VSAM, VSAM builds and uses the access-method control block (ACB). The ACB describes a VSAM data set in much the same way that a DCB describes a nonVSAM data set.

In addition to processing records and data sets, VSAM opens and closes data sets and does most of its own space management, that is, VSAM makes only minor use of VS2 Open and Close and relies on VS2 DADSM for only part of its space management. To do much of this work, VSAM uses the VS2 catalog. VS2 catalogs contain a description of VSAM space, where available space is, how space is used, and the location of data sets. For additional information on the catalog, see *OS/VS2 Catalog Management Logic.*

VSAM is logically grouped into the following functional areas:

- Data-Set Management (sometimes referred to in program documentation as "I/O Support"), which comprises Open and Close for VSAM and for the ISAM Interface, Virtual-Storage Management, and BLDVRP/DLVRP processing
  - Open connects a user's program to a VSAM data set and builds the control blocks required to permit the user to read from and write to the data set.

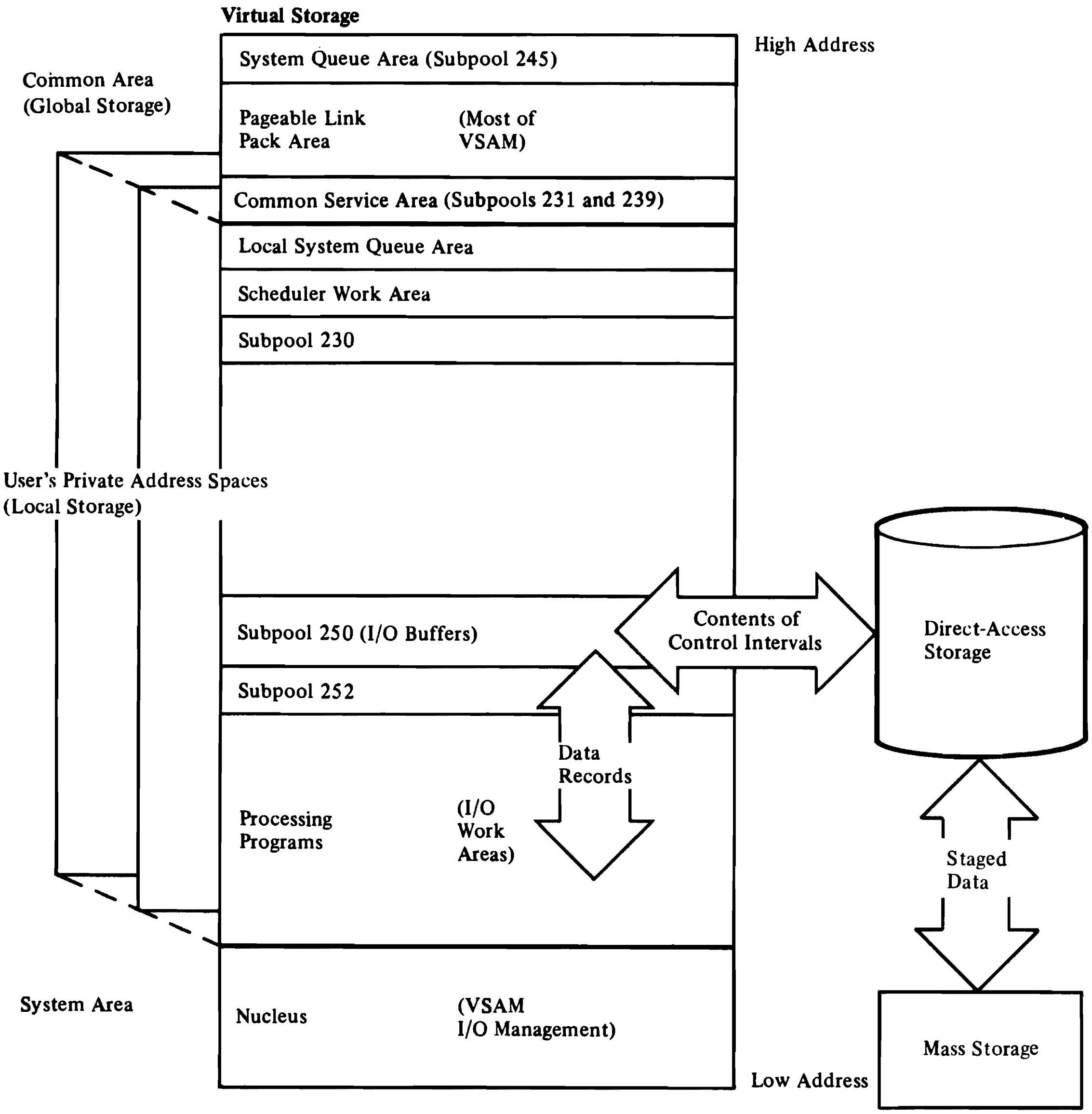


Figure 1. Relationship of VSAM, OS/VS2, User's Processing Program, and Stored Data

- Close disconnects a user's program from a data set and releases the data set's control blocks built by Open. Close also updates statistics in the catalog.
- Virtual-Storage Management centralizes the processing of most requests for virtual storage.
- BLDVRP/DLVRP processing builds and deletes VSAM resource pools for processing with local or global shared resources. (Processing with shared resources is described from the user's point of view in *OS/VS VSAM Options for Advanced Applications.*)

- Record Management, which comprises processing to satisfy user requests for access to data, including end-of-volume processing
  - Data-Request Processing requests I/O Management to read and write records in response to user-issued VSAM and ISAM macros (the latter by way of the ISAM Interface). It also requests I/O Management to read and write records for VS2 Catalog Management.

  - End of Volume mounts volumes and allocates space. It modifies the existing control blocks to reflect the newly mounted volumes and newly allocated space.

- Control Block Manipulation, which allows a user's program to generate some control blocks (ACB, EXLST, and RPL) dynamically and to modify, display; and test their contents

- I/O Management, which comprises the Problem-State I/O Driver, the Supervisor-State I/O Driver, the Actual Block Processor, end appendages, an asynchronous routine, and a purge routine

  - The drivers and the Actual Block Processor translate requests for access to the contents of control intervals to requests for reading and writing physical records. They build a channel program to give to the VS2 I/O Supervisor.

  - The appendages and the asynchronous routine get control back to the requester after I/O is finished.

METHOD OF OPERATION

Method of operation diagrams are functional descriptions of VSAM. The diagram and descriptive notes, keyed to the diagram, are on facing pages.

Reading Method of Operation Diagrams

The diagrams contain three blocks of information: input, processing, and output. The left-hand side of the diagram shows the data that serves as input to the processing steps in the center of the diagram, and the right-hand side shows the data that is output from the processing steps. Input is anything a program function refers to or gets. Processing is the steps required to fulfill the function represented by the diagram. Output is any change effected by a function; for example, register contents, or control blocks created or modified. The processing steps are numbered; the numbers correspond to notes on the facing page. The notes include cross-references to the listings. Figure 2 shows a method of operation figure.

The left-hand side of the diagram shows the input required by the function shown in the diagram. For example, register 1 points to a list of DCB pointers for an ISAM user. The SYS1.SYSJOBQE contains the JFCB, which indicates the data set's organization. The data-set information in the DCB is input to

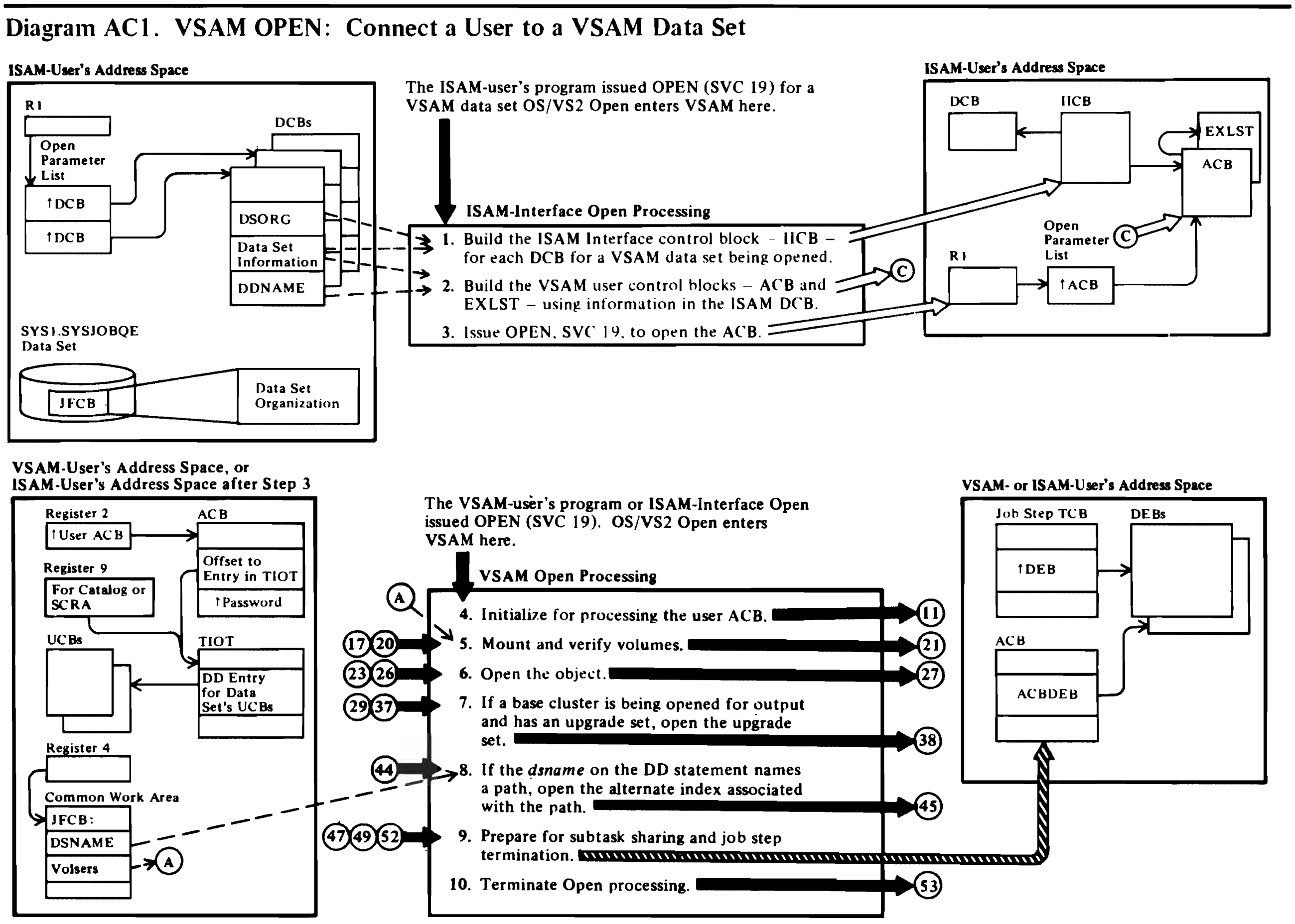


Diagram AC1. VSAM OPEN: Connect a User to a VSAM Data Set
ISAM-User's Address Space
R1
Open Parameter List
↑DCB
↑DCB
DCBs
DSORG
Data Set Information
DDNAME
SYS1.SYSJOBQE Data Set
JFCB
Data Set Organization
The ISAM-user's program issued OPEN (SVC 19) for a VSAM data set OS/VS2 Open enters VSAM here.
ISAM-Interface Open Processing
1. Build the ISAM Interface control block – IICB – for each DCB for a VSAM data set being opened.
2. Build the VSAM user control blocks – ACB and EXLST – using information in the ISAM DCB.
3. Issue OPEN, SVC 19, to open the ACB.
C
ISAM-User's Address Space
DCB
IICB
EXLST
ACB
Open Parameter List
C
R1
↑ACB
VSAM-User's Address Space, or ISAM-User's Address Space after Step 3
Register 2
↑User ACB
ACB
Offset to Entry in TIOT
↑Password
Register 9
For Catalog or SCRA
UCBs
TIOT
DD Entry for Data Set's UCBs
Register 4
Common Work Area
JFCB:
DSNAME
Volsers
A
The VSAM-user's program or ISAM-Interface Open issued OPEN (SVC 19). OS/VS2 Open enters VSAM here.
VSAM Open Processing
A
4. Initialize for processing the user ACB.
11
17 20
5. Mount and verify volumes.
21
23 26
6. Open the object.
27
29 37
7. If a base cluster is being opened for output and has an upgrade set, open the upgrade set.
38
44
8. If the dsname on the DD statement names a path, open the alternate index associated with the path.
45
47 49 52
9. Prepare for subtask sharing and job step termination.
10. Terminate Open processing.
53
VSAM- or ISAM-User's Address Space
Job Step TCB
↑DEB
DEBs
ACB
ACBDEB

Figure 2. Method of Operation Diagram

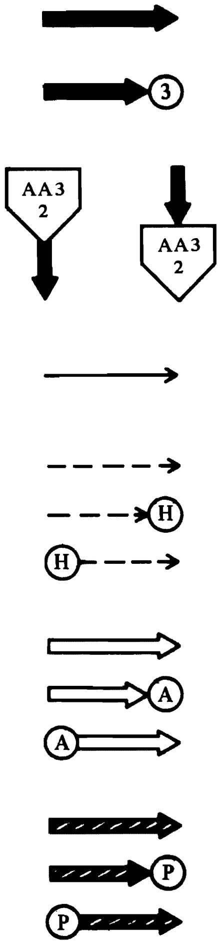
steps 1 and 2 in the processing portion of the diagram. The DDNAME is input to step 2 in the processing portion of the diagram.

The processing portion of the diagram shows the processing steps required to fulfill the function described by the diagram. Note that the function described by one diagram might be performed by one or more VSAM modules; that is, the diagrams describe functions, not physical parts of the program.

The figure shows two conditions for which VSAM Open is called: (1) at step 1 when processing is to be done for an ISAM user program and (2) at step 4 when processing is to be done for a VSAM user program or for an ISAM user program that has been processed by steps 1 through 3. The numbers 1, 2, 3, 4, and 5 are keys to the notes for this diagram.

The output created by each processing step is shown in the diagram. Step 1, for example, builds a control block (the IICB); step 2 builds VSAM user control blocks (the ACB and EXLST).

Reading the method of operation diagrams requires that you understand the symbols they use. Figure 3 shows the symbols and describes their meaning.



Flow of control on the same diagram; '3' indicates a number of a process step on the same diagram.

Flow of control between diagrams; 'AA3' is the diagram number and '2' is the number of a process step on that diagram.

Pointers

Reference to data or testing of data by a process step; 'H' is an arbitrary designation.

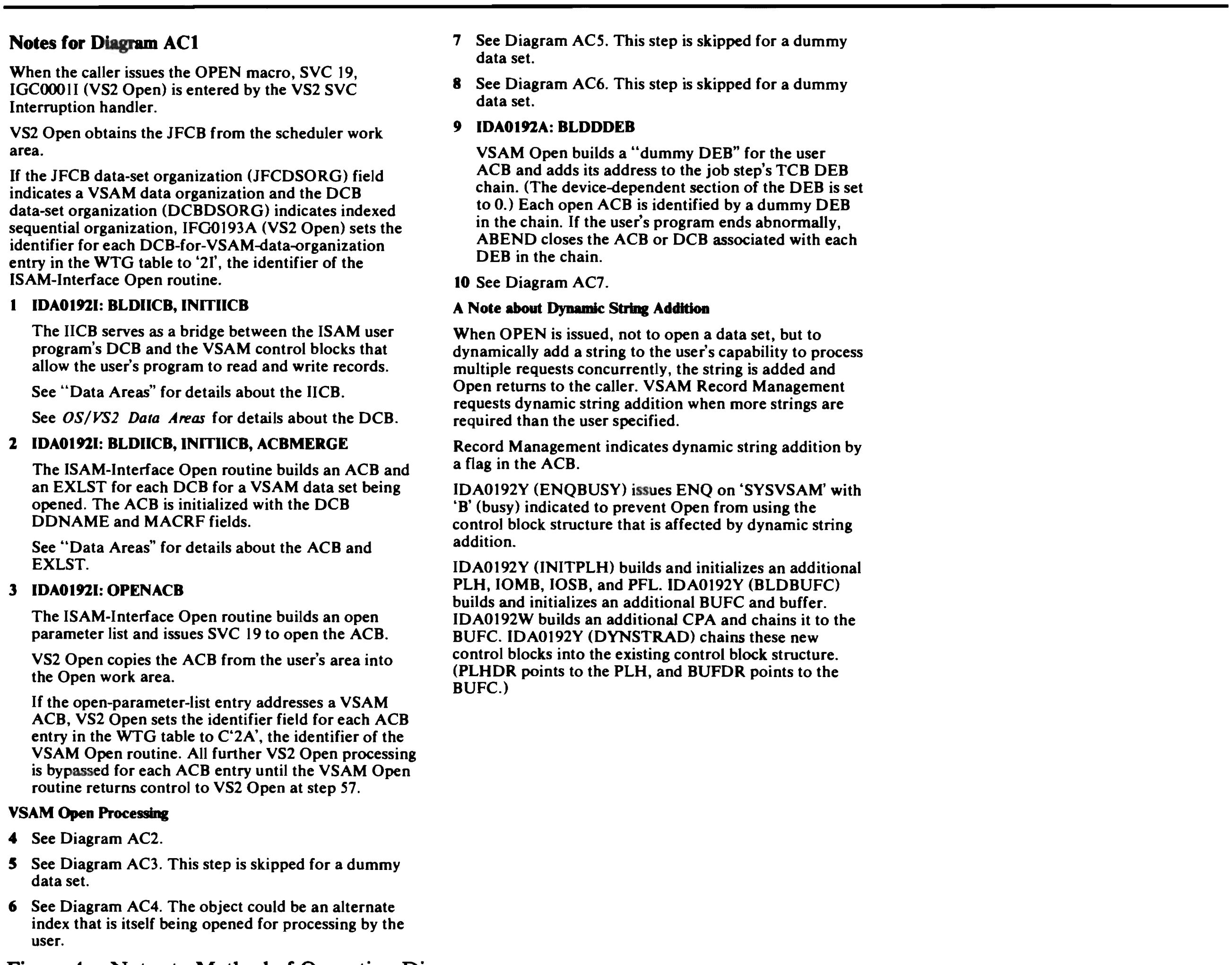
Input to process steps and output from process steps; 'A' is an arbitrary designation.

Modification of data by a process step; 'P' is an arbitrary designation.

Figure 3. Graphic Symbols Used in Method of Operations Diagrams

Figure 4 shows part of the notes to Figure 3.

The notes provide details about the processing shown in the diagram. For example, the entry process and conditions are described by the first (unnumbered) note. This note tells which OS/VS Open modules allow an ISAM user's program to open an ACB for a VSAM data set; note 1 describes the use of the IICB and directs you to "Data Areas" in this publication for detailed information on the IICB. The notes also name the modules and routines that perform the functions represented. The module and procedure names allow you to relate a process step to a unit of code in the VSAM program listings.



Notes for Diagram AC1

When the caller issues the OPEN macro, SVC 19, IGC0001I (VS2 Open) is entered by the VS2 SVC Interruption handler.

VS2 Open obtains the JFCB from the scheduler work area.

If the JFCB data-set organization (JFCDSORG) field indicates a VSAM data organization and the DCB data-set organization (DCBDSORG) indicates indexed sequential organization, IFG0193A (VS2 Open) sets the identifier for each DCB-for-VSAM-data-organization entry in the WTG table to '2I', the identifier of the ISAM-Interface Open routine.

1 **IDA0192I: BLDIICB, INITIICB**

The IICB serves as a bridge between the ISAM user program's DCB and the VSAM control blocks that allow the user's program to read and write records.

See "Data Areas" for details about the IICB.

See *OS/VS2 Data Areas* for details about the DCB.

2 **IDA0192I: BLDIICB, INITIICB, ACBMERGE**

The ISAM-Interface Open routine builds an ACB and an EXLST for each DCB for a VSAM data set being opened. The ACB is initialized with the DCB DDNAME and MACRF fields.

See "Data Areas" for details about the ACB and EXLST.

3 **IDA0192I: OPENACB**

The ISAM-Interface Open routine builds an open parameter list and issues SVC 19 to open the ACB.

VS2 Open copies the ACB from the user's area into the Open work area.

If the open-parameter-list entry addresses a VSAM ACB, VS2 Open sets the identifier field for each ACB entry in the WTG table to C'2A', the identifier of the VSAM Open routine. All further VS2 Open processing is bypassed for each ACB entry until the VSAM Open routine returns control to VS2 Open at step 57.

VSAM Open Processing

4 See Diagram AC2.

5 See Diagram AC3. This step is skipped for a dummy data set.

6 See Diagram AC4. The object could be an alternate index that is itself being opened for processing by the user.

7 See Diagram AC5. This step is skipped for a dummy data set.

8 See Diagram AC6. This step is skipped for a dummy data set.

9 **IDA0192A: BLDDDEB**

VSAM Open builds a "dummy DEB" for the user ACB and adds its address to the job step's TCB DEB chain. (The device-dependent section of the DEB is set to 0.) Each open ACB is identified by a dummy DEB in the chain. If the user's program ends abnormally, ABEND closes the ACB or DCB associated with each DEB in the chain.

10 See Diagram AC7.

A Note about Dynamic String Addition

When OPEN is issued, not to open a data set, but to dynamically add a string to the user's capability to process multiple requests concurrently, the string is added and Open returns to the caller. VSAM Record Management requests dynamic string addition when more strings are required than the user specified.

Record Management indicates dynamic string addition by a flag in the ACB.

IDA0192Y (ENQBUSY) issues ENQ on 'SYSVSAM' with 'B' (busy) indicated to prevent Open from using the control block structure that is affected by dynamic string addition.

IDA0192Y (INITPLH) builds and initializes an additional PLH, IOMB, IOSB, and PFL. IDA0192Y (BLDBUFC) builds and initializes an additional BUFC and buffer. IDA0192W builds an additional CPA and chains it to the BUFC. IDA0192Y (DYNSTRAD) chains these new control blocks into the existing control block structure. (PLHDR points to the PLH, and BUFDR points to the BUFC.)

Figure 4. Notes to Method of Operation Diagram

Diagram AA. Method of Operation Contents

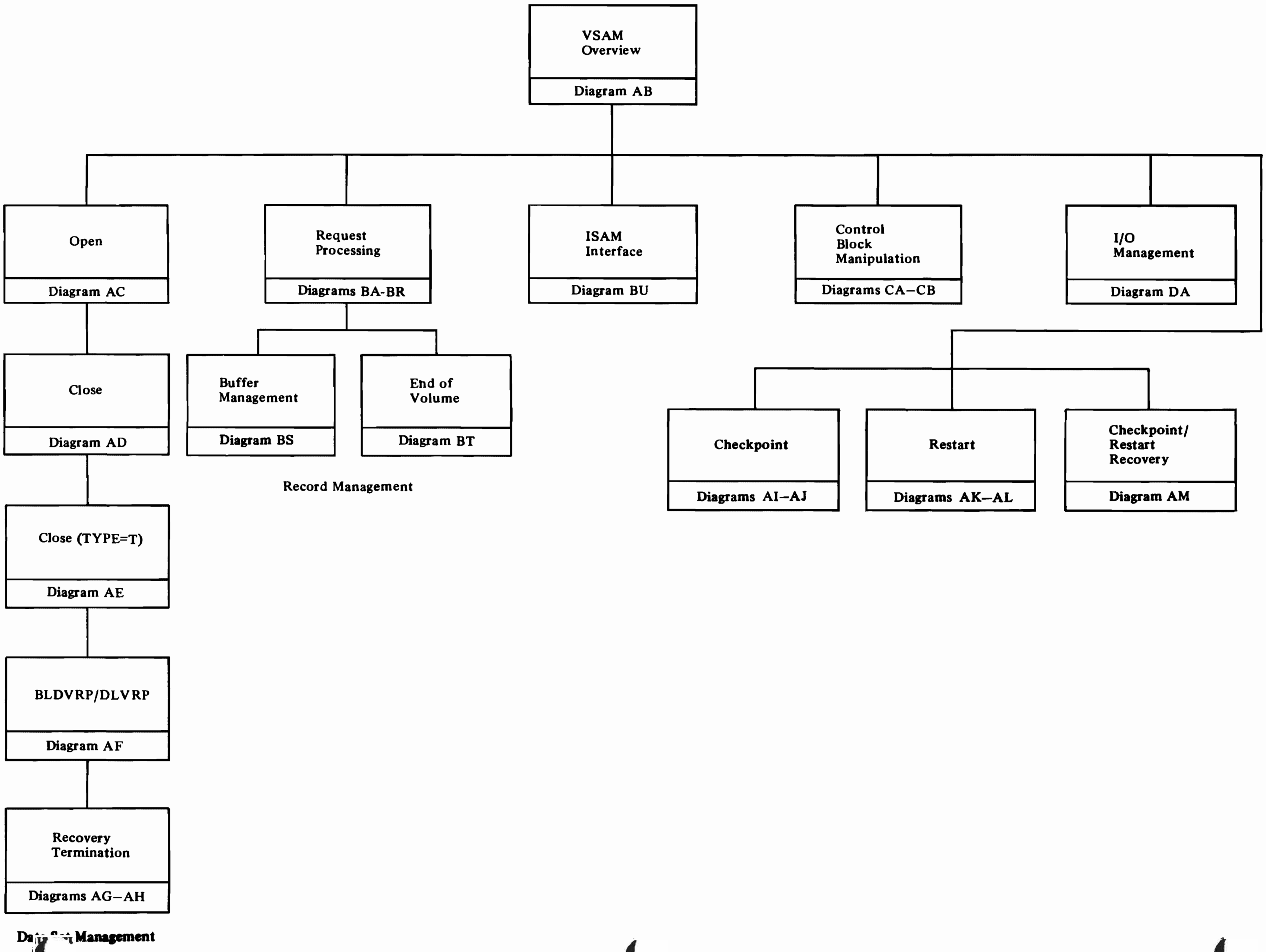


Diagram AB. VSAM Overview

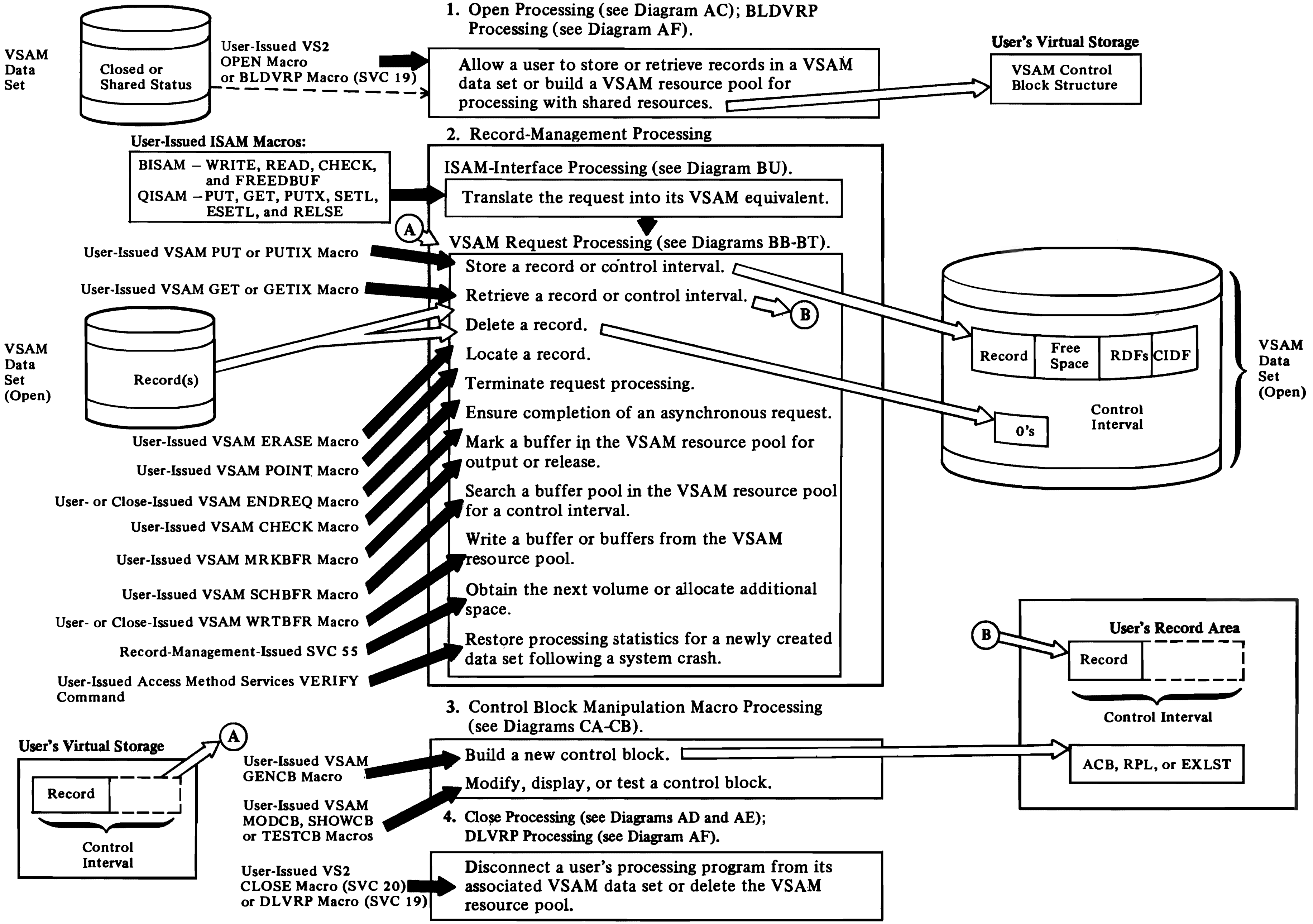
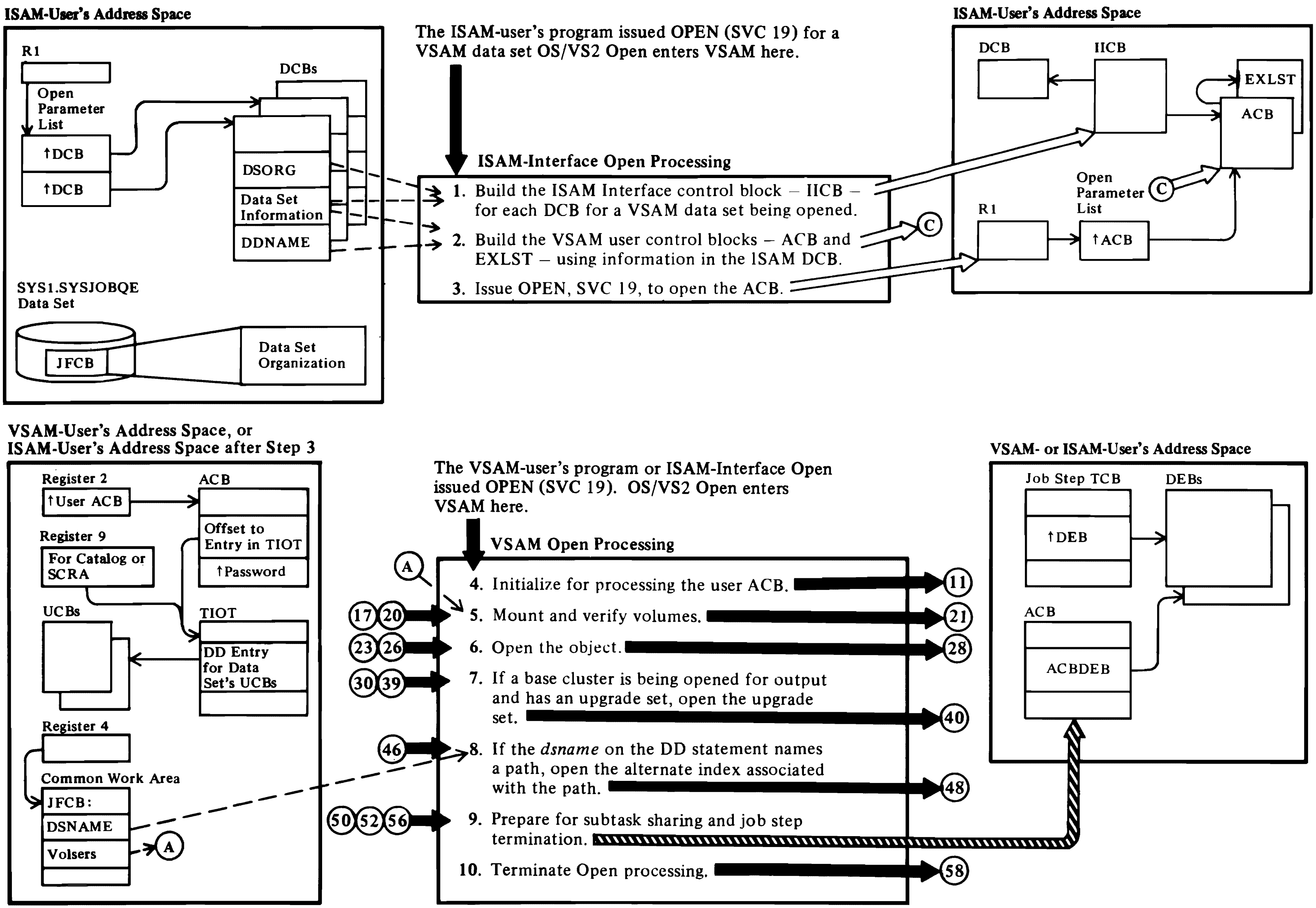

VSAM Data Set
Closed or Shared Status
User-Issued VS2 OPEN Macro or BLDVRP Macro (SVC 19)
1. Open Processing (see Diagram AC); BLDVRP Processing (see Diagram AF).
Allow a user to store or retrieve records in a VSAM data set or build a VSAM resource pool for processing with shared resources.
User's Virtual Storage
VSAM Control Block Structure
User-Issued ISAM Macros:
BISAM – WRITE, READ, CHECK, and FREEDBUF
QISAM – PUT, GET, PUTX, SETL, ESETL, and RELSE
2. Record-Management Processing
ISAM-Interface Processing (see Diagram BU).
Translate the request into its VSAM equivalent.
A
VSAM Request Processing (see Diagrams BB-BT).
User-Issued VSAM PUT or PUTIX Macro
User-Issued VSAM GET or GETIX Macro
VSAM Data Set (Open)
Record(s)
User-Issued VSAM ERASE Macro
User-Issued VSAM POINT Macro
User- or Close-Issued VSAM ENDREQ Macro
User-Issued VSAM CHECK Macro
User-Issued VSAM MRKBFR Macro
User-Issued VSAM SCHBFR Macro
User- or Close-Issued VSAM WRTBFR Macro
Record-Management-Issued SVC 55
User-Issued Access Method Services VERIFY Command
Store a record or control interval.
Retrieve a record or control interval.
B
Delete a record.
Locate a record.
Terminate request processing.
Ensure completion of an asynchronous request.
Mark a buffer in the VSAM resource pool for output or release.
Search a buffer pool in the VSAM resource pool for a control interval.
Write a buffer or buffers from the VSAM resource pool.
Obtain the next volume or allocate additional space.
Restore processing statistics for a newly created data set following a system crash.
Record
Free Space
RDFs
CIDF
0's
Control Interval
VSAM Data Set (Open)
B
User's Record Area
Record
Control Interval
ACB, RPL, or EXLST
User's Virtual Storage
A
Record
Control Interval
3. Control Block Manipulation Macro Processing (see Diagrams CA-CB).
User-Issued VSAM GENCB Macro
User-Issued VSAM MODCB, SHOWCB or TESTCB Macros
Build a new control block.
Modify, display, or test a control block.
4. Close Processing (see Diagrams AD and AE); DLVRP Processing (see Diagram AF).
User-Issued VS2 CLOSE Macro (SVC 20) or DLVRP Macro (SVC 19)
Disconnect a user's processing program from its associated VSAM data set or delete the VSAM resource pool.

Diagram AC1. VSAM OPEN: Connect a User to a VSAM Data Set



The ISAM-user's program issued OPEN (SVC 19) for a VSAM data set OS/VS2 Open enters VSAM here.

ISAM-Interface Open Processing

1. Build the ISAM Interface control block – IICB – for each DCB for a VSAM data set being opened.
2. Build the VSAM user control blocks – ACB and EXLST – using information in the ISAM DCB.
3. Issue OPEN, SVC 19, to open the ACB.

The VSAM-user's program or ISAM-Interface Open issued OPEN (SVC 19). OS/VS2 Open enters VSAM here.

VSAM Open Processing

4. Initialize for processing the user ACB.
5. Mount and verify volumes.
6. Open the object.
7. If a base cluster is being opened for output and has an upgrade set, open the upgrade set.
8. If the *dsname* on the DD statement names a path, open the alternate index associated with the path.
9. Prepare for subtask sharing and job step termination.
10. Terminate Open processing.

Notes for Diagram AC1

When the caller issues the OPEN macro, SVC 19, IGC0001I (VS2 Open) is entered by the VS2 SVC Interruption handler.

VS2 Open obtains the JFCB from the scheduler work area.

If the JFCB data-set organization (JFCDSORG) field indicates a VSAM data organization and the DCB data-set organization (DCBDSORG) indicates indexed sequential organization, IFG0193A (VS2 Open) sets the identifier for each DCB-for-VSAM-data-organization entry in the WTG table to '2I', the identifier of the ISAM-Interface Open routine.

1 **IDA0192I: BLDIICB, INITIICB**

The IICB serves as a bridge between the ISAM user program's DCB and the VSAM control blocks that allow the user's program to read and write records.

See "Data Areas" for details about the IICB.

See *OS/VS2 Data Areas* for details about the DCB.

2 **IDA0192I: BLDIICB, INITIICB, ACBMERGE**

The ISAM-Interface Open routine builds an ACB and an EXLST for each DCB for a VSAM data set being opened. The ACB is initialized with the DCB DDNAME and MACRF fields.

See "Data Areas" for details about the ACB and EXLST.

3 **IDA0192I: OPENACB**

The ISAM-Interface Open routine builds an open parameter list and issues SVC 19 to open the ACB.

VS2 Open copies the ACB from the user's area into the Open work area.

If the open-parameter-list entry addresses a VSAM ACB, VS2 Open sets the identifier field for each ACB entry in the WTG table to C'2A', the identifier of the VSAM Open routine. All further VS2 Open processing is bypassed for each ACB entry until the VSAM Open routine returns control to VS2 Open at step 57.

VSAM Open Processing

4 See Diagram AC2.

5 See Diagram AC3. This step is skipped for a dummy data set.

6 See Diagram AC4. The object could be an alternate index that is itself being opened for processing by the user.

7 See Diagram AC5. This step is skipped for a dummy data set.

8 See Diagram AC6. This step is skipped for a dummy data set.

9 **IDA0192A: BLDDDEB**

VSAM Open builds a "dummy DEB" for the user ACB and adds its address to the job step's TCB DEB chain. (The device-dependent section of the DEB is set to 0.) Each open ACB is identified by a dummy DEB in the chain. If the user's program ends abnormally, ABEND closes the ACB or DCB associated with each DEB in the chain.

10 See Diagram AC7.

A Note about Dynamic String Addition

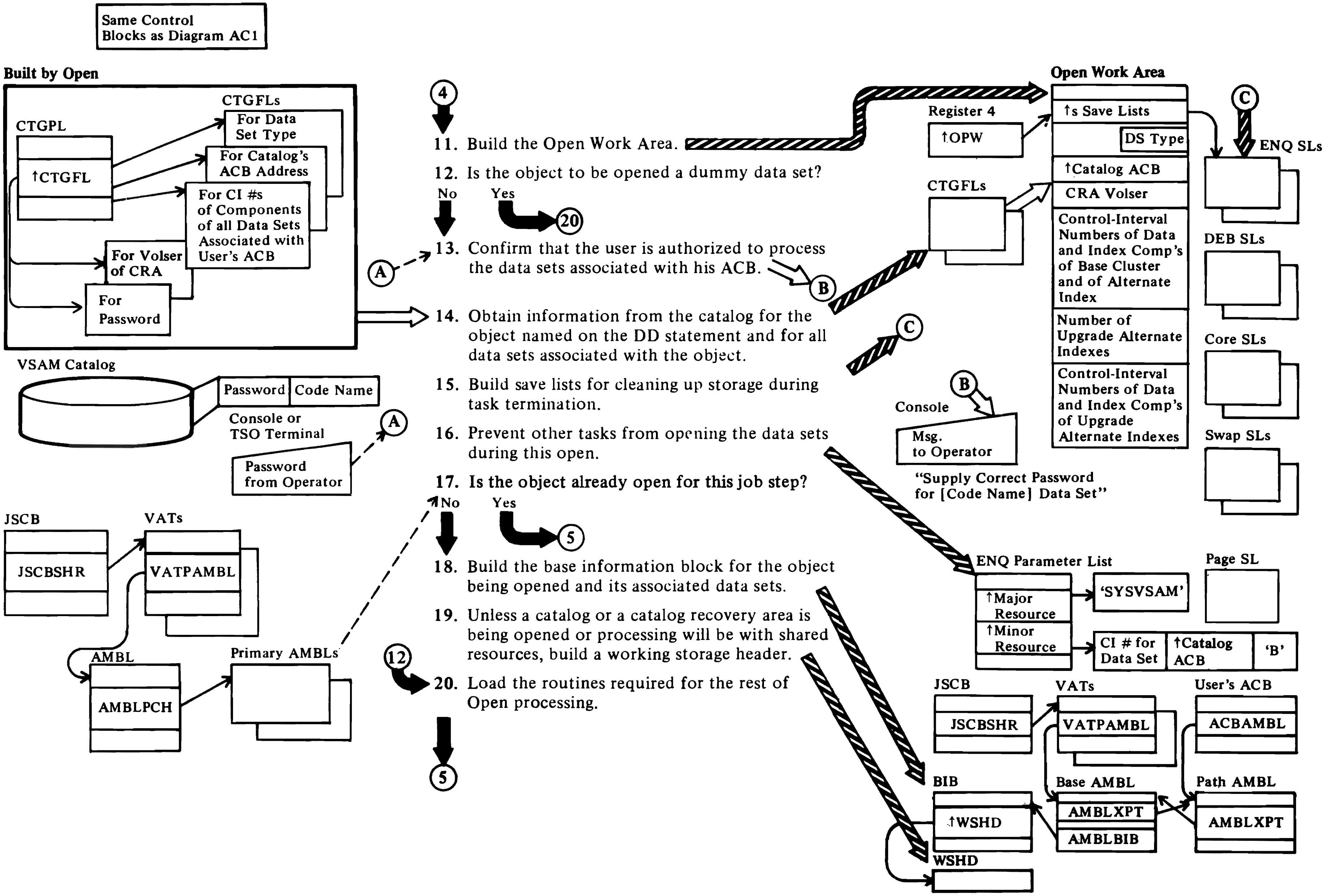
When OPEN is issued, not to open a data set, but to dynamically add a string to the user's capability to process multiple requests concurrently, the string is added and Open returns to the caller. VSAM Record Management requests dynamic string addition when more strings are required than the user specified.

Record Management indicates dynamic string addition by a flag in the ACB.

IDA0192Y (ENQBUSY) issues ENQ on 'SYSVSAM' with 'B' (busy) indicated to prevent Open from using the control block structure that is affected by dynamic string addition.

IDA0192Y (INITPLH) builds and initializes an additional PLH, IOMB, IOSB, and PFL. IDA0192Y (BLDBUFC) builds and initializes an additional BUFC and buffer. IDA0192W builds an additional CPA and chains it to the BUFC. IDA0192Y (DYNSTRAD) chains these new control blocks into the existing control block structure. (PLHDR points to the PLH, and BUFDR points to the BUFC.)

Diagram AC2. VSAM OPEN: Initialize for Processing the User ACB



11. Build the Open Work Area.

12. Is the object to be opened a dummy data set?

No / Yes → 20

13. Confirm that the user is authorized to process the data sets associated with his ACB.

14. Obtain information from the catalog for the object named on the DD statement and for all data sets associated with the object.

15. Build save lists for cleaning up storage during task termination.

16. Prevent other tasks from opening the data sets during this open.

17. **Is the object already open for this job step?**

No / Yes → 5

18. Build the base information block for the object being opened and its associated data sets.

19. Unless a catalog or a catalog recovery area is being opened or processing will be with shared resources, build a working storage header.

20. Load the routines required for the rest of Open processing.

Notes for Diagram AC2

11 IDA0192A: INIT192A

The open work area is mapped by the IDAOPWRK macro.

13 IDA0192C

The user establishes the number of times the operator may attempt to supply the correct password, as described in *OS/VS2 Access Method Services.* If the correct password isn't supplied, VSAM Open sets the 'ACB not opened' return code in register 15 and the 'user password invalid' flag in ACBERFLG.

14 IDA0192C: LOC1

LOC1 issues a LOCATE (SVC 26) to obtain data-set type, catalog ACB address, catalog recovery area volume serial number, and control-interval number for each data set associated with the object named on the DD statement.

15 IDA0192A: BLDLISTS

During termination the ENQs indicated in the ESL (enqueue save list) will be dequeued, the DEBs indicated in the DSL will be unchained, the storage ("core") indicated in the CSL will be freed, and the pages indicated in the PLS will be freed. The SSL enables Open to chain control blocks at the end of Open processing.

16 IDA0192A: BLDENQPL, INIT192A

Open enqueues on each data set to prevent it from being opened by other tasks during the current Open processing.

17 IDA0192A: CONBASE

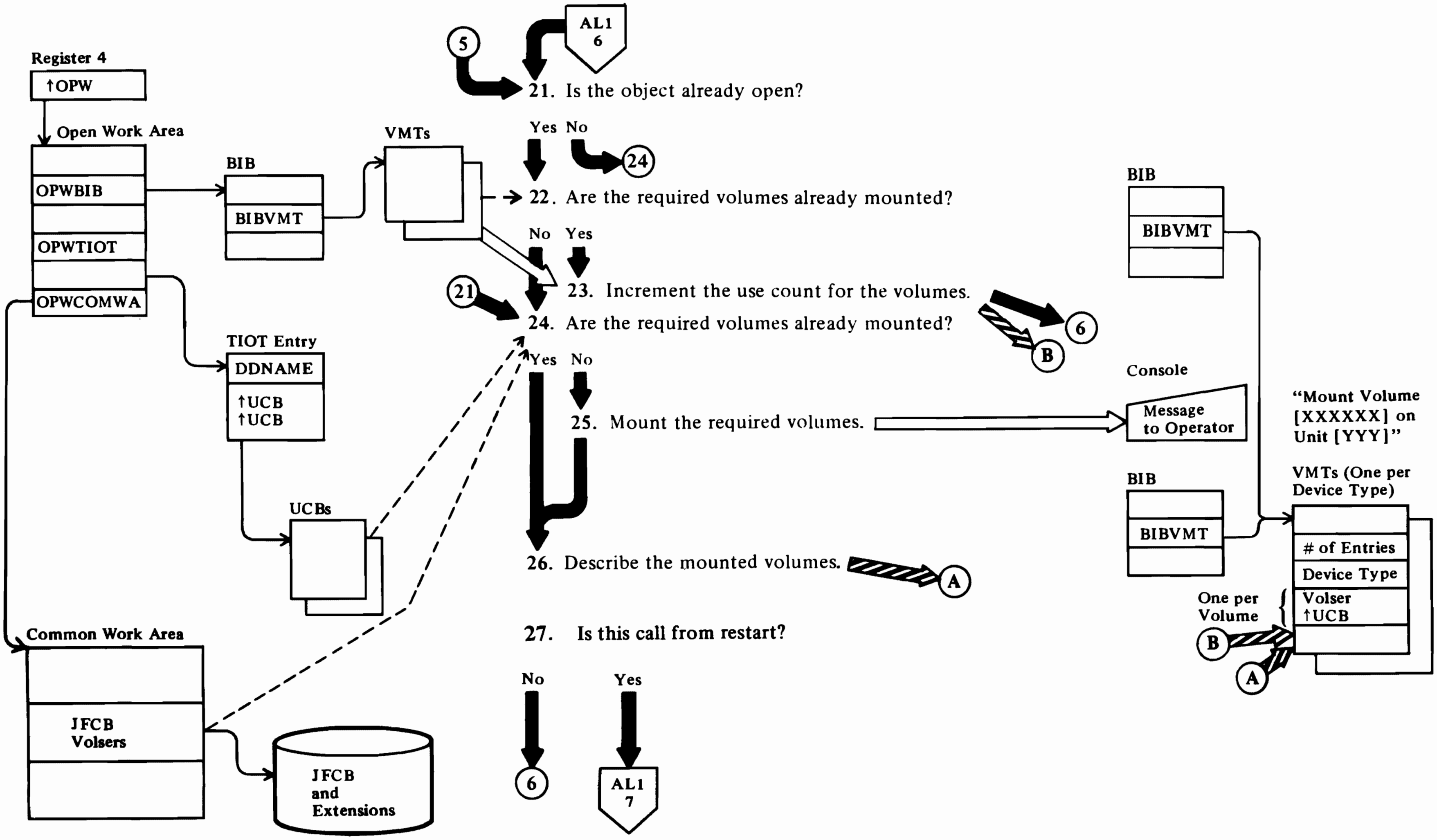
If the IDF field in the AMBL of the data set being opened matches the IDF field of an AMBL on the primary chain, the control blocks for the base cluster already exist.

18 The base information block contains the addresses of many of the control blocks built by Open for Record Management.

19 No working storage header is used for processing a catalog, which is a special case.

20 The addresses of various VSAM routines (Record-Management modules, I/O appendages, special processing routines) are placed in various control blocks (AMBL, IOSB, IRB, DEB, EXLST).

Diagram AC3. VSAM OPEN: Mount and Verify Volumes



Notes for Diagram AC3

21 IDA0192F: VOLMNT

22 IDA0192F: OLDDEV

24 IDA0192V

A volume in the JFCB and extensions is already mounted if a UCB allocated to the DD statement that is associated with the user ACB indicates so.

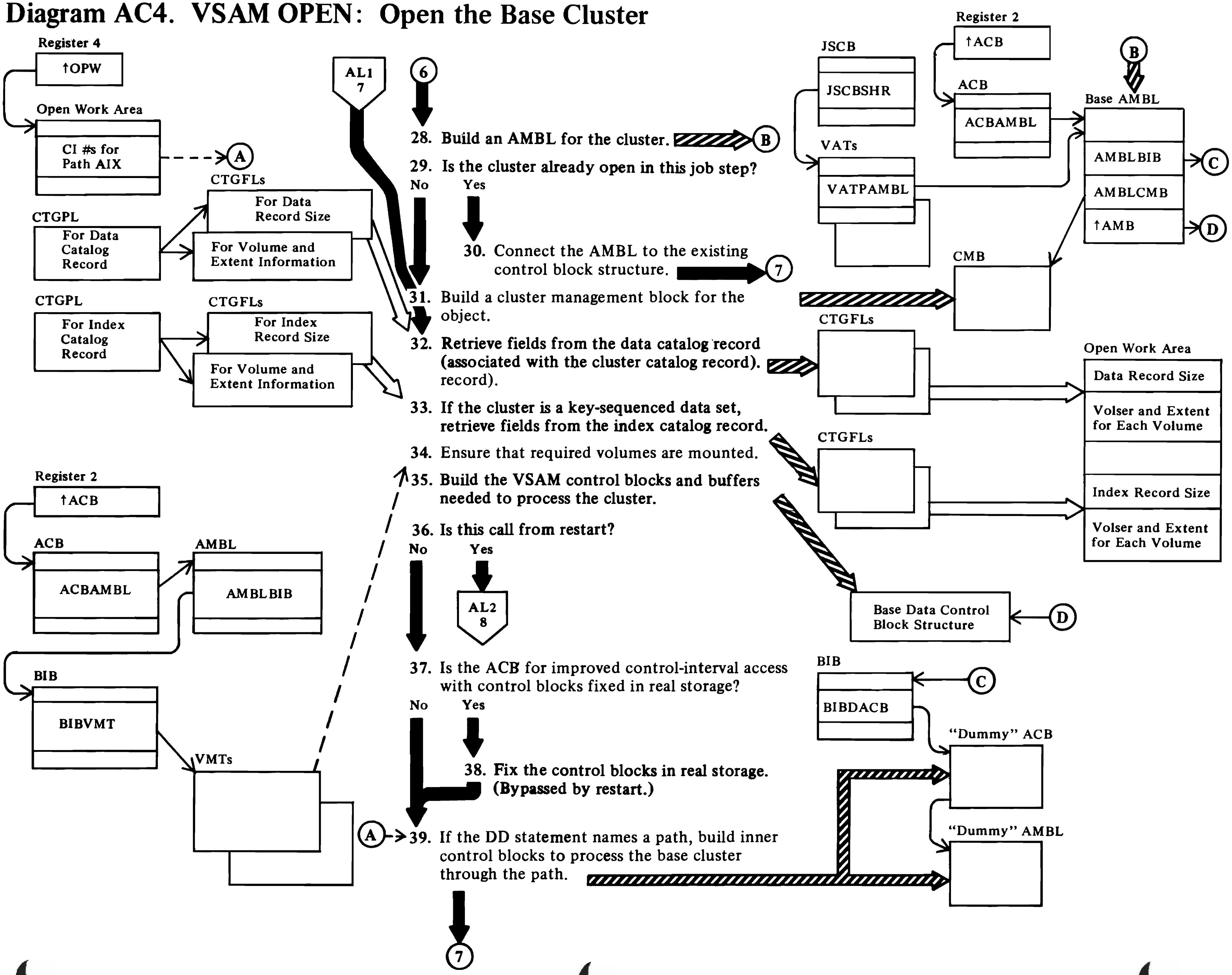
26 IDA0192F: OLDDEV, NEWDEV

A volume mount table is built for each device type allocated to the DD statement that is associated with the user ACB. Each VMT contains an entry for each successfully mounted volume of that device type. If a VMT already exists for a device type, the new VMT replaces the old one.

27 **IDA0192F**

If called from IDA0A05B (VSAM restart), OPWRSTRT will be on in the Open work area.

Diagram AC4. VSAM OPEN: Open the Base Cluster



28. Build an AMBL for the cluster.

29. Is the cluster already open in this job step?
No Yes

30. Connect the AMBL to the existing control block structure.

31. Build a cluster management block for the object.

32. Retrieve fields from the data catalog record (associated with the cluster catalog record). record).

33. If the cluster is a key-sequenced data set, retrieve fields from the index catalog record.

34. Ensure that required volumes are mounted.

35. Build the VSAM control blocks and buffers needed to process the cluster.

36. Is this call from restart?
No Yes

37. Is the ACB for improved control-interval access with control blocks fixed in real storage?
No Yes

38. Fix the control blocks in real storage. (Bypassed by restart.)

39. If the DD statement names a path, build inner control blocks to process the base cluster through the path.

Notes for Diagram AC4

28 **IDA0192F: OPNBASE, BLDAMBL, CHNAMBL, VATUPD**

Unless the user ACB indicates that a catalog is to be opened or that a catalog recovery area is to be built in system storage (SCRA), the AMBL is added to the chain and its address is added to the valid-AMBL table. The VAT is used for checking AMBLs for validity. AMBLVC identifies the VAT and the entry in the VAT that contains the address of the AMBL.

29 **IDA0192F: CHNAMBL**

30 **IDA0192F: CHNAMBL**

The AMBL is put on the secondary chain, off the primary AMBL for the base cluster.

31 **IDA0192F: BLDCMB**

32 **IDA0192B, IDA0192C: OPCAT1** (calls **LOC2** and **LOC3)**

A separate CTGFL is built for each catalog record field requested by VSAM Open. A CTGFL gives the field's length and its address in the open work area.

See *OS/VS2 Catalog Management Logic* for details about the data set catalog record, the CTGPL, and the CTGFL.

33 **IDA0192B, IDA0192C: LOC2, LOC3**

The index catalog record is pointed to by the cluster catalog record. See *OS/VS2 Catalog Management Logic* for details about the index catalog record.

34 **IDA0192B**

A volume mount table must exist for each device type required by the cluster.

35 **IDA0192Z, IDA0192Y, IDA0192W**

The following figures in "Data Areas" show the VSAM control block structure:

- VSAM Control Block Structure for a Key-Sequenced Data Set (VSAM User)
- VSAM Control Block Structure for a Key-Sequenced Data Set Accessed through a Path
- Shared VSAM Control Block Structure for a Key-Sequenced Data Set Accessed through Two Paths
- Data AMB Control Block Structure
- Alternate-Index AMB Control Block Structure
- Index AMB Control Block Structure
- Shared Resources Control Block Structure
- AMB Control Block Structure with Shared Resources

"Data Areas" also describes each VSAM control block.

36 **IDA0192B**

If the caller was VSAM restart, the return (register 14 in the caller's standard save area) will be to IDA0A05B.

37 If the caller (issuer of SVC 19) is authorized, and if ACBICI and ACBNCFX were specified, then AMBLFIX is set.

38 **IDA0192F: OPNBASE, PAGEFIX**

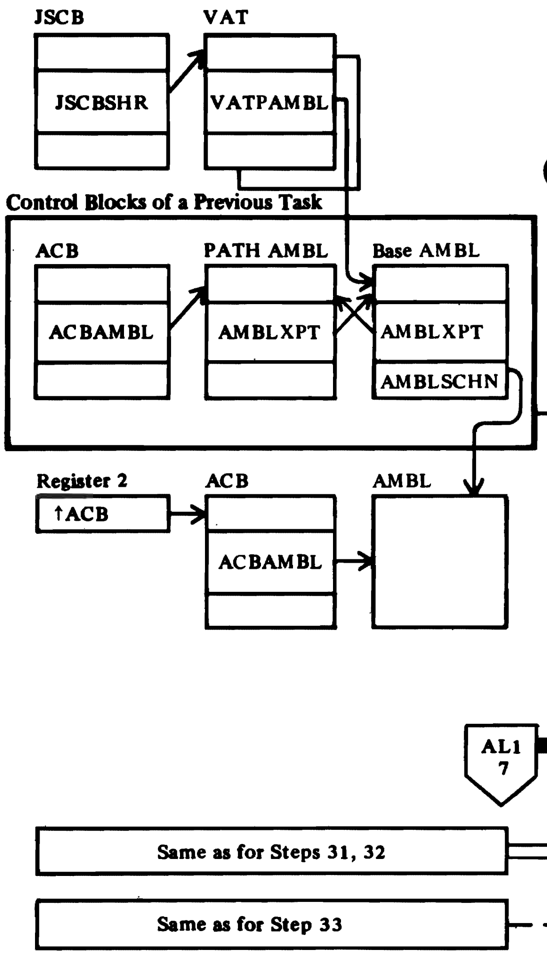
The user must be authorized to have pages fixed in real storage—his program must be in supervisor state with protection key 0 or link-edited with APF authorization.

All storage identified by the cluster management block is fixed.

39 **IDA0192F: OPNBASE**

The internal ACB and AMBL are used for gaining access to the base cluster through a path via the alternate index.

Diagram AC5. VSAM OPEN: Open the Upgrade Set



7

40. Build an upgrade table for the upgrade set.

41. Build control blocks for processing each alternate index in the upgrade set.

Repeat steps 42-46 for each alternate index in the upgrade set.

42. If the alternate index is already open for processing by way of a path, build additional control blocks for upgrading the alternate index and process the next alternate index in the upgrade set.

43. Build a CMB for the alternate index.

44. Retrieve fields from the data and index component catalog records.

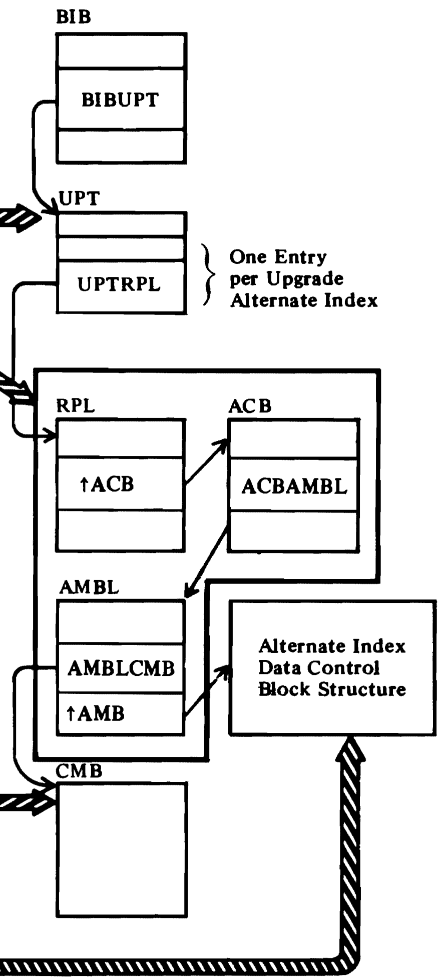
45. Ensure that required volumes are mounted.

46. Build the VSAM control blocks and buffers needed to process the alternate index.

47. **Is this call from restart?**

No 8

Yes AL2 8


BIB
BIBUPT
UPT
UPTRPL
One Entry per Upgrade Alternate Index
RPL
↑ACB
ACB
ACBAMBL
AMBL
AMBLCMB
↑AMB
Alternate Index Data Control Block Structure
CMB

Notes for Diagram AC5

40 **IDA0192F: OPNUPGR**

The upgrade table contains an entry for each alternate index in the upgrade set.

41 **IDA0192F: OPNUPGR, BLDAMBL**

An RPL, an ACB, and an AMBL are built for each alternate index.

42 **IDA0192F: OPNUPGR**

The AMBLs for paths already open in the job step are searched for the alternate index being processed.

IDA0192Y

To provide an additional string for upgrading an alternate index that is already open for processing by way of a path, IDA0192Y builds the PLH, BUFC, IOMB, IOSB, CPA, and buffers. These control blocks are described in "Data Areas."

43 **IDA0192F: BLDCMB**

44 See notes for steps 32 and 33.

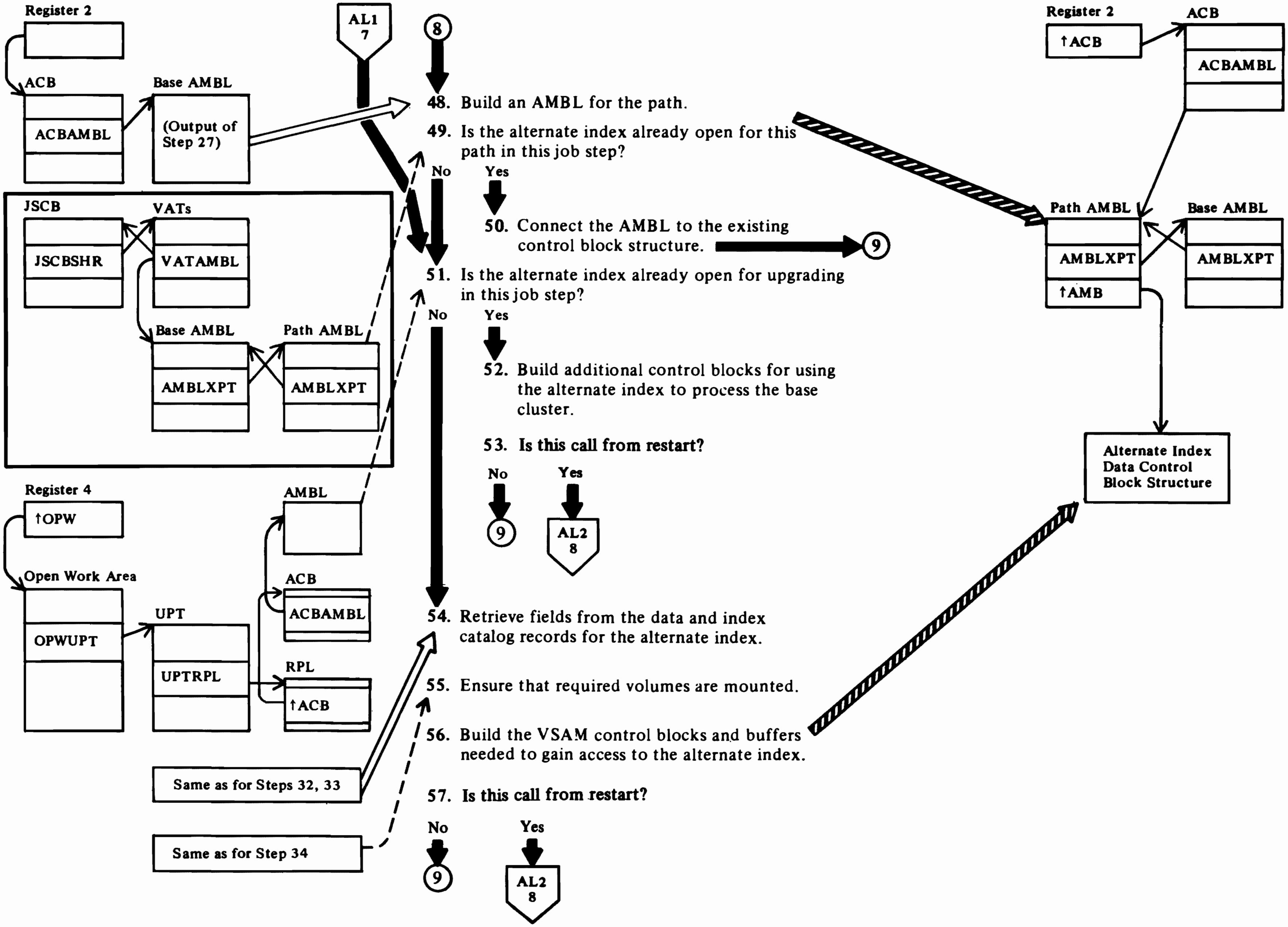
45 See note for step 34.

46 See note for step 35.

47 **IDA0192B**

If called from VSAM restart, the return (register 14 in the caller's standard save area) will be to IDA0A05B.

Diagram AC6. VSAM OPEN: Open the Alternate Index Associated with the Path



48. Build an AMBL for the path.

49. Is the alternate index already open for this path in this job step?

No Yes

50. Connect the AMBL to the existing control block structure.

51. Is the alternate index already open for upgrading in this job step?

No Yes

52. Build additional control blocks for using the alternate index to process the base cluster.

53. **Is this call from restart?**

No Yes

54. Retrieve fields from the data and index catalog records for the alternate index.

55. Ensure that required volumes are mounted.

56. Build the VSAM control blocks and buffers needed to gain access to the alternate index.

57. **Is this call from restart?**

No Yes

Notes for Diagram AC6

48 **IDA0192F: OPNPATH, BLDAMBL**

The AMBL is chained off the current AMBL for the base cluster. Its address is added to the valid-AMBL table. The VAT is used for checking AMBLs for validity. AMBLVC identifies the VAT and the entry in the VAT that contains the address of the AMBL.

49 **IDA0192F: CONPATH**

The alternate index is already open for this path if one of the path AMBLs contains the same ID as this alternate index.

50 **IDA0192F: OPNPATH**

The AMBL is chained off the existing AMBL for the path.

51 **IDA0192F: CONPATH**

The alternate index is already open for upgrading if one of the AMBLs pointed to by the upgrade table contains the same ID as this alternate index.

52 **IDA0192F: CONPATH**

For each string required for processing the path, IDA0192F builds the PLH, BUFC, CPA, IOMB, IOSB, SRB, and buffers. These control blocks are described in "Data Areas."

53 **IDA0192F**

If called from VSAM restart, the return (register 14 ir the caller's standard save area) will be to IDA0A05B.

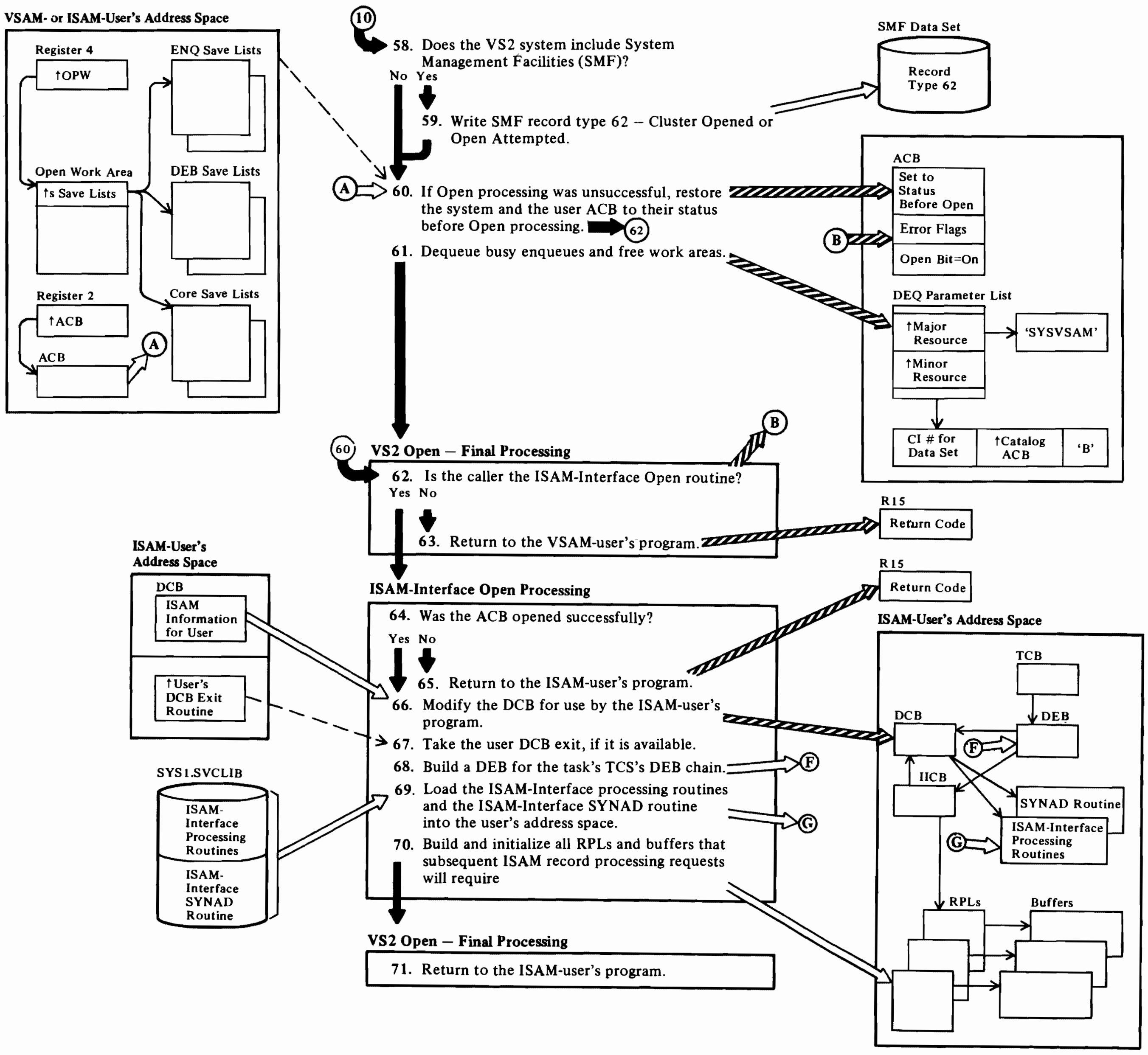
54 See notes for steps 32 and 33.

55 See note for step 34.

56 See note for step 35.

57 See note for step 53.

Diagram AC7. VSAM OPEN: Terminate Open Processing



58. Does the VS2 system include System Management Facilities (SMF)?

59. Write SMF record type 62 – Cluster Opened or Open Attempted.

60. If Open processing was unsuccessful, restore the system and the user ACB to their status before Open processing.

61. Dequeue busy enqueues and free work areas.

VS2 Open – Final Processing

62. Is the caller the ISAM-Interface Open routine?

63. Return to the VSAM-user's program.

ISAM-Interface Open Processing

64. Was the ACB opened successfully?

65. Return to the ISAM-user's program.

66. Modify the DCB for use by the ISAM-user's program.

67. Take the user DCB exit, if it is available.

68. Build a DEB for the task's TCS's DEB chain.

69. Load the ISAM-Interface processing routines and the ISAM-Interface SYNAD routine into the user's address space.

70. Build and initialize all RPLs and buffers that subsequent ISAM record processing requests will require

VS2 Open – Final Processing

71. Return to the ISAM-user's program.

Notes for Diagram AC7

58 **IDA0192A: TERM192A, UPSMF**

59 **IDA0192S**

See *OS/VS System Management Facilities (SMF)* for details about SMF record type 62.

60 **IDA0192A: TERM192A, CLNUP**

CLNUP resets open indicators in the VSAM catalog for data sets that were processed. It unchains AMBLs and deletes entries from the valid-AMBL table. It unchains DEBs. It decrements any use counts that were incremented.

CLNUP deletes all volume mount table entries that were added.

61 **IDA0192A: DEQBUSY**

A DEQ is issued for each data set that was enqueued busy (in step 16) to allow other tasks to open them.

63 **IDA0192A**

The VSAM Open routine sets the ACB's open bit (ACBOFLGS) on if the ACB is opened successfully. If an error occurs while opening an ACB, the VSAM Open routine or VS2 Open sets the appropriate error flag.

The VSAM Open routine returns control to VS2 Open by putting the identifier of the Open Final Termination routine, C'8N', in the WTG table and transferring control (through the IECRES macro) to the Open/Close/End-of-Volume resident routine. The resident routine examines the open parameter list and, if all ACB entries have been processed by the VSAM Open routine, returns to the VS2 Open Final Termination routine. If not, the next ACB entry in the open parameter list is processed (return to step 4).

VS2 Open modules (IFG0196V and IFG0196W) ensure that an ACB entry in the open parameter list is not processed by any access method executor routine.

IFG0196V sets the identifier for each VSAM ACB entry in the WTG table to 0.

IFG0196W sets the identifier for each VSAM ACB entry in the WTG table to C'8N', the identifier of the VS2 Open Final Termination routine.

IFG0198N sets the return code in register 15.

See "Diagnostic Aids" for details about the VSAM Open return codes.

64 **IDA0192I: OPENACB**

The ISAM-Interface Open routine sets the DCB open bit (DCBOFLGS) to 1 if the DCB's associated ACB was opened correctly.

66 **IDA0192I: DCBMERGE, AMSMERGE, VALIDCHK**

See *OS/VS2 Data Areas* for details about the DCB.

67 **IDA0192I: DCBEXIT**

Register contents passed to the user's DCB exit routine are:

- R1: address of DCB
- R2 through 13: User's registers
- R14: return address
- R15: address of user's DCB exit routine

IDA0192I: BFRMERGE

Merge buffer-related information into the DCB.

68 **IDA0192I: BUILDDEB**

The ISAM-Interface Open routine builds a DEB so that:

- There is meaningful DEB information for the user's program to examine;
- The DEB fields on which COBOL, PL/I, and ISAM System Integrity routines depend are properly initialized;
- The checkpoint/restart or abnormal end (ABEND) routines can examine the task's DEB chain and close all of the user's DCBs and ACBs; and
- The user's program cannot modify the IICB address or other fields in the DEB.

The DEB's ISAM-Interface indicator is now set on.

See *OS/VS2 Data Areas* for details about the DCB, DEB, and TCB.

69 **IFG0192I: LOADMOD**

Each DCB module-address field addresses an ISAM-Interface processing routine that will translate an ISAM record-processing request into a VSAM request.

The ISAM SYNAD routine is loaded when it is specified in the user's JCL AMP parameter.

The EXLST (built in step 2) addresses ISAM exit routines.

See "Data Areas" for details about the EXLST.

The DEB (built in step 68) is initialized to point to the ISAM-Interface FREEDBUF routine.

70 **IDA0192I: BLDRPL, INITRPL, BLDBUFR**

RPLs and ISAM-Interface buffers are built for each ACB (the number of RPLs and buffers is based on the ACB's STRNO value for BISAM; one of each is built for QISAM) that the ISAM user opens. Two of the uses of the ISAM-Interface buffers are to support ISAM locate mode and dynamic buffer processing.

IDA0192I: DCBINIT

When the ISAM-Interface Open processing completes, the DCB open flags (DCBOFLGS) field contains:

- Busy bit on (set to 0)
- Open bit on (set to 1)
- Lock bit off (set to 1)

71

VS2 Open modules (IFG0196V and IFG0196W) ensure that a DCB for a VSAM entry in the open parameter list is not processed by any access method executor routine.

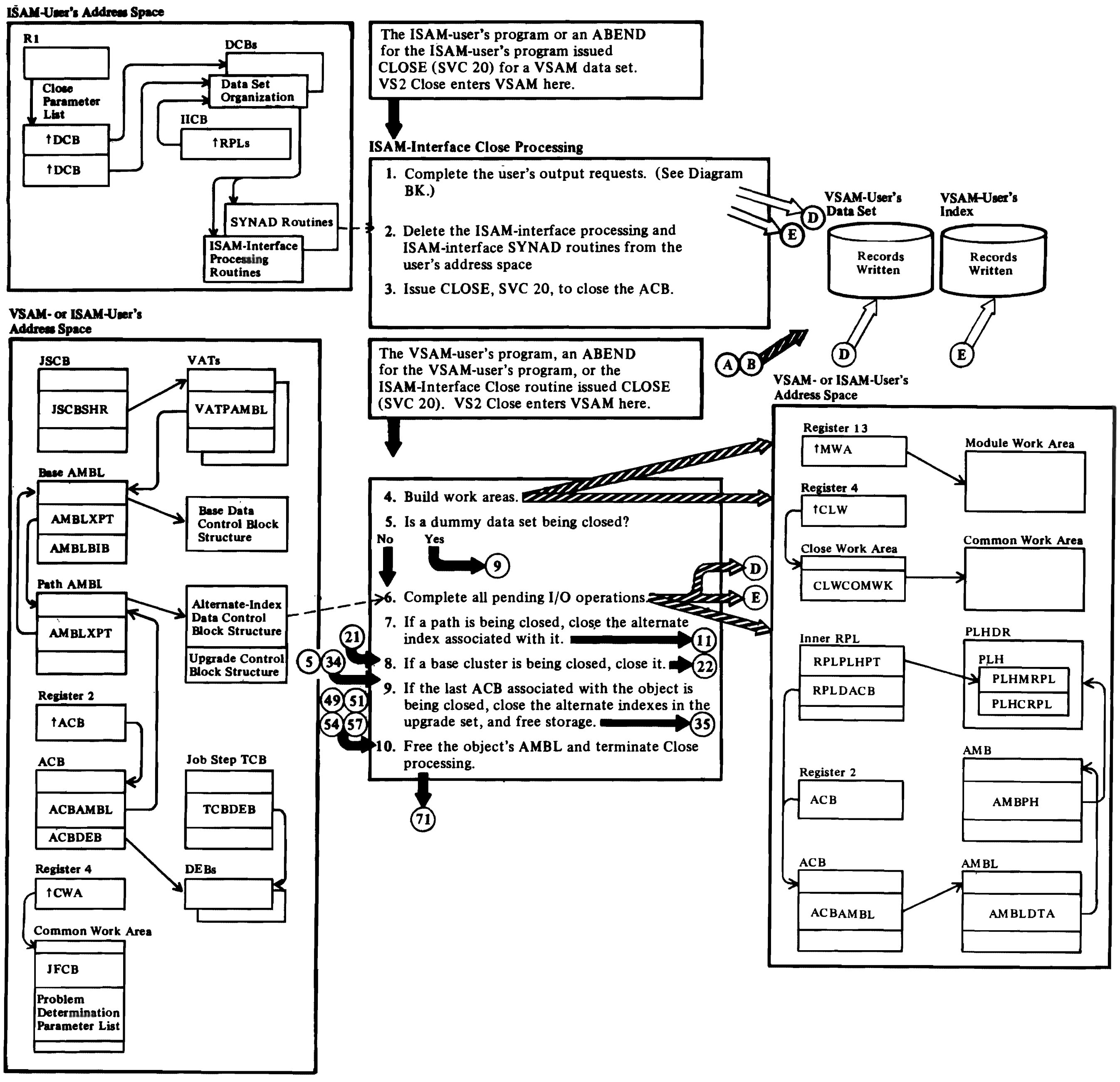
IFG0196V sets the ID field for each DCB-for-VSAM entry in the WTG table to 0.

IFG0196W sets the identifier field for each DCB-for-VSAM entry in the WTG table to C'8N', the identifier of the VS2 Open Final Termination module (IFG0198N).

IFG0198N sets the return code in register 15.

If the ACB (built by the ISAM-Interface Open routine in step 2) is not opened correctly by the VSAM Open routine, the ISAM-Interface Open routine sets the DCB open bit to 0 (DCBOFLGS) and sets all DCB module-address fields to 0. If the user's ISAM program issues an ISAM record processing request without confirming that the DCB is successfully opened, an ABEND 0C4 (caused by a branch to address 000) results.

Diagram AD1. VSAM CLOSE: Disconnect a User from a VSAM Data Set



The ISAM-user's program or an ABEND for the ISAM-user's program issued CLOSE (SVC 20) for a VSAM data set. VS2 Close enters VSAM here.

ISAM-Interface Close Processing

1. Complete the user's output requests. (See Diagram BK.)
2. Delete the ISAM-interface processing and ISAM-interface SYNAD routines from the user's address space
3. Issue CLOSE, SVC 20, to close the ACB.

The VSAM-user's program, an ABEND for the VSAM-user's program, or the ISAM-Interface Close routine issued CLOSE (SVC 20). VS2 Close enters VSAM here.

4. Build work areas.
5. Is a dummy data set being closed?
   No / Yes → 9
6. Complete all pending I/O operations.
7. If a path is being closed, close the alternate index associated with it. → 11
8. If a base cluster is being closed, close it. → 22
9. If the last ACB associated with the object is being closed, close the alternate indexes in the upgrade set, and free storage. → 35
10. Free the object's AMBL and terminate Close processing. → 71

Notes for Diagram AD1

If the DCB data-set organization (DCBDSORG) field indicates that an ACB is being processed and if the DEBFLGS1 field (in the DEB) indicates ISAM-Interface processing, VS2 Close modules (IGC00020 and IFG0200V) do the following:

IGC00020: Bypasses purging of the outstanding EXCP requests.

IFG0200V: Bypasses DSCB processing and transfers control to the ISAM-Interface Close routine, IDA0200S.

1 **IDA0200S: FLUSHBFR**

The ISAM-Interface Close routine issues a SYNCH macro to transfer control to the ISAM-Interface Load routine, which issues the final PUT request, if all of these conditions exist:

- The DCB was opened for output in the locate mode and a PUT request was issued prior to the CLOSE request (indicated in the DCBMACRF field).
- No errors occurred (indicated in the DCBEXCD field).
- The ACB associated with the user program's DCB was not previously closed (indicated in the ACBOFLGS field).

See "Data Areas" for details about the ACB.

See *OS/VS2 Data Areas* for details about the DCB and the DEB.

2 **IDA0200S: DELETRTN**

The ISAM-Interface Close routine resets each DCB module address field. Virtual storage for the routines is released to the system by issuing a DELETE macro against the ISAM-Interface routines that were loaded by ISAM-Interface Open processing.

3 **IDA0200S: CLOSEACB**

The ISAM-Interface Close routine issues a CLOSE macro (SVC 20) to close the VSAM ACB.

VS2 Close modules (IGC00020 and IFG0200V) allow an ACB to be closed and copy it into the Close work area.

IGC00020 bypasses the DEB validity check and the purging of outstanding EXCP requests and, if a VS2 catalog is being closed, calls IFG0200N to locate the TIOT entry and read the JFCB for the catalog ACB.

IFG0200V reads the JFCB for non-catalog ACBs and tests for the user program's diagnostic options (that is, Generalized Trace Facility), and sets the ID field for each ACB entry in the WTG table to C'0T', the identifier of the VSAM Close module.

VSAM Close Processing

The input is from IFG0200T.

4 **IDA0200T: INIT200T, GETCORE**

The module work area and the close work area are built.

If neither a catalog nor a catalog recovery area in system storage (SCRA) is being closed, the dummy DEB is verified. Unless a dummy data set is being closed, IDA0200T (ENQFUNC, ENQINIT, PARMINIT) builds an ENQ parameter list and issues ENQ for every data set associated with the user ACB. The parameter list indicates 'SYSVSAM' as the major resource and control-interval number of the data set, catalog ACB address, and 'B' (busy) as the minor resource.

6 **IDA0200T: FLQUIS, ENDIO**

If the close is not for an ABEND and is not for improved control-interval access to load a data set or process the mass storage volume inventory data set, the data set is flushed and quiesced (that is, any I/O activity yet to be done or already started is done):

An inner RPL is built and pointed to the user ACB. The PLH chain is searched for PLHs connected to the user ACB. The inner RPL is connected to each PLH and an ENDREQ macro is issued. No record is returned for an incomplete input request (GET or POINT). The output buffer is written to the VSAM data set for an incomplete output request (PUT or ERASE). After I/O completes, the inner RPL is freed.

7 **IDA0200T: CLSPATH** calls **IDA0200B**

The alternate index in a path is closed before the base cluster. See Diagram AD2.

8 **IDA0200T: CLSBASE** calls **IDA0200B**

The cluster being closed may be a base cluster (part of a path), a cluster that was not processed through a path, or an alternate index that was itself processed by the user. See Diagram AD3.

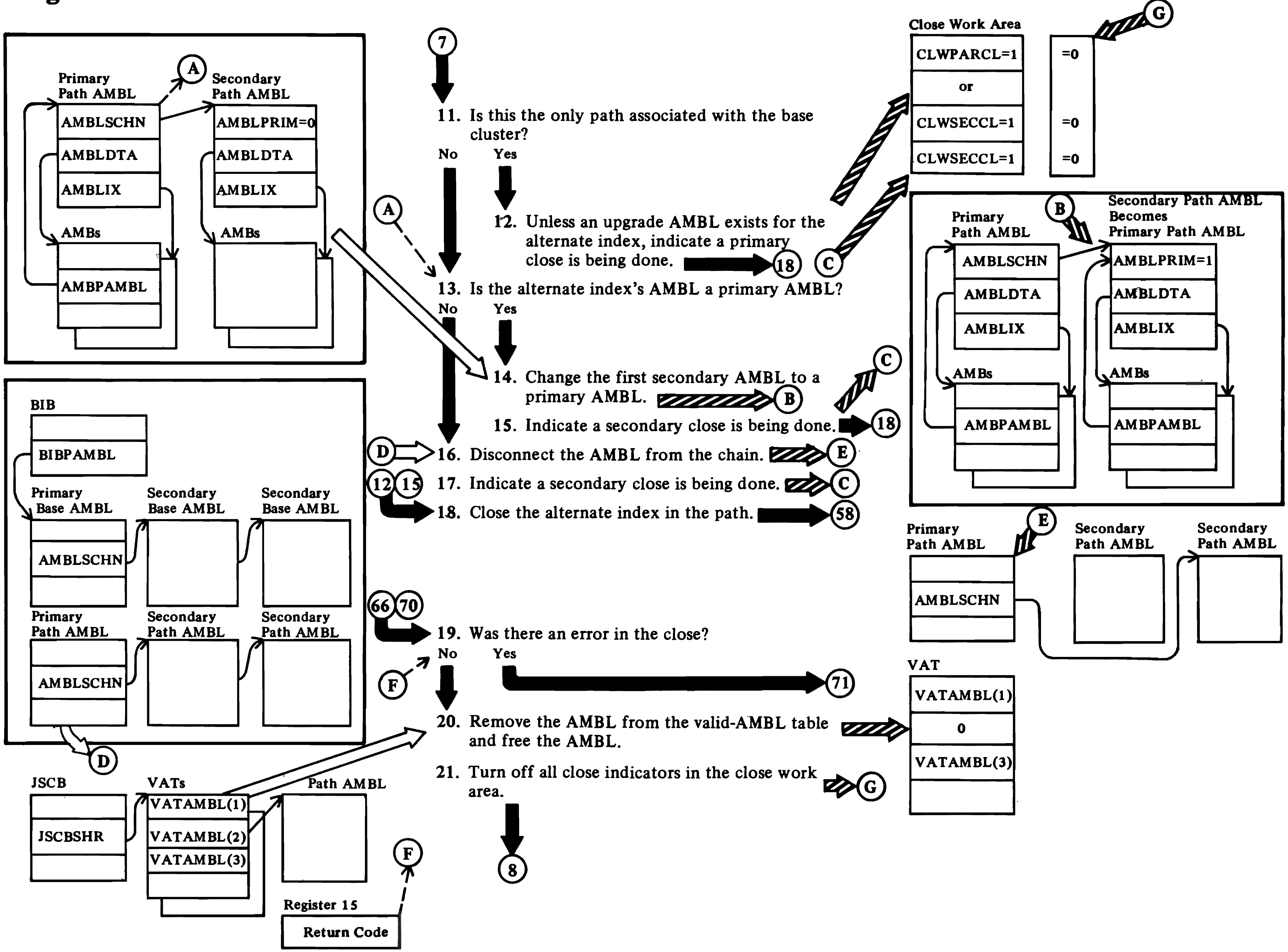
9 **IDA0200T: CLSPHERE**

This processing is not done if an ACB for the cluster is still open. For example, two users might have been processing a cluster, and the first user is closing his ACB. See Diagrams AD4 and AD5.

10 **IDA0200T: FREECORE, TERM200T**

See Diagram AD7 for a description of termination processing.

Diagram AD2. VSAM CLOSE: Close the Alternate Index in a Path



11. Is this the only path associated with the base cluster?
No Yes

12. Unless an upgrade AMBL exists for the alternate index, indicate a primary close is being done.

13. Is the alternate index's AMBL a primary AMBL?
No Yes

14. Change the first secondary AMBL to a primary AMBL.

15. Indicate a secondary close is being done.

16. Disconnect the AMBL from the chain.

17. Indicate a secondary close is being done.

18. Close the alternate index in the path.

19. Was there an error in the close?
No Yes

20. Remove the AMBL from the valid-AMBL table and free the AMBL.

21. Turn off all close indicators in the close work area.

Notes for Diagram AD2

12 IDA0200T

When an upgrade AMBL exists for the alternate index being closed, a partial close is indicated for Diagram AD6 processing. For a partial close, only the string blocks for the path, not for the upgrade set, are closed.

For a primary close, the last user is closing his ACB for the base cluster—no primary AMBL or related control blocks need be kept for further user processing.

15 IDA0200T

For a secondary close, at least one more user still has an ACB open for the base cluster—the primary AMBLs and related control blocks must be kept for further user processing.

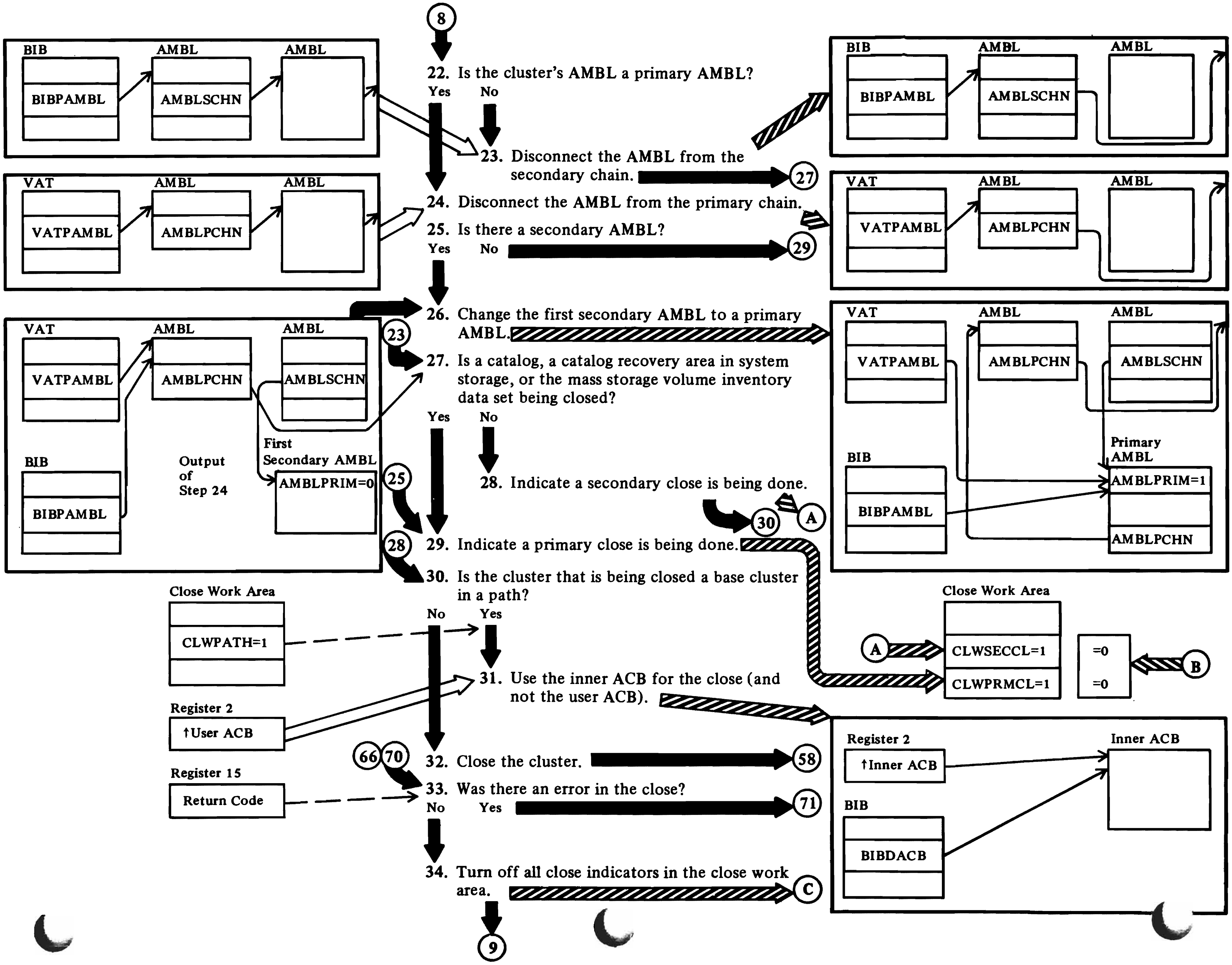
17 See note for step 15.

18 See Diagram AD6.

20 IDA0200T: FREECORE, RMOVAMBL

The AMBL entry is removed from the valid-AMBL table and the storage for the AMBL is freed.

Diagram AD3. VSAM CLOSE: Close the Base Cluster



8

22. Is the cluster's AMBL a primary AMBL?
Yes No

23. Disconnect the AMBL from the secondary chain. → 27

24. Disconnect the AMBL from the primary chain.

25. Is there a secondary AMBL?
Yes No → 29

26. Change the first secondary AMBL to a primary AMBL.

27. Is a catalog, a catalog recovery area in system storage, or the mass storage volume inventory data set being closed?
Yes No

28. Indicate a secondary close is being done.

29. Indicate a primary close is being done.

30. Is the cluster that is being closed a base cluster in a path?
No Yes

31. Use the inner ACB for the close (and not the user ACB).

32. Close the cluster. → 58

33. Was there an error in the close?
No Yes → 71

34. Turn off all close indicators in the close work area. → C

9

Notes for Diagram AD3

The cluster being closed can be a base cluster that was being processed through a path, a cluster that was *not* being processed through a path, or an alternate index that was itself processed by the user.

24 IDA0200T

For disconnecting the AMBL and changing AMBL pointers (step 26), an ENQ is issued to exclusively control the resources for the job step.

26 IDA0200T

After AMBL pointers are changed, a DEQ is issued to free the resources for the job step.

28 See note for step 15.

29 See the explanation for a primary close in the note for step 12.

31 IDA0200T

The inner ACB is used because the user ACB contains parameters for closing a path, not for closing a base cluster.

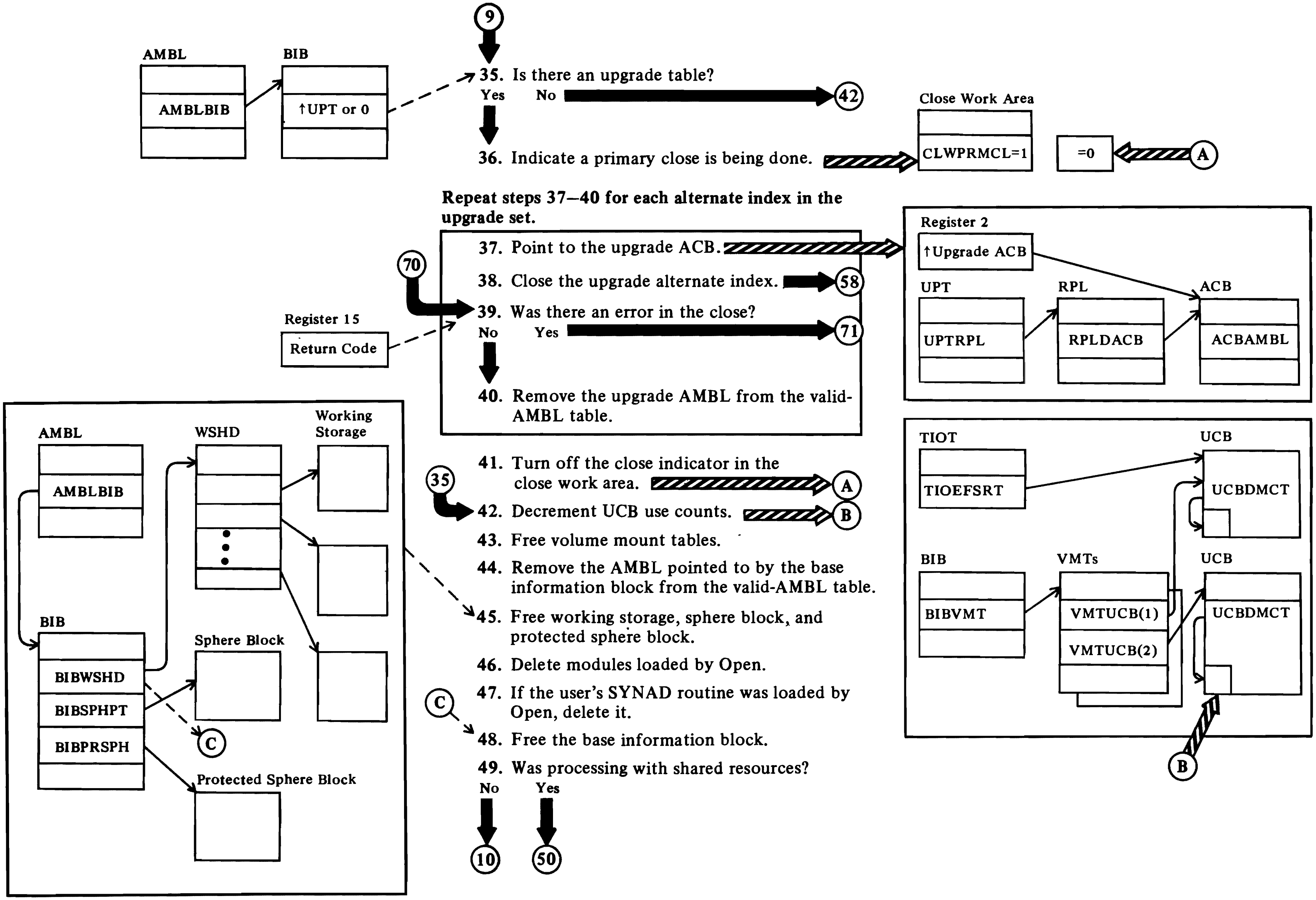
32 IDA0200T calls **IDA0200B**

See Diagram AD6.

33 IDA0200T: RMOVAMBL

If there was no error, register 2 is pointed back to the user ACB. Unless a catalog, a catalog recovery area in system storage (SCRA), or the mass storage volume inventory data set is being closed, the AMBL is removed from the valid-AMBL table.

Diagram AD4. VSAM CLOSE: Close Upgrade Alternate Indexes and Free Storage



35. Is there an upgrade table?
Yes No

36. Indicate a primary close is being done.

Repeat steps 37–40 for each alternate index in the upgrade set.

37. Point to the upgrade ACB.

38. Close the upgrade alternate index.

39. Was there an error in the close?
No Yes

40. Remove the upgrade AMBL from the valid-AMBL table.

41. Turn off the close indicator in the close work area.

42. Decrement UCB use counts.

43. Free volume mount tables.

44. Remove the AMBL pointed to by the base information block from the valid-AMBL table.

45. Free working storage, sphere block, and protected sphere block.

46. Delete modules loaded by Open.

47. If the user's SYNAD routine was loaded by Open, delete it.

48. Free the base information block.

49. Was processing with shared resources?
No Yes

Notes for Diagram AD4

35 IDA0200T: CLSUPGR

37 IDA0200T: CLSUPGR

After the last upgrade alternate index is closed, register 2 is pointed back to the user ACB.

38 IDA0200T calls **IDA0200B**

See Diagram AD6.

40 IDA0200T: RMOVAMBL

42 IDA0200T: VMTPROC, DCRUCBCT

Use counts are decremented one way for closing a catalog and another way for closing other data sets:

For closing a catalog, the UCB use count is decremented if the UCB indicated by the task I/O table is the same UCB as that indicated in the volume mount table.

If neither a catalog nor a catalog recovery area is being closed and restart isn't indicated, the UCB use counts in the volume mount table are decremented for those volumes with valid serial numbers.

43 IDA0200T: FREECORE

44 IDA0200T: RMOVAMBL

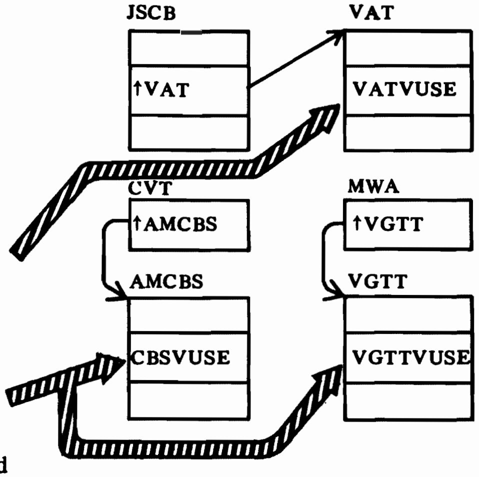
45 IDA0200T: FREECORE, FREESPHR

For information about the sphere block and the protected sphere block, see "Virtual-Storage Management" in "Diagnostic Aids."

48 IDA0200T: FREECORE

The base information block is described in "Virtual-Storage Management" in "Diagnostic Aids."

Diagram AD5. VSAM CLOSE: Close Upgrade Alternate Indexes



49

50. Were resources shared locally (LSR) or globally (GSR)?

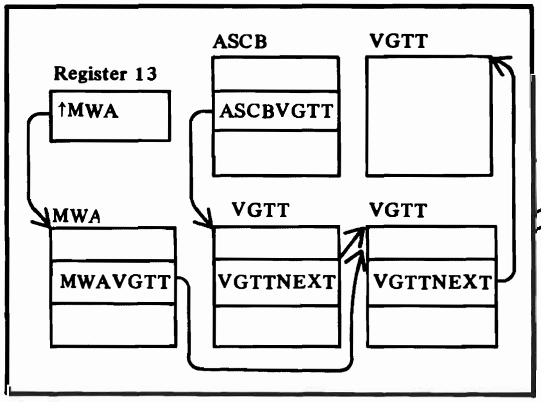
GSR LSR

51. Decrement the use count for the VSAM shared resource table. → 10

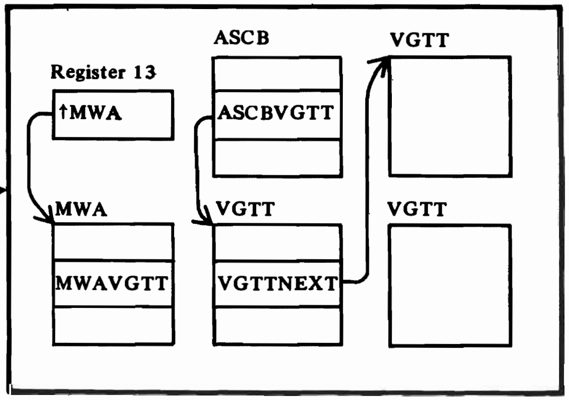
52. Decrement the use count for the VSAM shared resource table.

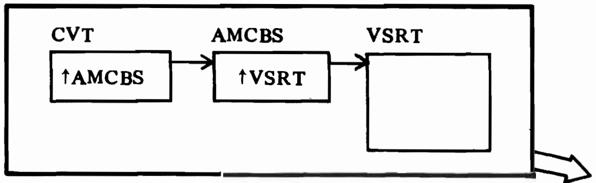
53. Is the global resource pool to be deleted (forced deletion)?

Yes No



54. If a catalog is not being closed, unchain the VSAM global termination table and free it. → 10

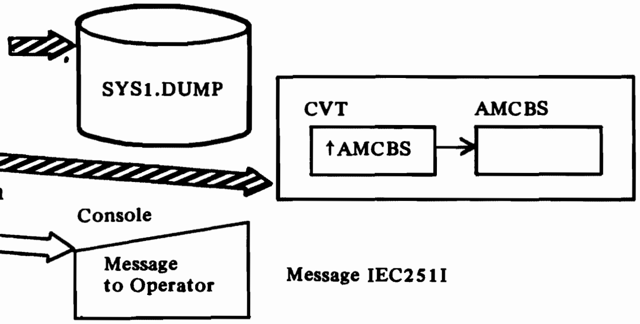




55. Dump the control blocks of the global resource pool.

56. Delete the global resource pool.

57. Write the forced-deletion message to the system operator. → 10



Notes for Diagram AD5

51 IDA0200T

The VAT use count in the valid-AMBL table is decremented by one.

52 IDA0200T

The VSRT use count in the access-method control block structure block and in the VSAM global termination table is decremented by one.

The AMCBS is described in *OS/VS2 Catalog Management Logic.*

54 IDA0200T: REMVGTT, FREVGTT

55 IDA0200T: GSRDUMP, SDLOAD

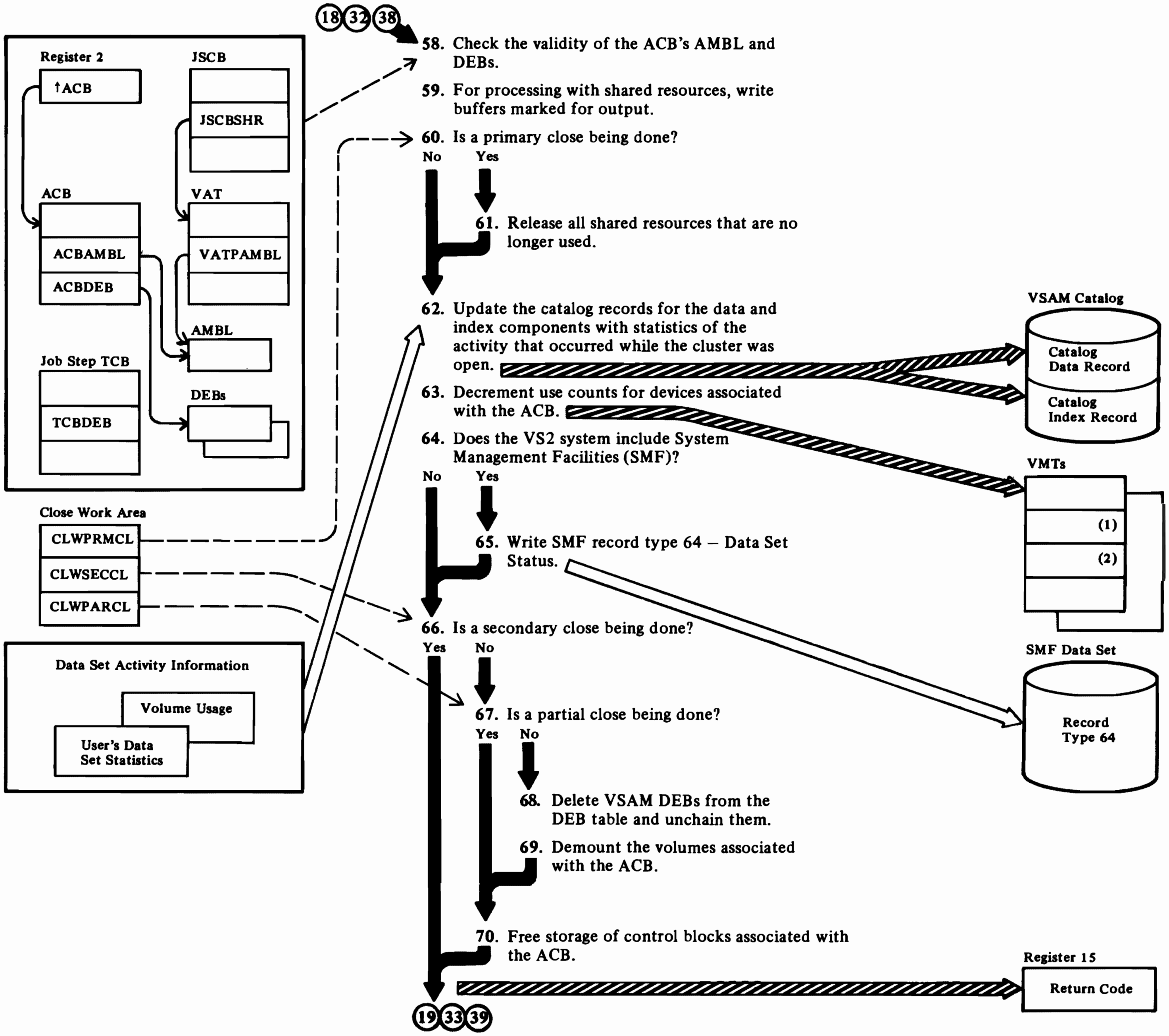
"Recovery with Global Shared Resources" in "Diagnostic Aids" describes the dumping of control blocks.

56 IDA0200T: FDLVRP calls **IDA0192Y**

IDA0192Y issues the DLVRP macro. Forced deletion is discussed in "Recovery with Global Shared Resources" in "Diagnostic Aids."

57 IDA0200T: FDLMSG calls **IDA0192P**

Diagram AD6. VSAM CLOSE: Close a Cluster



58. Check the validity of the ACB's AMBL and DEBs.

59. For processing with shared resources, write buffers marked for output.

60. Is a primary close being done?

61. Release all shared resources that are no longer used.

62. Update the catalog records for the data and index components with statistics of the activity that occurred while the cluster was open.

63. Decrement use counts for devices associated with the ACB.

64. Does the VS2 system include System Management Facilities (SMF)?

65. Write SMF record type 64 – Data Set Status.

66. Is a secondary close being done?

67. Is a partial close being done?

68. Delete VSAM DEBs from the DEB table and unchain them.

69. Demount the volumes associated with the ACB.

70. Free storage of control blocks associated with the ACB.

Notes for Diagram AD6

58 IDA0200B: INIT200B, VALCHECK, PROBDT (calls IDA0192P)

The DEBCHK SVC is used to check the validity of DEBs.

59 IDA0200B: WRITBUFR, GETCORE, WRBUFFER, CBINIT, FREECORE, PROBDT

Inner control blocks are built and the WRTBFR macro is issued to write data still in buffers.

60 See the explanation for a primary close in the note for step 12.

61 IDA0200B: SHARE, SHAREDEQ

DEQ is issued.

62 IDA0200B: UPCATACB, UPCATDEQ (calls IDA0192C), PROBDT

Catalog records are described in *OS/VS2 Catalog Management Logic.*

63 IDA0200B: VMTPROC

65 IDA0200B: UPSMF (calls **IDA0192S)**

One SMF record type 64, is written for each AMB (for data set or index) connected to the ACB's AMBL.

See *OS/VS2 System Programming Library: System Management Facilities (SMF)* for a description of SMF record type 64—Data Set Status.

See "Data Areas" for details about the AMDSB, AMB, AMBL, and ACB.

66 See note for step 15.

67 See the explanation of a partial close in the note for step 12. If neither a partial nor a secondary close is being done, a primary close is being done.

68 IDA0200B: DEHOOK

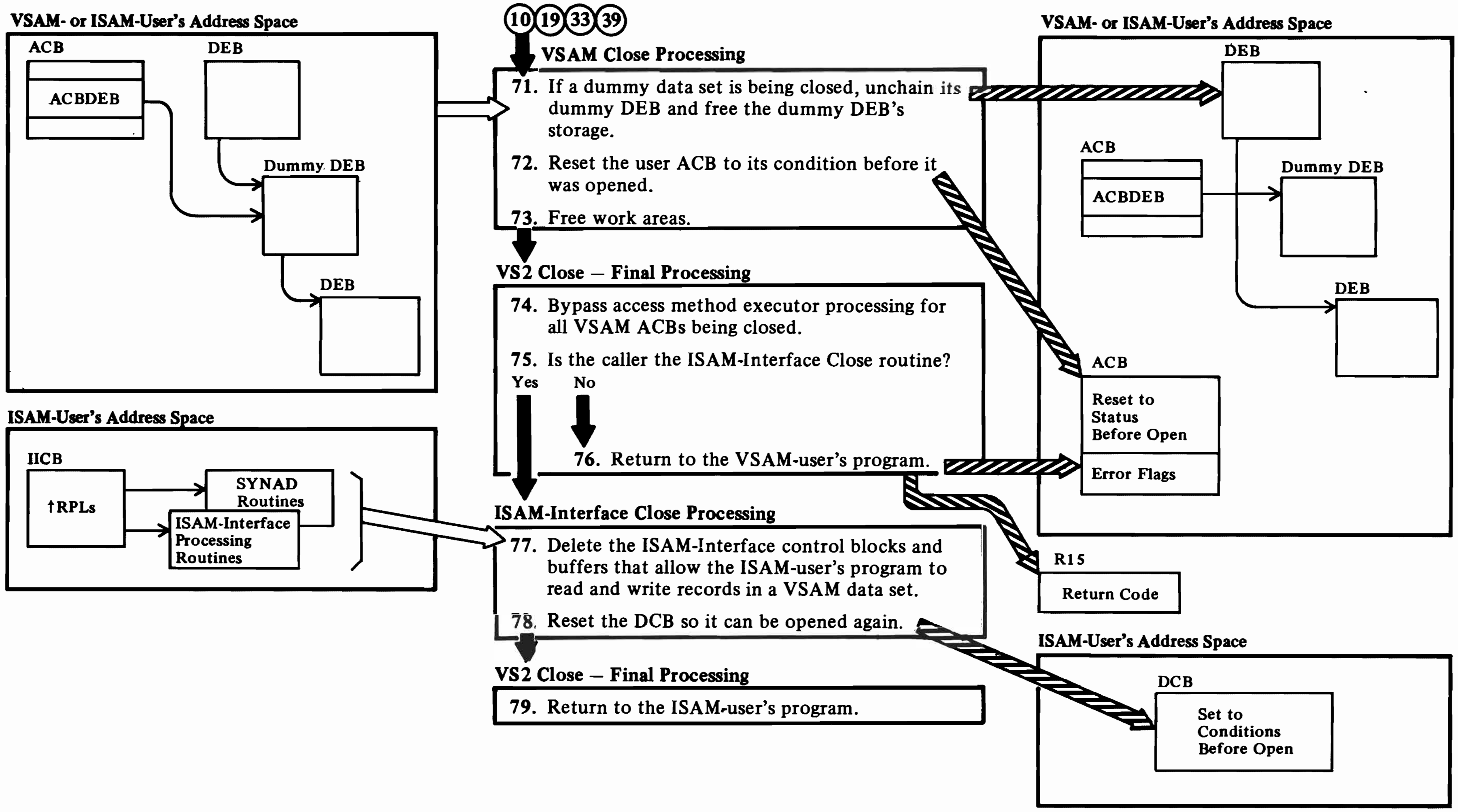
The DEBCHK SVC is used. It removes VSAM DEBs from the TCB DEB chain.

69 IDA0200B: VIRTPROC (calls **IDA0192D)**

IDA0192D destages data from the direct-access storage staging drive to mass storage.

70 IDA0200B: CBRELE

Diagram AD7. VSAM CLOSE: Terminate Close Processing


VSAM- or ISAM-User's Address Space
ACB
ACBDEB
DEB
Dummy DEB
DEB
ISAM-User's Address Space
IICB
↑RPLs
SYNAD Routines
ISAM-Interface Processing Routines
10
19
33
39
Yes
No
VSAM- or ISAM-User's Address Space
DEB
ACB
ACBDEB
Dummy DEB
DEB
ACB
Reset to Status Before Open
Error Flags
R15
Return Code
ISAM-User's Address Space
DCB
Set to Conditions Before Open

VSAM Close Processing

71. If a dummy data set is being closed, unchain its dummy DEB and free the dummy DEB's storage.

72. Reset the user ACB to its condition before it was opened.

73. Free work areas.

VS2 Close — Final Processing

74. Bypass access method executor processing for all VSAM ACBs being closed.

75. Is the caller the ISAM-Interface Close routine?

76. Return to the VSAM-user's program.

ISAM-Interface Close Processing

77. Delete the ISAM-Interface control blocks and buffers that allow the ISAM-user's program to read and write records in a VSAM data set.

78. Reset the DCB so it can be opened again.

VS2 Close — Final Processing

79. Return to the ISAM-user's program.

Notes for Diagram AD7

71 IDA0200T: DEHOOK, DECHNDEB

IDA0200T calls **IDA0192C**

If a catalog is being closed, IDA0192C issues a dummy LOCATE to indicate that the closing of the catalog is complete.

Unless a dummy data set has been closed (see note between notes for steps 4 and 6), a DEQ parameter list is built and a DEQ is issued for every data set associated with the user ACB. The parameter list indicates 'SYSVSAM' as the major resource and control-interval number of the data set, catalog ACB address, and 'B' (busy) as the minor resource.

72 IDA0200T: RESTORE

The ACB condition before it was opened is:

- Open bit (ACBOFLGS) is off
- Address of the VSAM interface routine (IDA019R1) is 0
- Address of the AMBL is 0
- DDNAME field contains the DDNAME from the TIOEDDNM field in the TIOT DD entry

73 IDA0200T: FREECORE

The storage for the close work area and the module work area is freed.

74 IDA0200T

The VSAM Close routine sets the ACB's open bit (ACBOFLGS) off if the ACB is closed successfully. If an error occurs while closing an ACB, the VSAM Close routine or VS2 Close sets the appropriate error flag.

The VSAM Close routine returns control to VS2 Close by putting the identifier of the Close Final Termination routine, X'2L', in the WTG table and transferring control (through the IECRES macro) to the Open/Close/End-of-Volume resident routine. The resident routine examines the close parameter list and, if all ACB entries have been processed by the VSAM Close routine, returns to the VS2 Close Final Termination routine. If not, the next ACB entry in the close parameter list is processed (return to step 4).

VS2 Close modules (IFG0200W and IFG0200Y) ensure that an ACB entry in the close parameter list is not processed by any access method executor routine.

IFG0200W sets the identifier for each VSAM ACB entry in the WTG table to **0**.

IFG0200Y sets the identifier for each VSAM ACB entry in the WTG table to C'2L', the identifier of the VS2 Close Final Termination routine.

77 IDA0200S: FREEBFRS, FREEDEB, RESETDCB, FREEWA, FREEMAIN

The ISAM-Interface Close routine releases the virtual storage obtained for the ACB, the IICB, the DEB, the RPLs, and the ISAM-Interface buffers.

78 IDA0200S: RESETDCB

The DCB conditions before open are:

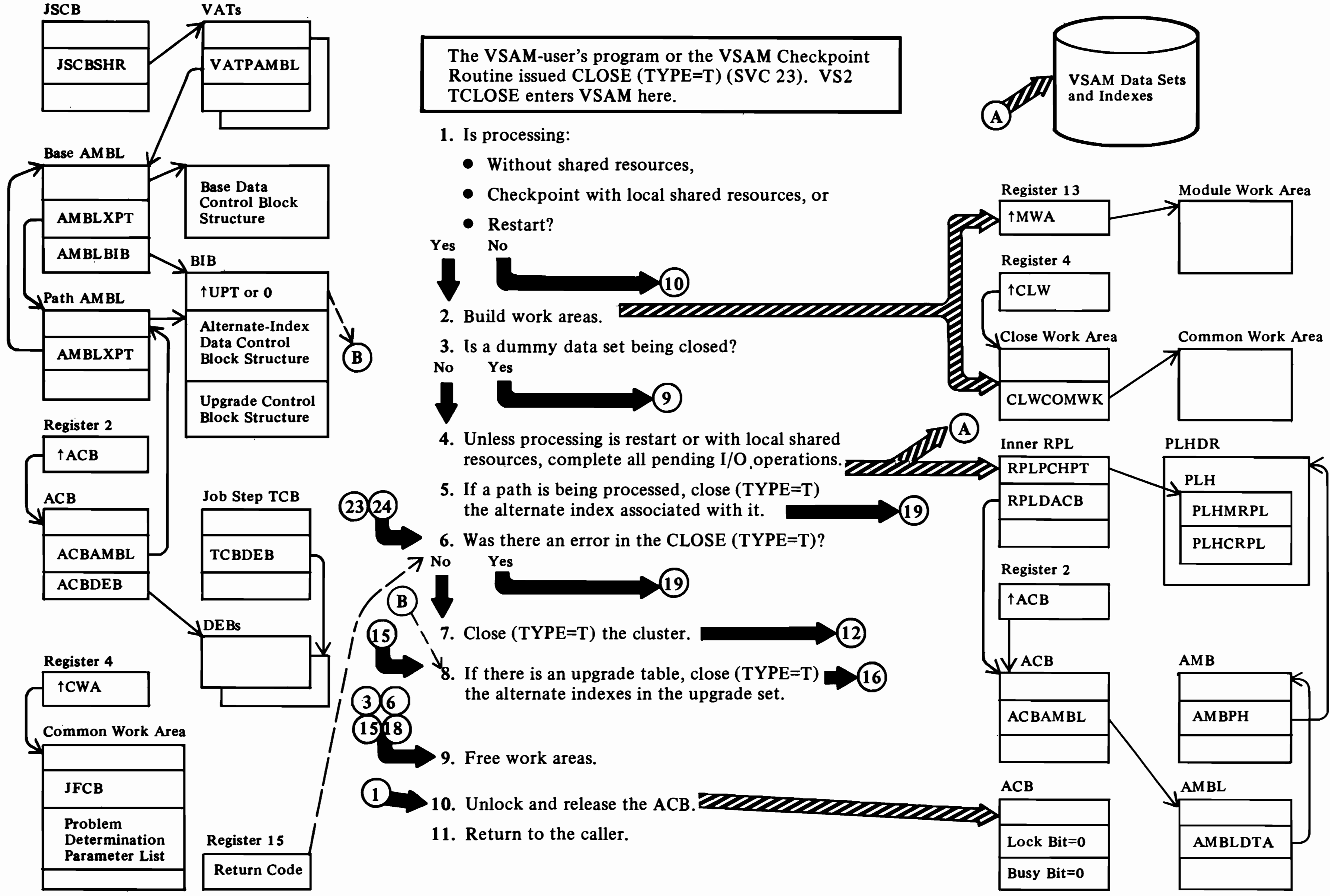
- DCBOFLGS: Open bit off, Lockbit off (set to 1), and Busy bit off
- DCBDSORG: ISAM-Interface bit off

79

IFG0202L sets the return code in register 15.

See "Diagnostic Aids" for details about the VSAM Close return codes.

Diagram AE1. VSAM CLOSE (TYPE=T)



The VSAM-user's program or the VSAM Checkpoint Routine issued CLOSE (TYPE=T) (SVC 23). VS2 TCLOSE enters VSAM here.

1. Is processing:
   - Without shared resources,
   - Checkpoint with local shared resources, or
   - Restart?

   Yes / No → 10
2. Build work areas.
3. Is a dummy data set being closed?

   No / Yes → 9
4. Unless processing is restart or with local shared resources, complete all pending I/O operations.
5. If a path is being processed, close (TYPE=T) the alternate index associated with it. → 19
6. Was there an error in the CLOSE (TYPE=T)?

   No / Yes → 19
7. Close (TYPE=T) the cluster. → 12
8. If there is an upgrade table, close (TYPE=T) the alternate indexes in the upgrade set. → 16
9. Free work areas.
10. Unlock and release the ACB.
11. Return to the caller.

Notes for Diagram AE1

The input is from IFG0231T.

2 IDA0231T: INIT231T, GETCORE

The module work area and the close work area are built.

The dummy DEB is verified. Unless a dummy data set is being closed, IDA0231T (ENQFUNC, ENQINIT, PARMINIT) builds an ENQ parameter list and issues ENQ for every data set associated with the user ACB. The parameter list indicates 'SYSVSAM' as the major resource and control-interval number of the data set, catalog ACB address, and 'B' (busy) as the minor resource.

4 IDA0231T: FLQUIS

If the CLOSE (TYPE=T) isn't for restart or checkpoint with local shared resources, the data set is flushed (that is, any I/O activity yet to be done or already started is done):

An inner RPL is built and pointed to the user ACB. The PLH chain is searched for PLHs connected to the user ACB. The inner RPL is connected to each PLH and a FRCIO macro is issued. No record is returned for an incomplete input request (GET or POINT). The output buffer is written to the VSAM data set for an incomplete output request (PUT or ERASE). After I/O completes, the inner RPL is freed.

5 IDA0231T: TCLSPATH calls **IDA0231B**

The alternate index in a path is closed (TYPE=T) before the base cluster. (See diagram AE2.)

7 IDA0231T: TCLSBASE calls **IDA0231B**

The cluster being closed (TYPE=T) can be a base cluster (part of a path), a cluster that wasn't processed through a path, or an alternate index that was itself processed by the user. (See Diagram AE2.)

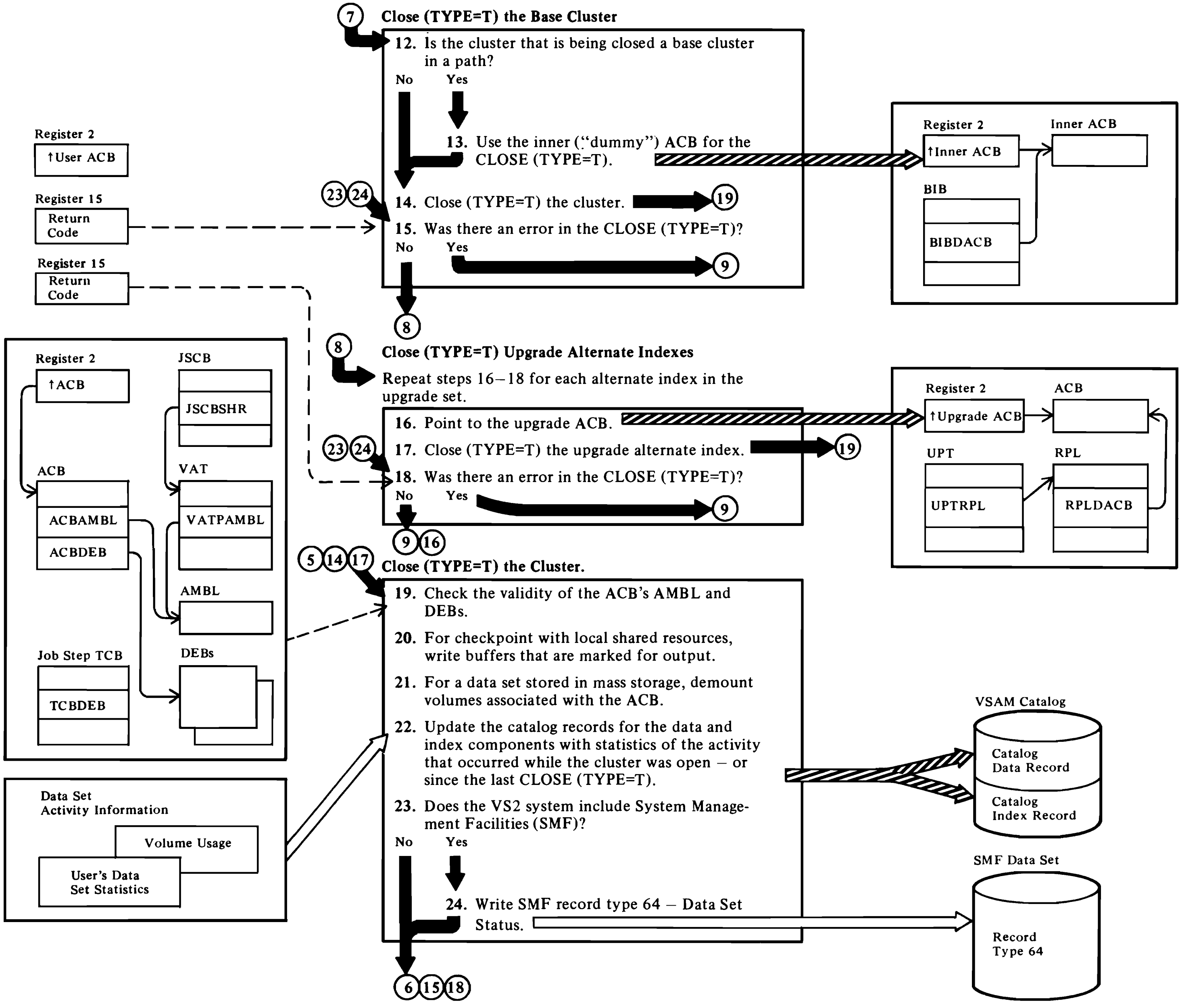
8 IDA0231T: TCLSUPGR calls **IDA0231B**

(See Diagram AE2.)

9 IDA0231T: FREECORE

The storage for the close work area and the module work area is freed.

Diagram AE2. VSAM CLOSE (TYPE=T)



Close (TYPE=T) the Base Cluster

12. Is the cluster that is being closed a base cluster in a path?
    No Yes
13. Use the inner ("dummy") ACB for the CLOSE (TYPE=T).
14. Close (TYPE=T) the cluster.
15. Was there an error in the CLOSE (TYPE=T)?
    No Yes

Close (TYPE=T) Upgrade Alternate Indexes

Repeat steps 16–18 for each alternate index in the upgrade set.

16. Point to the upgrade ACB.
17. Close (TYPE=T) the upgrade alternate index.
18. Was there an error in the CLOSE (TYPE=T)?
    No Yes

Close (TYPE=T) the Cluster.

19. Check the validity of the ACB's AMBL and DEBs.
20. For checkpoint with local shared resources, write buffers that are marked for output.
21. For a data set stored in mass storage, demount volumes associated with the ACB.
22. Update the catalog records for the data and index components with statistics of the activity that occurred while the cluster was open – or since the last CLOSE (TYPE=T).
23. Does the VS2 system include System Management Facilities (SMF)?
    No Yes
24. Write SMF record type 64 – Data Set Status.

Notes for Diagram AE2

Close (TYPE=T) the Base Cluster

The cluster being closed (TYPE=T) can be a base cluster that was being processed through a path, a cluster that was *not* being processed through a path, or an alternate index that was itself processed by the user.

13 IDA0231T: TCLSBASE

The inner ACB is used because the user ACB contains parameters for closing a path, not for closing a base cluster.

14 IDA0231T: TCLSBASE calls **IDA0231B**

Close (TYPE=T) Upgrade Alternate Indexes

16 IDA0231T: TCLSUPGR

After the last upgrade alternate index is closed (TYPE=T), register 2 is pointed back to the user ACB.

17 IDA0231T: TCLSUPGR calls **IDA0231B**

Close (TYPE=T) the Cluster

19 IDA0231B: INIT200B, VALCHECK, PROBDT (calls **IDA0192P), ERRORFLG**

The DEBCHK SVC is used to check the validity DEBs.

20 IDA0231B: WRITBUFR, GETCORE, WRBUFFER, FREECORE, PROBDT, ERRORFLG

Inner control blocks are built and the WRTBFR macro is issued to write data still in buffers.

21 IDA0231B: VIRTPROC calls **IDA0192D**

IDA0192D destages data from the direct-access storage staging drive to mass storage.

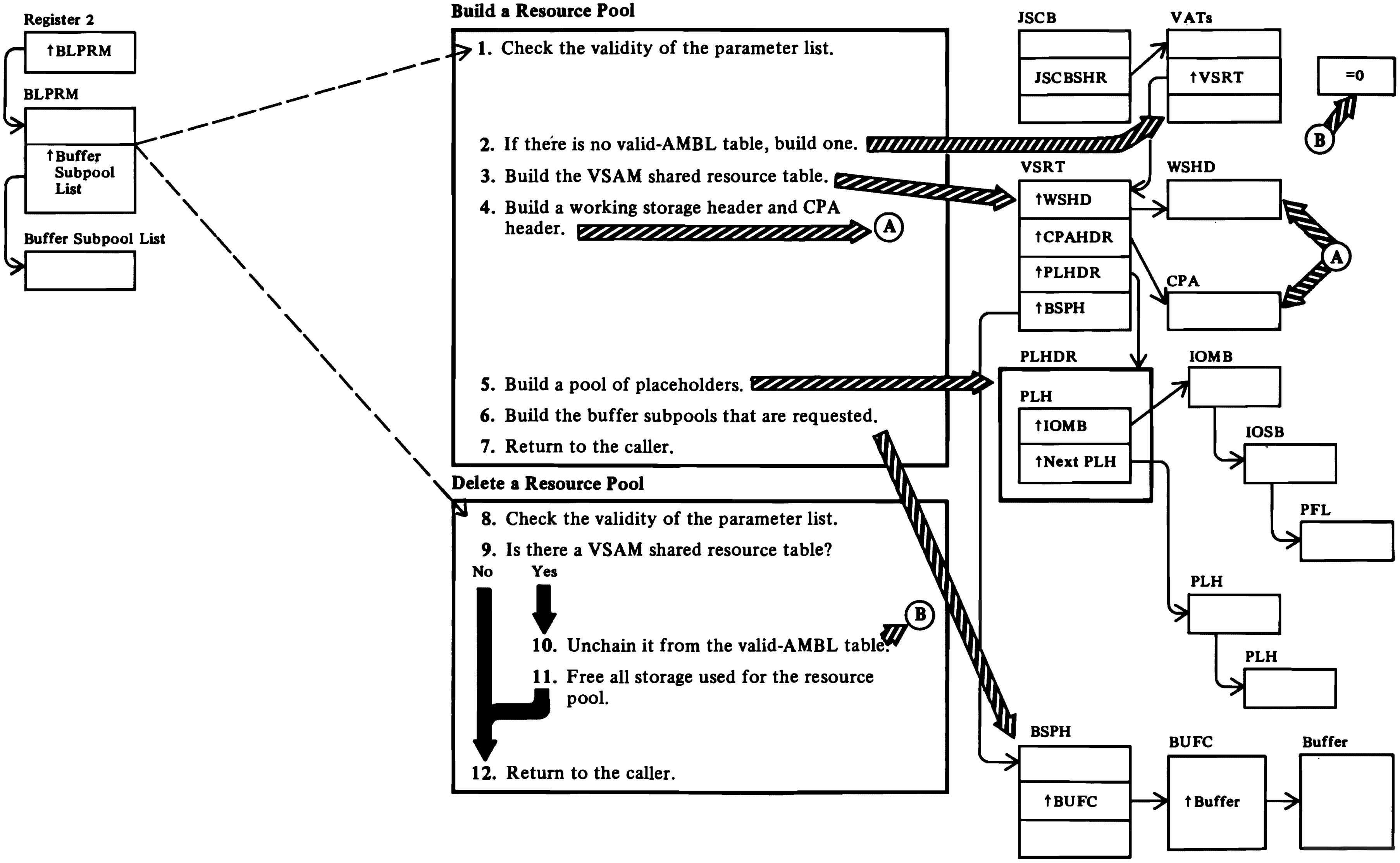
22 IDA0231B: UPCATACB, UPCATDEQ (calls **IDA0192C), PROBDT, ERRORFLG**

24 IDA0231B: UPSMF calls **IDA0192S**

One SMF record type 64 is written for each AMB (for data set or index) connected to the ACB's AMBL.

See *OS/VS2 System Programming Library: System Management Facilities (SMF)* for a description of SMF record type 64—Data Set Status.

Diagram AF. BLDVRP/DLVRP: Build or Delete a VSAM Resource Pool



Build a Resource Pool

1. Check the validity of the parameter list.
2. If there is no valid-AMBL table, build one.
3. Build the VSAM shared resource table.
4. Build a working storage header and CPA header.
5. Build a pool of placeholders.
6. Build the buffer subpools that are requested.
7. Return to the caller.

Delete a Resource Pool

8. Check the validity of the parameter list.
9. Is there a VSAM shared resource table?
No Yes
10. Unchain it from the valid-AMBL table.
11. Free all storage used for the resource pool.
12. Return to the caller.

Notes for Diagram AF

BLDVRP

1 IDA0192Y: DBDCVAL

BLPRM is the BLDVRP parameter list. There must be no conflicting parameters, and buffer sizes must be valid.

2 IDA0192Y: BLDVAT

3 IDA0192Y: BLDVSRT

The VSAM shared resource table is initialized to receive pointers in subsequent processing. The control block structure for processing with shared resources is illustrated in "Control Block Interrelationships" in "Data areas."

4 IDA0192Y: BLDWSHD

5 IDA0192Y: INITPLHP

6 IDA0192Y: BLDBUFC

IDA0192Y: BLDVRP

The address of the VSAM shared resource table is put into the valid-AMBL table. If this chaining couldn't be done, the DLVRP procedure gets control to delete the resource pool.

DLVRP

8 There must be no conflicting parameters and no ACBs open to use the resource pool. If an ACB is open to use it, the DLVRP is rejected.

9 If DLVRP is issued without a previous BLDVRP, there is no VSAM shared resource table.

10 IDA0192Y: DELVRP

11 IDA0192Y: FREEVSRT

Forced Deletion of a Global Resource Pool

When the task that issues BLDVRP to build a global resource pool terminates without issuing DLVRP, VS2 forces deletion of the resource pool. Forced deletion of a global resource pool is described in Diagram AH and in "Recovery with Global Shared Resources" in "Diagnostic Aids."

IDAOCEA4—BLDVRP/DLVRP ESTAE Routine

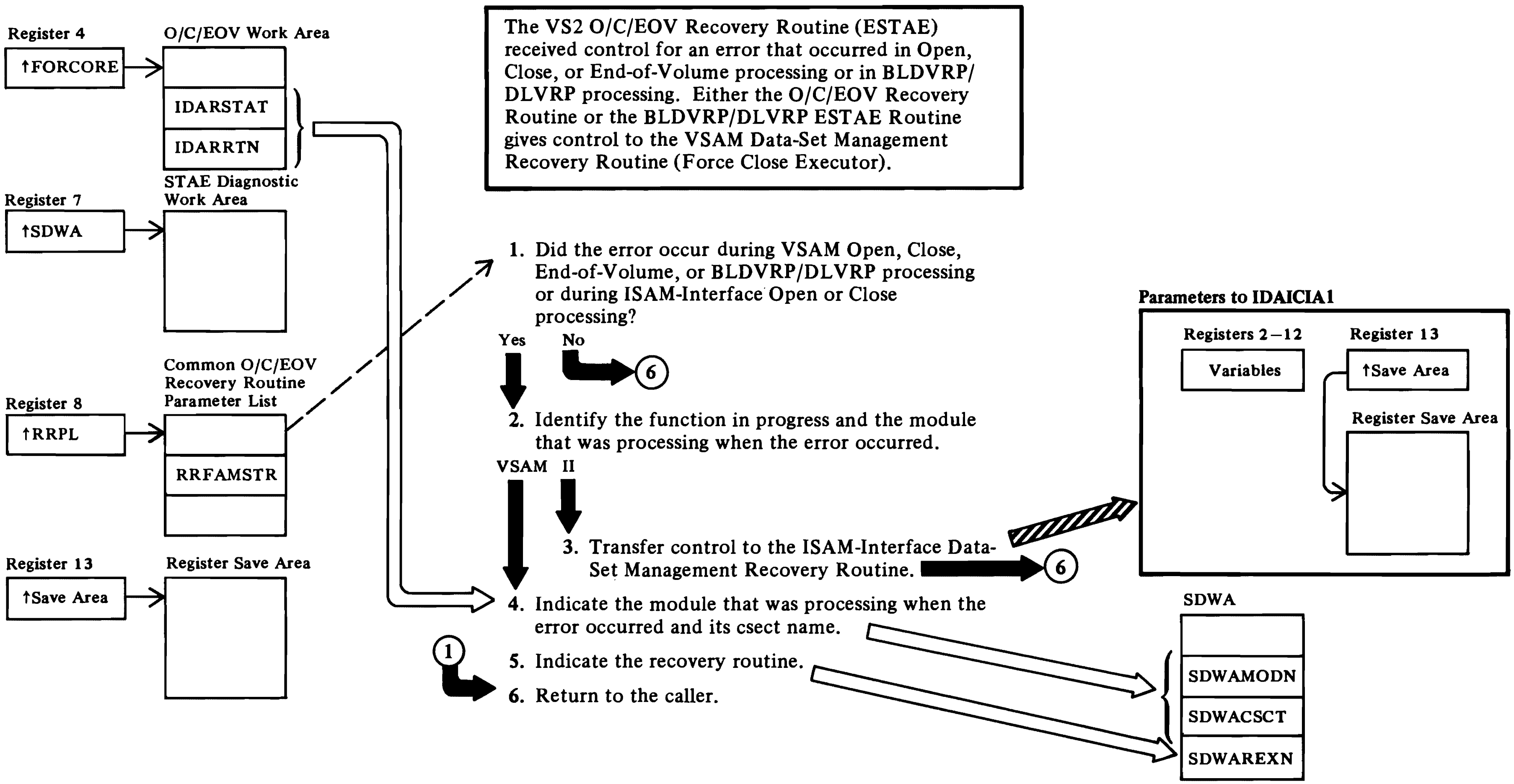
IDAOCEA4 is a csect in load module IFG0192A. The VS2 Recovery Termination Manager (an ESTAE routine, also called the VS2 I/O Support Recovery Routine) receives control for an error that occurs in BLDVRP/DLVRP processing and gives control to IDAOCEA4 for a program check, SVC 13, or abnormal termination.

Unless an SDWA (STAE diagnostic work area, also called RTCA—recovery termination communication area) is passed, IDAOCEA4 returns to the Recovery Termination Manager without processing.

SDWA contains the address of a pseudo FORCORE area that was built during BLDVRP/DLVRP processing. IDAOCEA4 uses portions of FORCORE for recovery.

IDAOCEA4 passes control to IDAOCEA1 for error recording (see Diagram AG). IDAOCEA4 initializes registers for transfering control. Register 8 contains the address of a dummy common O/C/EOV Recovery Routine parameter list (mapped by IECRRPL) in the pseudo FORCORE. Register 13 contains the address of a register save area in the pseudo FORCORE.

Diagram AG. Recovery Termination for Open, Close, and End of Volume



The VS2 O/C/EOV Recovery Routine (ESTAE) received control for an error that occurred in Open, Close, or End-of-Volume processing or in BLDVRP/DLVRP processing. Either the O/C/EOV Recovery Routine or the BLDVRP/DLVRP ESTAE Routine gives control to the VSAM Data-Set Management Recovery Routine (Force Close Executor).

1. Did the error occur during VSAM Open, Close, End-of-Volume, or BLDVRP/DLVRP processing or during ISAM-Interface Open or Close processing?
2. Identify the function in progress and the module that was processing when the error occurred.
3. Transfer control to the ISAM-Interface Data-Set Management Recovery Routine.
4. Indicate the module that was processing when the error occurred and its csect name.
5. Indicate the recovery routine.
6. Return to the caller.

Notes for Diagram AG

The VSAM Data-Set Management Recovery Routine (IDAOCEA1) runs under the direct control of either the VS2 O/C/EOV Recovery Routine or the BLDVRP/DLVRP ESTAE Routine (IDAOCEA4), each of which is an ESTAE routine. IDAOCEA1 causes logging of information that indicates the processing that preceded the error. See "Open, Close, and End-of-Volume Diagnostics" in "Diagnostic Aids" for a discussion of dumps associated with errors in Open, Close, and End of Volume.

1 **IDAOCEA1**

RRFAMSTR equal to 0 indicates that neither VSAM nor ISAM-Interface processing was involved in the error.

3 **IDAOCEA1** calls **IDAICIA1**

IDAICIA1 frees the ISAM-Interface work areas when the error cannot be recovered from and records information in the SYS1.DUMP of the SYSABEND data set. IDAICIA1 is discussed further in "Open, Close, and End-of-Volume Diagnostics" in "Diagnostic Aids."

4 **IDAOCEA1**

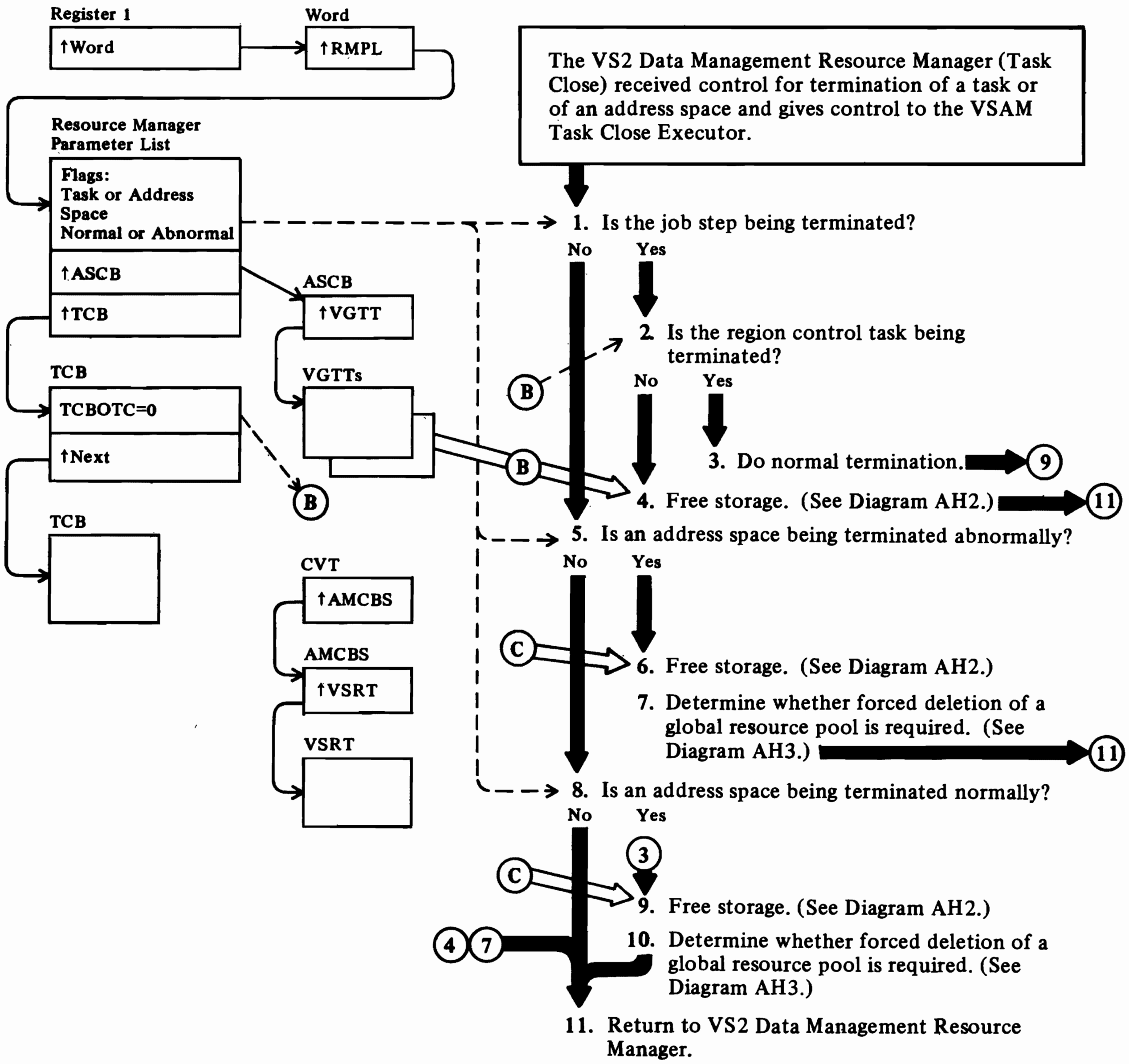
If the name of the module or of the csect can't be determined, the name 'IDA0192X' is indicated. If the error occurred while control was being passed between IFG0192A and the main load module IDA0192A, the name "IDA0ƀƀƀƀ" is indicated.

The indicators set in SDWA are discussed in detail in "Open, Close, and End-of-Volume" Diagnostics in "Diagnostic Aids."

5 **IDAOCEA1**

'IDAOCEA1' is indicated, unless it received control from IDAOCEA4, in which case 'IDAOCEA4' is indicated. (IDAOCEA4 is described in Notes for Diagram AF.)

Diagram AH1. Recovery Termination: VSAM Task Close Executor



The VS2 Data Management Resource Manager (Task Close) received control for termination of a task or of an address space and gives control to the VSAM Task Close Executor.

1. Is the job step being terminated?
2. Is the region control task being terminated?
3. Do normal termination.
4. Free storage. (See Diagram AH2.)
5. Is an address space being terminated abnormally?
6. Free storage. (See Diagram AH2.)
7. Determine whether forced deletion of a global resource pool is required. (See Diagram AH3.)
8. Is an address space being terminated normally?
9. Free storage. (See Diagram AH2.)
10. Determine whether forced deletion of a global resource pool is required. (See Diagram AH3.)
11. Return to VS2 Data Management Resource Manager.

Notes for Diagram AH1

The VSAM Task Close Executor (IDAOCEA2) gets control from the VS2 Data Management Resource Manager (IFG0TC0A, also called VS2 Task Close) for normal or abnormal termination of a task or of an address space, including "out-of-core" ABEND.

1 **IDAOCEA2**

2 **IDAOCEA2: JSTERM**

RMPLTCBA gives the location of the terminating TCB. TCBOTC indicates whether the region control task is being terminated.

3 **IDAOCEA2: JSTERM** calls **NMEMTERM**

4 **IDAOCEA2: JSTERM** calls **FALLVGTT**

5 **IDAOCEA2**

6 **IDAOCEA2: AMEMTERM** calls **FALLVGTT**

7 **IDAOCEA2: AMEMTERM** calls **SCANGSR**

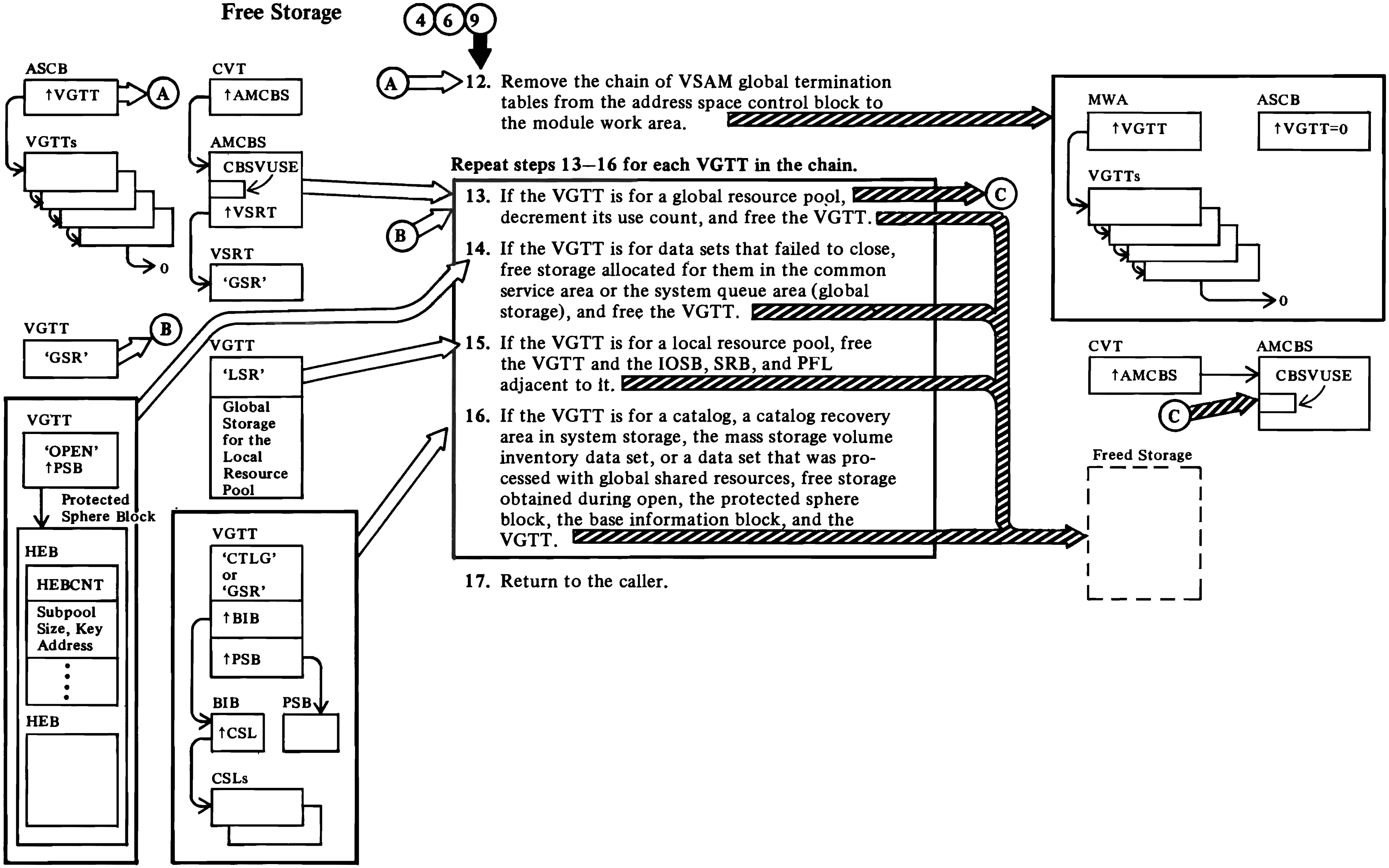
8 **IDAOCEA2**

9 **IDAOCEA2: NMEMTERM** calls **FALLVGTT**

10 **IDAOCEA2: NMEMTERM** calls **SCANGSR**

Diagram AH2. Recovery Termination: VSAM Task Close Executor

Free Storage



12. Remove the chain of VSAM global termination tables from the address space control block to the module work area.

Repeat steps 13–16 for each VGTT in the chain.

13. If the VGTT is for a global resource pool, decrement its use count, and free the VGTT.

14. If the VGTT is for data sets that failed to close, free storage allocated for them in the common service area or the system queue area (global storage), and free the VGTT.

15. If the VGTT is for a local resource pool, free the VGTT and the IOSB, SRB, and PFL adjacent to it.

16. If the VGTT is for a catalog, a catalog recovery area in system storage, the mass storage volume inventory data set, or a data set that was processed with global shared resources, free storage obtained during open, the protected sphere block, the base information block, and the VGTT.

17. Return to the caller.

Notes for Diagram AH2

12 IDAOCEA2: FALLVGTT

The pointer in the address-space control block to the first VGTT in the chain of VGTTs is removed. MWANVGTT is pointed to the first VGTT to make a local VGTT chain.

In processing each VGTT in the chain (steps 13-16), it is made the current VGTT by pointing MWANVGTT to the next one, until there is no next one (in which case MWANVGTT is set to 0).

13 IDAOCEA2: FALLVGTT calls **VDECHAIN,** which calls **DECGVSRT**

If the AMCBS VSRT use count isn't 0, it is decremented by the amount in the current VGTT. If the use count becomes negative, it is set to 0.

14 IDAOCEA2: FALLVGTT calls **FOPEN,** which calls **FEEECORE**

A data set may not be closed because it was only partially opened or End of Volume or Close failed. The header elements in header element blocks describe storage that has been obtained for each data sets. "Virtual-Storage Management" in "Diagnostic Aids" describes HEBs and indicates what subpools contain each type of control block.

FOPEN uses the VS2 GDT (global data area) to determine the address boundaries of global storage. If there is a protected sphere block, FOPEN processes each header element in it, using HEBCNT as an index. If the storage indicated in a header element is within the boundaries of global storage and in subpool 231, 239, 241, or 245, FOPEN uses its key to free it. After all HEBs are processed, FOPEN frees the protected sphere block and the VGTT.

15 IDAOCEA2: FALLVGTT calls **FLSR,** which calls **FREECORE**

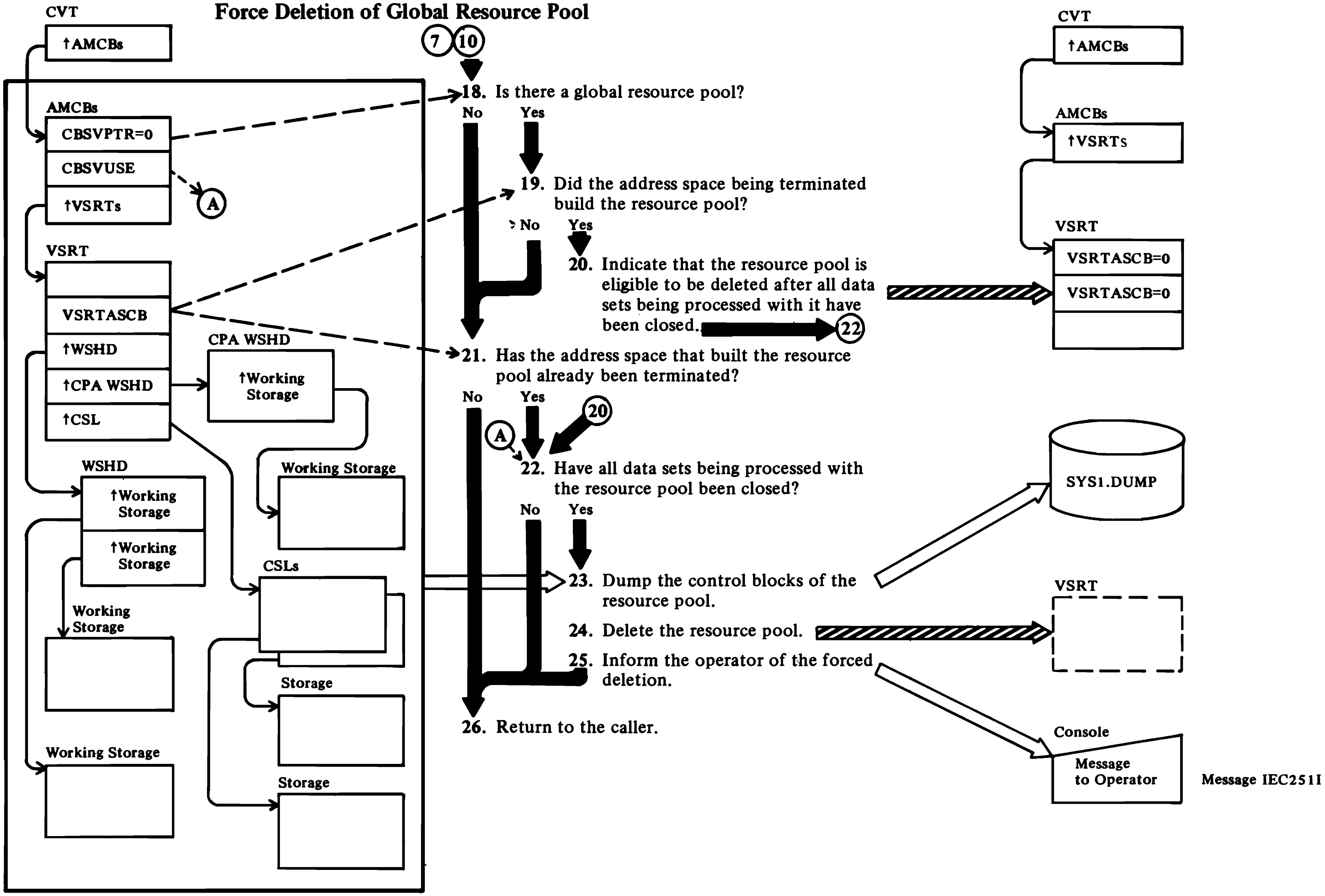
When a local resource pool is built (during BLDVRP processing), storage in the system queue area is obtained for each trio of IOSB-SRB-PFL, and a VGTT is prefixed to each.

16 IDAOCEA2: FALLVGTT calls **FCTLG**

At the end of Open processing for a catalog, a catalog recovery area in system storage, or the mass storage volume inventory data set, Open frees the VGTT, so IDAOCEA2 cleans up for these only for termination that occurs during Open processing.

Diagram AH3. Recovery Termination: VSAM Task Close Executor

Force Deletion of Global Resource Pool



18. Is there a global resource pool?

19. Did the address space being terminated build the resource pool?

20. Indicate that the resource pool is eligible to be deleted after all data sets being processed with it have been closed.

21. Has the address space that built the resource pool already been terminated?

22. Have all data sets being processed with the resource pool been closed?

23. Dump the control blocks of the resource pool.

24. Delete the resource pool.

25. Inform the operator of the forced deletion.

26. Return to the caller.

Notes for Diagram AH3

The address space that built the global resource pool, if there is one, is responsible for deleting it. If that address space terminates without deleting the resource pool, the global storage used for it would be lost to the system, unless a recovery routine forced its deletion. "Recovery with Global Shared Resources" in "Diagnostic Aids" further discusses forced deletion and the dumping of control blocks for the resource pool.

18 IDAOCEA2: SCANGSR

19 IDAOCEA2: SCANGSR

VSRTASCB equal to the ASCB address of the address space being terminated indicates that the address space built the global resource pool.

20 IDAOCEA2: SCANGSR

22 IDAOCEA2: SCANGSR

CBSVUSE equal to zero indicates that all data sets processing with the global resource pool have been closed.

23 IDAOCEA2: SCANGSR calls **FDLVRP**

FDLVRP calls **GSRDUMP,** which calls **SDLOAD** (which calls **SDUMP)**

GSRDUMP passes to SDLOAD the address and length of each control block or storage area to be dumped. SDLOAD saves each return code from SDUMP; FDLMSG (step 25) examines the return codes to determine what return code to include in the message to the operator. (The return code indicates whether all, some, or no areas were dumped.)

(Areas dumped and messages are discussed in "Recovery with Global Shared Resources" in "Diagnostic Aids.")

24 FDLVRP calls **IDA0192Y**

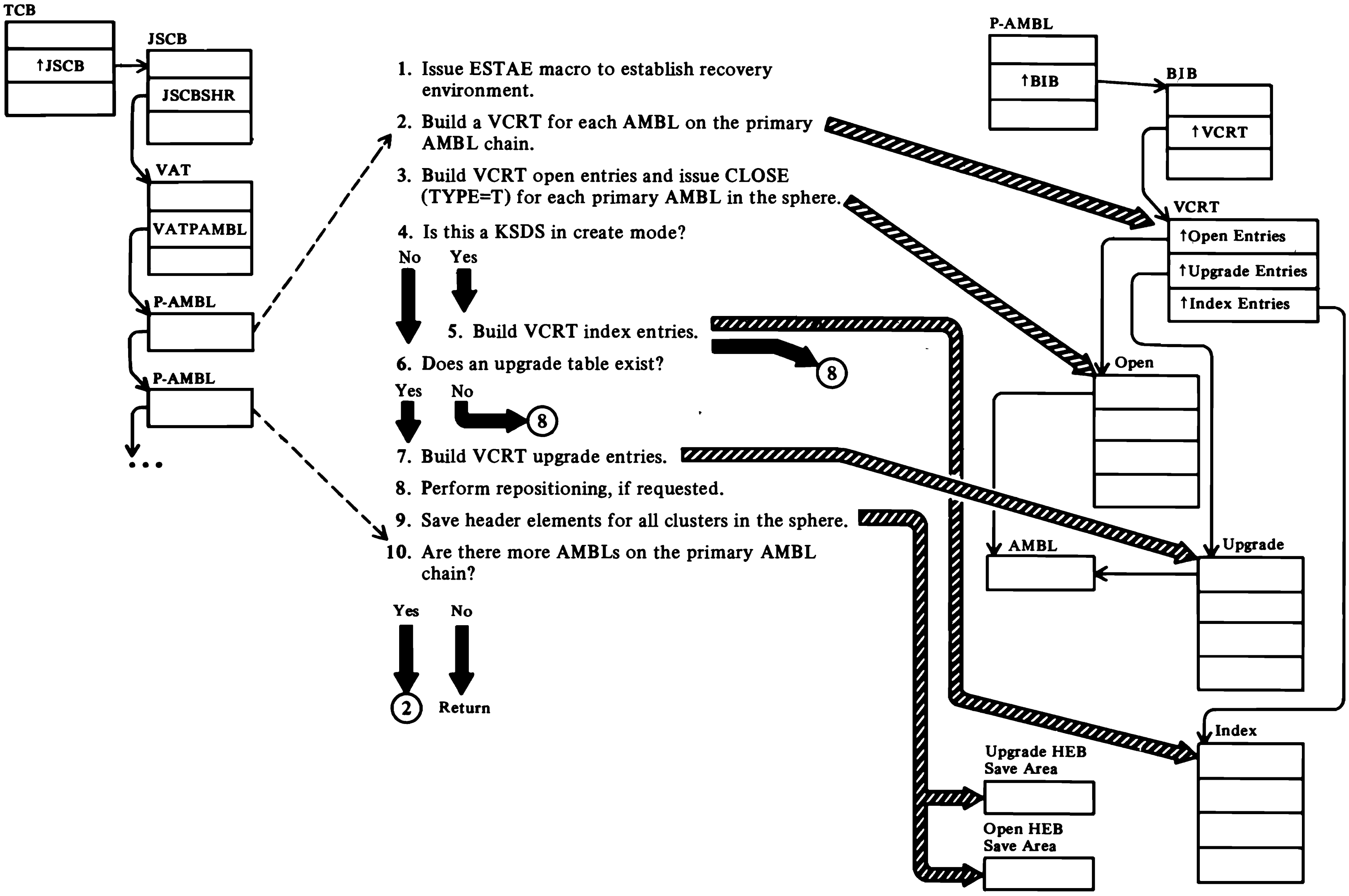
IDA0192Y deletes the global resource pool.

25 FDLVRP calls **FDLMSG,** which calls **IDA0192P**

See note about FDLMSG in step 23.

IDA0192P suppresses the GTF trace and the construction of a message area.

Diagram AI. VSAM Checkpoint: Build the VSAM Checkpoint/Restart Record



1. Issue ESTAE macro to establish recovery environment.
2. Build a VCRT for each AMBL on the primary AMBL chain.
3. Build VCRT open entries and issue CLOSE (TYPE=T) for each primary AMBL in the sphere.
4. Is this a KSDS in create mode?

   No → 6. Yes → 5.

5. Build VCRT index entries. → 8
6. Does an upgrade table exist?

   Yes → 7. No → 8

7. Build VCRT upgrade entries.
8. Perform repositioning, if requested.
9. Save header elements for all clusters in the sphere.
10. Are there more AMBLs on the primary AMBL chain?

    Yes → 2 No → Return

Notes for Diagram AI

When IHJACP02 (IGC0206C) receives control after the caller issues the CHKPT macro (SVC 63), the DEB chain is scanned for a VSAM DEB. If any VSAM data sets are open, IGC0206C loads and branches to IDA0C06C. Standard linkage conventions are observed and register 1 contains the address of the checkpoint work area. IDA0C06C obtains a module work area in subpool 252 and a work area for I/O in subpool 250.

1 **IDA0C06C: RECOVERY**

An ESTAE macro is issued to establish IDACKRA1 as the ESTAE routine. The parameter area to be passed to IDACKRA1 is initialized. This parameter area consists of fields in a dummy Open work area used for compatibility with VSAM open modules. The fields in the work area are DXATCOM1 (base register), DXATCOM2 (data register), DXATCOM3 (address of retry routine), DXATCOM4 (address of checkpoint/restart work area), DXATEXC1 (status information), and DXATEXC2 (last four characters of module in control).

2 **IDA0C06C: BLDVCRT, GETCORE**

BLDVCRT calls GETCORE, specifying the minimum storage required to build the VCRT but requesting the maximum storage available for suballocation from a 4K block in subpool 252. GETCORE returns the length actually obtained.

The VCRT header is initialized and BLDOPEN is called.

3 **IDA0C06C: BLDOPEN, TCLOSE**

Build an Open entry for every primary AMBL in the sphere. Search the secondary chain for each primary AMBL to determine if an ACB was opened that required a higher level of authorization. The address of the AMBL with the highest level of authorization is replaced in the Open entry. A CLOSE (TYPE=T) macro is issued against the corresponding ACB if the AMBL is not the base AMBL accessed through a path. The maximum control interval size for the Open entries is set in the VCRT.

4 **IDA0C06C: BLDVCRT**

If AMBCREAT is on in the AMB and if AMDDST (a KSDS) is on in the AMDSB, build the VCRT index entries.

5 **IDA0C06C: BLDINDEX, IDXGET**

Build an index entry for every Index Create Work Area (ICWA) that is in use. The index GET function of record management is used to save the control interval for each ICWA in use.

6 **IDA0C06C: BLDVCRT**

If BIBUPT in the BIB is nonzero, build the VCRT upgrade entries.

7 **IDA0C06C: BLDUPGRD**

Build an upgrade entry for every Upgrade Table (UPT) entry. For each upgrade entry, set the VCRCISIZ field (initially set by BLDOPEN) to the CI size for the upgrade cluster or the current maximum CI size, whichever is greater.

8 **IDA0C06C: BLDVCRT, GETRTN**

If the cluster was not open for create mode and it is an entry-sequenced data set with repositioning requested, the current control interval is saved. The repositioning request will not be honored if the cluster is not open for output. The checkpoint will fail if an upgrade path exists and repositioning was requested.

9 **IDA0C06C: HEBSAVE, BLDHEBS**

For every cluster in the sphere, save all of the header elements with the exception of the DEB block , the protected string block, and the protected upgrade string block. The address of the save area is placed in the VCRT upgrade entries and open entries, unless there is a corresponding upgrade entry for the cluster.

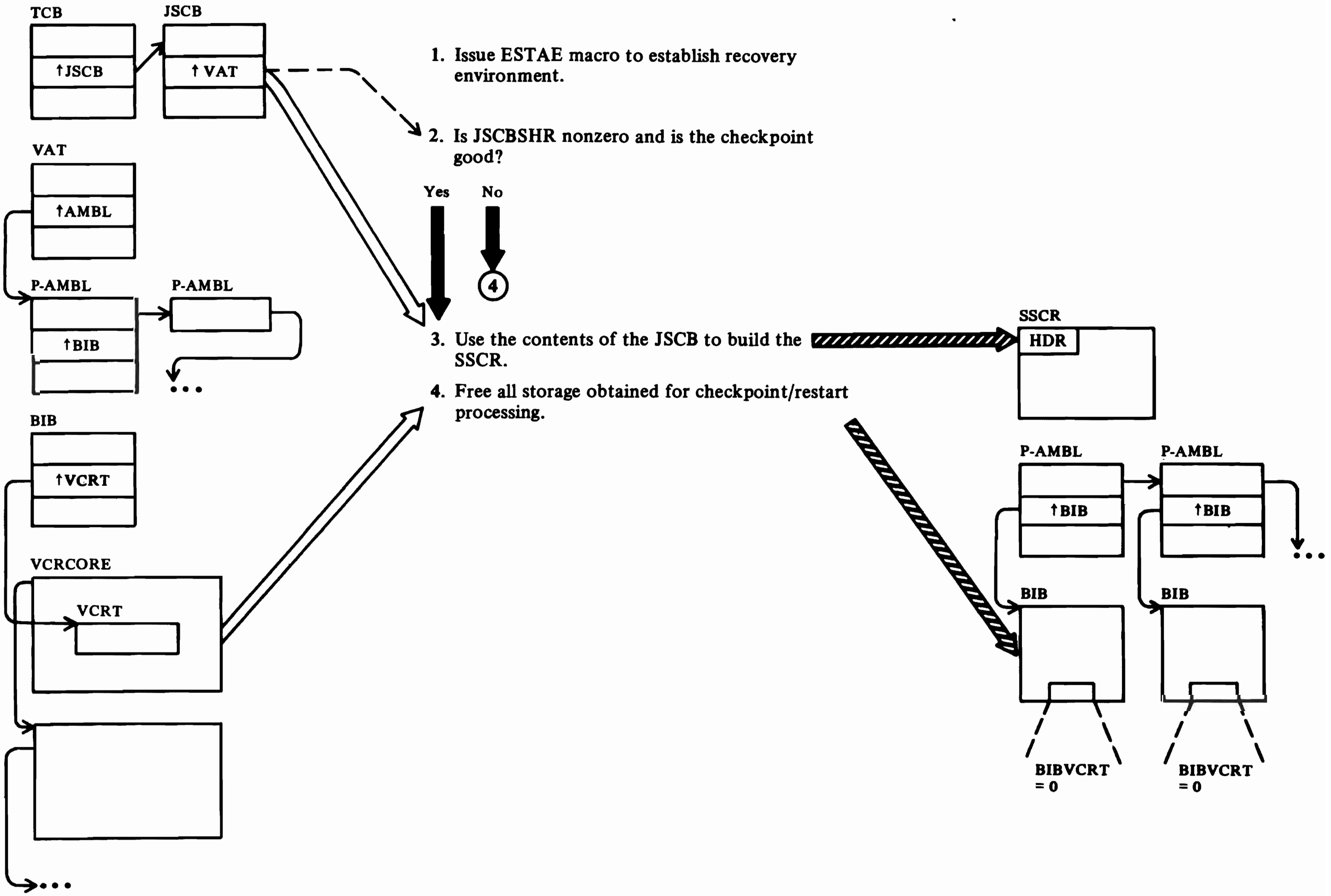
If one or more sphere blocks exist, save the header elements which are chained off BIBSPHPT.

If the sphere was opened with the LSR option, save only the EDB block header elements.

10 **IDA0C06C**

If there are no more AMBLs on the primary AMBL chain, module work areas are freed, registers are reloaded, and control is returned to IGC0206C.

Diagram AJ. VSAM Checkpoint: Build the Subsystem Checkpoint Record



1. Issue ESTAE macro to establish recovery environment.

2. Is JSCBSHR nonzero and is the checkpoint good?

3. Use the contents of the JSCB to build the SSCR.

4. Free all storage obtained for checkpoint/restart processing.

Notes for Diagram AJ

IGC0N06C is given control after the Core Image Records (CIRs) have been written to the checkpoint data set. IDA0I96C is loaded and branched to unconditionally. Standard linkage conventions are used, and register 1 contains the address of the checkpoint work area.

1 **IDA0I96C: RECOVERY**

Issue an ESTAE macro to establish IDACKRA1 as the ESTAE routine. Initialize the parameter area to be passed to IDACKRA1.

2 If CKRETCOD in the checkpoint work area is not zero or not X'10', bypass building the SSCR.

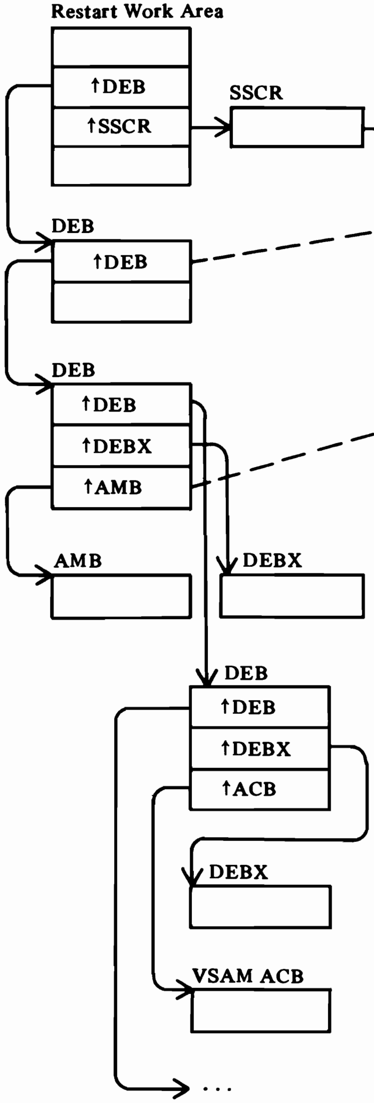
3 **IDA0I96C**

Obtain storage for the VSAM SSCR from subpool 253 and place the address in the checkpoint work area (CKSSCR). Initialize the SSCR header and copy the JSCBSHR field for the data.

4 **IDACI96C: CLEANUP**

IDACI96C is an external entry called for cleanup processing by IDA0C06C (error) and IDA0A05B (normal). It is processed in-line by IDA0I96C (normal or error). If JSCBSHR is nonzero and an AMBL exists which points to a BIB (which in turn points to a VCRT), then the core chain pointed to by the VCRT is freed and all pointers to VCRTs are cleared in all BIBs.

Diagram AK. VSAM Restart: Process Subsystem Checkpoint Records



1. Issue ESTAE macro to establish recovery environment.

2. Validate JSCBSHR in the SSCR and reestablish it in the JSCB.

3. Is the first or next DEB on the DEB chain for a VSAM ACB?

   Yes → 8. No → 4.

4. Is the DEB for an AMB?

   Yes → 6. No → 5.

5. Are there more DEBs?

   Yes → 3 No → Return

6. Free the DEB and the DEB extension.

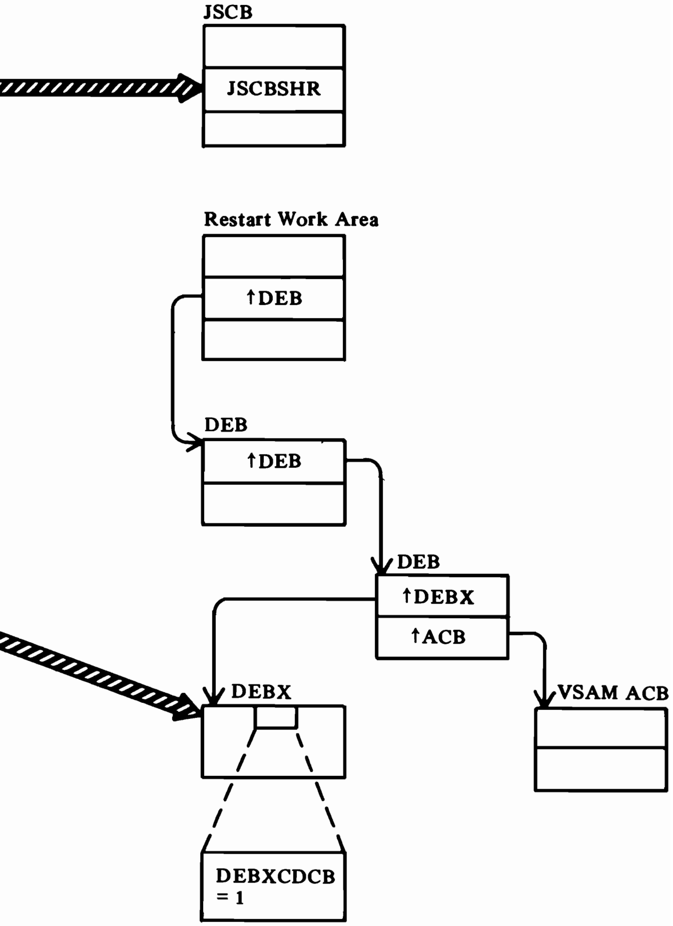
7. Are there more DEBs?

   Yes → 3 No → Return

8. Indicate to task close that the ACB may not be closed.

9. Are there more DEBs?

   Yes → 3 No → Return



Notes for Diagram AK

IDA0C05B restores the address of the VAT in the JSCB, frees the DEBs for AMBs, and marks VSAM ACBs not closeable by task close. IDA0C05B is called from IGC0N05B. Standard linkage conventions are used, and register 1 contains the address of the restart work area.

1 **IDA0C05B: RECOVERY**

The ESTAE macro is issued to establish IDACKRA1 as the ESTAE routine. The parameter area is initialized.

2 **IDA0C05B**

The SSCR and the address of the VAT are validity-checked. The JSCBSHR field is initialized from the SSCR.

3 **IDA0C05B: FIXDEBS, DEBFREE**

The address of the DEB chain is obtained from RSINT in the restart work area. A VSAM DEB is identified by X‘01’ in DEBAMTYP. DEBDCBAD points to an ACB if the first byte contains X‘A0’.

4 DEBDCBAD points to an AMB if the first byte contains X‘40’.

5 If DEBDEBB is zero, the end of the DEB chain has been reached.

6 The DEB extension is pointed to by DEBXTNP (offset - 8). A modeset to key 5 is done to issue the FREEMAIN.

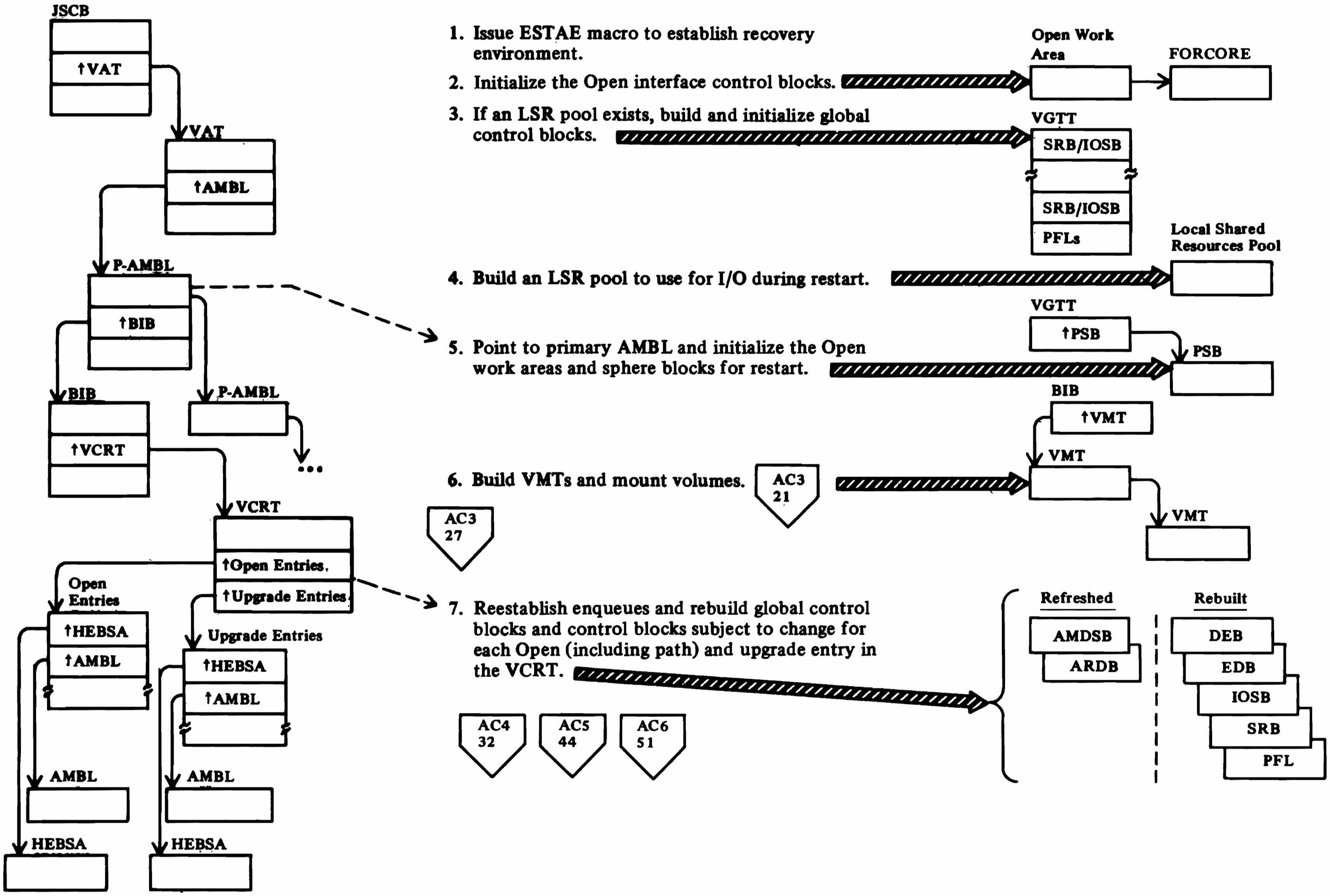
7 If DEBDEBB is zero, the end of the DEB chain has been reached.

8 **IDA0C05B: FIXDEBS**

Set DEBXCDCB to indicate to task close not to close the VSAM ACB.

9 If DEBDEBB is zero, the end of the DEB chain has been reached.

Diagram AL1. VSAM Restart: Rebuild VSAM Control Blocks



1. Issue ESTAE macro to establish recovery environment.
2. Initialize the Open interface control blocks.
3. If an LSR pool exists, build and initialize global control blocks.
4. Build an LSR pool to use for I/O during restart.
5. Point to primary AMBL and initialize the Open work areas and sphere blocks for restart.
6. Build VMTs and mount volumes.
7. Reestablish enqueues and rebuild global control blocks and control blocks subject to change for each Open (including path) and upgrade entry in the VCRT.

Notes for Diagram AL1

IDA0A05B constructs control blocks required by VSAM Open modules and then gives control to those modules to reestablish the control block structure and data set contents as required for repositioning. IDA0A05B is called from IGC0T05B. Standard linkage conventions are used, and register 1 contains the address of the restart work area.

1 **IDA0A05B: RECOVERY**

An ESTAE macro is issued to establish IDACKRA1 as the ESTAE routine. The ESTAE parameter area is initialized.

2 The common Open work area and the VSAM Open work area are initialized. An enqueue parameter list is initialized for locking a cluster during restart Open processing.

3 **IDA0A05B: BLDVSRP, IDA0192Y**

A BLDVRP parameter list is initialized and IDA0192Y is called to build the SRBs, IOSBs, and PFLs in global storage and to chain those blocks to the existing LSR pool.

4 **IDA0A05B: BLDVSRP, IDA0192Y**

After the VATVSRT field is saved and cleared, BLDVRP function is used to build an LSR pool to use for all I/O during restart. One set of buffers is obtained with the buffer size equal to the nearest 4K mutliple higher than the largest CI size for any cluster open in the memory.

5 The Open work areas are initialized with information pertinent to the sphere currently being processed. The VMTs for the checkpointed structure are freed.

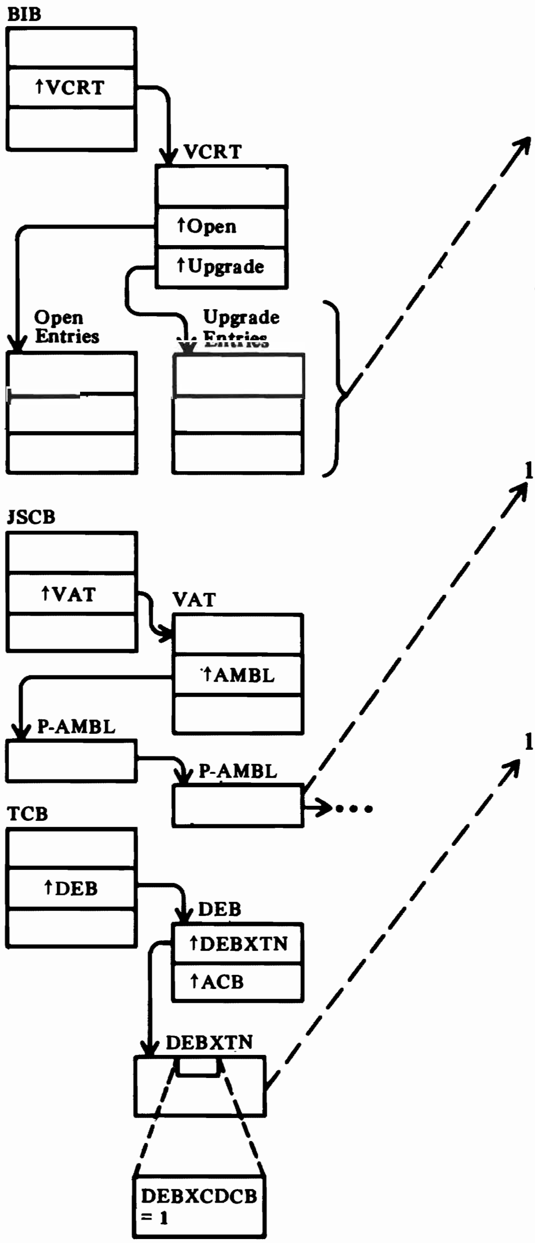
6 For every user ACB open in the sphere, the JFCB is read into the common Open work area and IDA0192F is called to rebuild the VMTs and mount the volumes.

7 **IDA0A05B: BLDOPEN, BLDUPGRD, IDA0192B, IDA0192Z, IDA0192Y, IDA0192C**

Before calling IDA0192B to reestablish enqueues and rebuild (or refresh) the control block structure for a cluster, the volume serial and high-used RBA are saved from each ARDB and the statistics fields are saved from the data (and index) AMDSB.

After returning from the Open modules, the AMDSB statistics are checked for modifications if AMP='CROPS=NRC or NCK' was specified. The fields tested are AMDNLR, AMDIREC, AMDDELR, and AMDUPR.

Diagram AL2. VSAM Restart: Rebuild VSAM Control Blocks



AC4 36 | AC5 47 | AC6 53 | AC6 57

8. For each Open entry, do reposition or verify processing; for each upgrade entry, do verify processing.

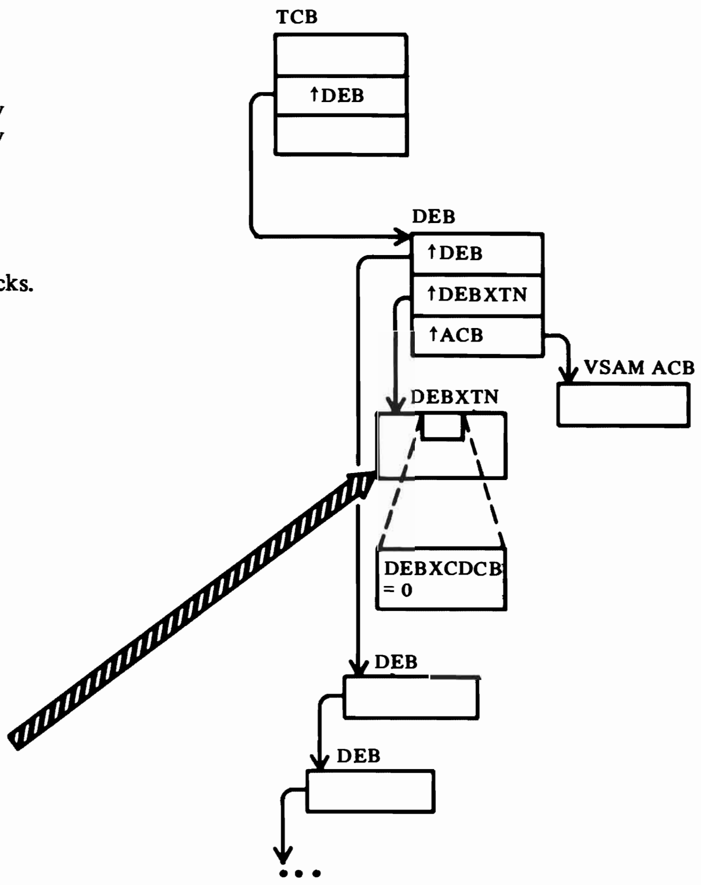
9. Free storage acquired to rebuild control blocks.

10. Are there more AMBLs?

No

Yes → AL1 5

11. Remove checkpoint/restart control blocks, delete the LSR pool, and reset the DEBs for VSAM ACBs.



Notes for Diagram AL2

8 IDA0A05B: REPOSITN, VERIFY

Reposition any data set in create mode. Reposition any entry-sequenced data set unless an associated upgrade path is open or NRE or NRC is specified. Unless the SPEED option was specified when the cluster was defined, preformat the index component for a key-sequenced data set and the data component for an etry-sequenced data set.

If the data set was not eligible for repositioning, SYNCH to record management to set the high-used RBA (VERIFY).

9 Free all EDBs saved from the checkpointed structure by scanning HEB save area headers for VCRHFREL being on. Free all save lists:

The CSL chain is pointed to by BIBCSL.

The DSL, ESL, SSL, and PSL chains are pointed to by their respective fields in the Open work area (OPW).

10 IDA0A05B: FREECORE, IDA0192M

Free all EDBs saved from the checkpoint control block structure. Call IDA0192M to free all excess space in the sphere. Free all save lists.

11 IDA0A05B: DELVSRP, FIXDEBS, IDACI96C, IDA0192Y

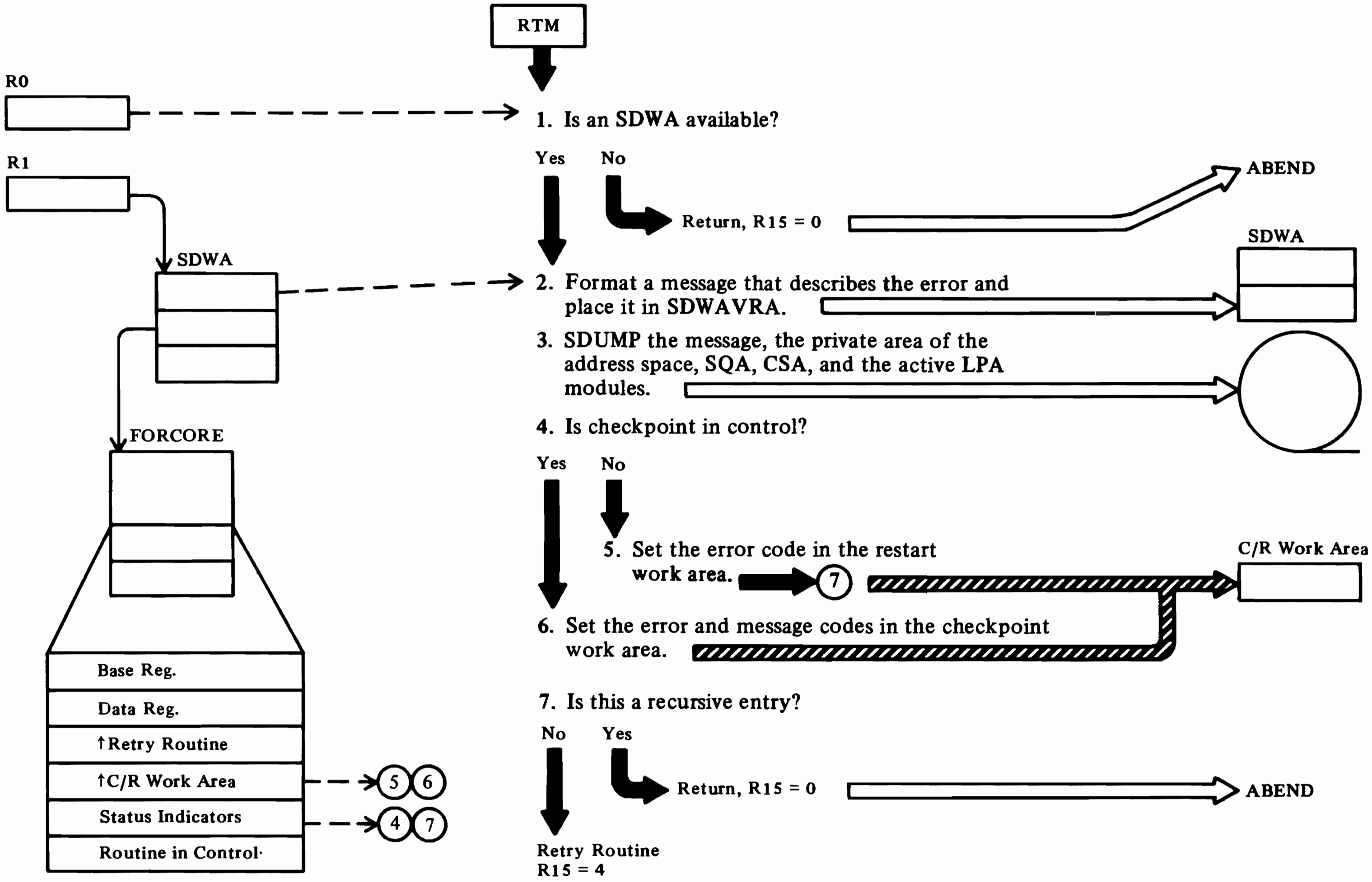
IDA0192Y is called to delete the resource pool used by restart.

The DEBS for the VSAM ACBs are moved to the top of the TCB DEB chain. The DEBXCDCB bit is reset to allow task close to issue a close against the ACB.

IDACI96C is called to clean up the control blocks used by VSAM checkpoint/restart.

If an error occurs during restart, message IHJ009I is issued and VSAM control blocks are not fixed.

Diagram AM. Checkpoint/Restart Recovery Processing



1. Is an SDWA available?

   Yes No

   Return, R15 = 0

2. Format a message that describes the error and place it in SDWAVRA.

3. SDUMP the message, the private area of the address space, SQA, CSA, and the active LPA modules.

4. Is checkpoint in control?

   Yes No

5. Set the error code in the restart work area.

6. Set the error and message codes in the checkpoint work area.

7. Is this a recursive entry?

   No Yes

   Return, R15 = 0

   Retry Routine
   R15 = 4

Notes for Diagram AM

IDACKRA1 performs ESTAE processing for VSAM checkpoint/restart. If an SDWA is available on entry, a message is formatted and an attempt is made to dump storage and exit to a retry routine.

1 If an SDWA is not available (indicated by register 0), control is returned to RTM so abnormal termination can continue.

2 The message below is formatted in the SDWA.

```
                                        ⎧ MACHINE CHECK                           ⎫
                                        ⎪ PROGRAM CHECK LOCATION=nnnnnn           ⎪
VSAM   ⎧ CHECKPOINT ⎫   (IDA0xxxx)      ⎨ RESTART KEY DEPRESSED                   ⎬
       ⎩ RESTART    ⎭                   ⎪ PAGING ERROR                            ⎪
                                        ⎩ ABEND Snnn,Unnn,REGISTER 15=nnnnnnnn    ⎭
```

3 The SDUMP parameter list is moved to the SDWA and the SDUMP macro is issued.

4 If IDARCKPT in the ESTAE parameter list (pseudo FORCORE) is on, checkpoint is in control.

5 The return code, either 241 or 242, is set in the second byte of RSRETCOD.

6 The message code, either 241 or 242, is set in the second byte of CKMSGCOD, and the return code, X'0C', is set in the second byte of CKRETCOD.

7 If IDARCURS in the ESTAE parameter list is on, this is a recursive execution of the ESTAE routine.

Diagram BA. Record Management Table of Contents

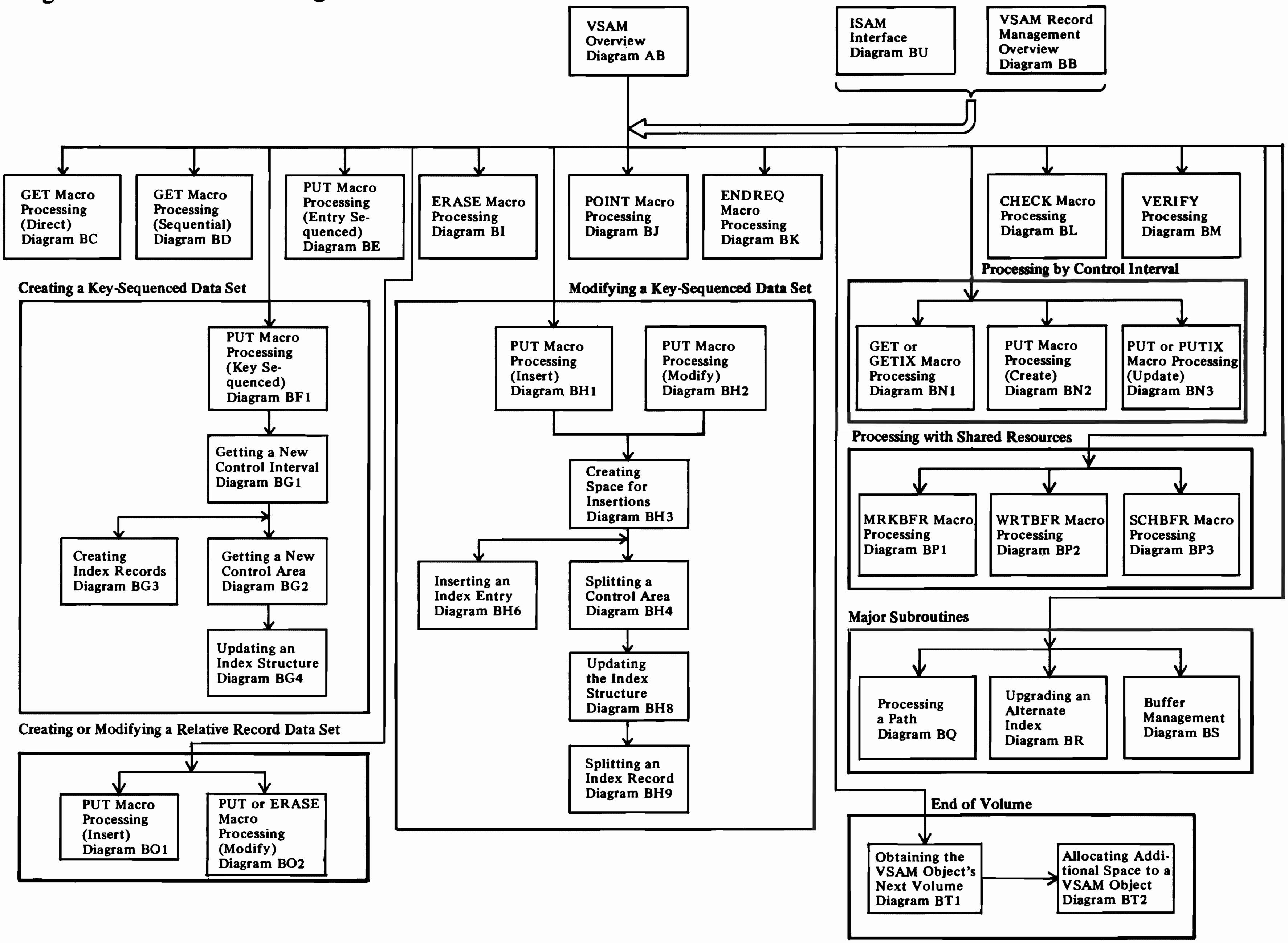
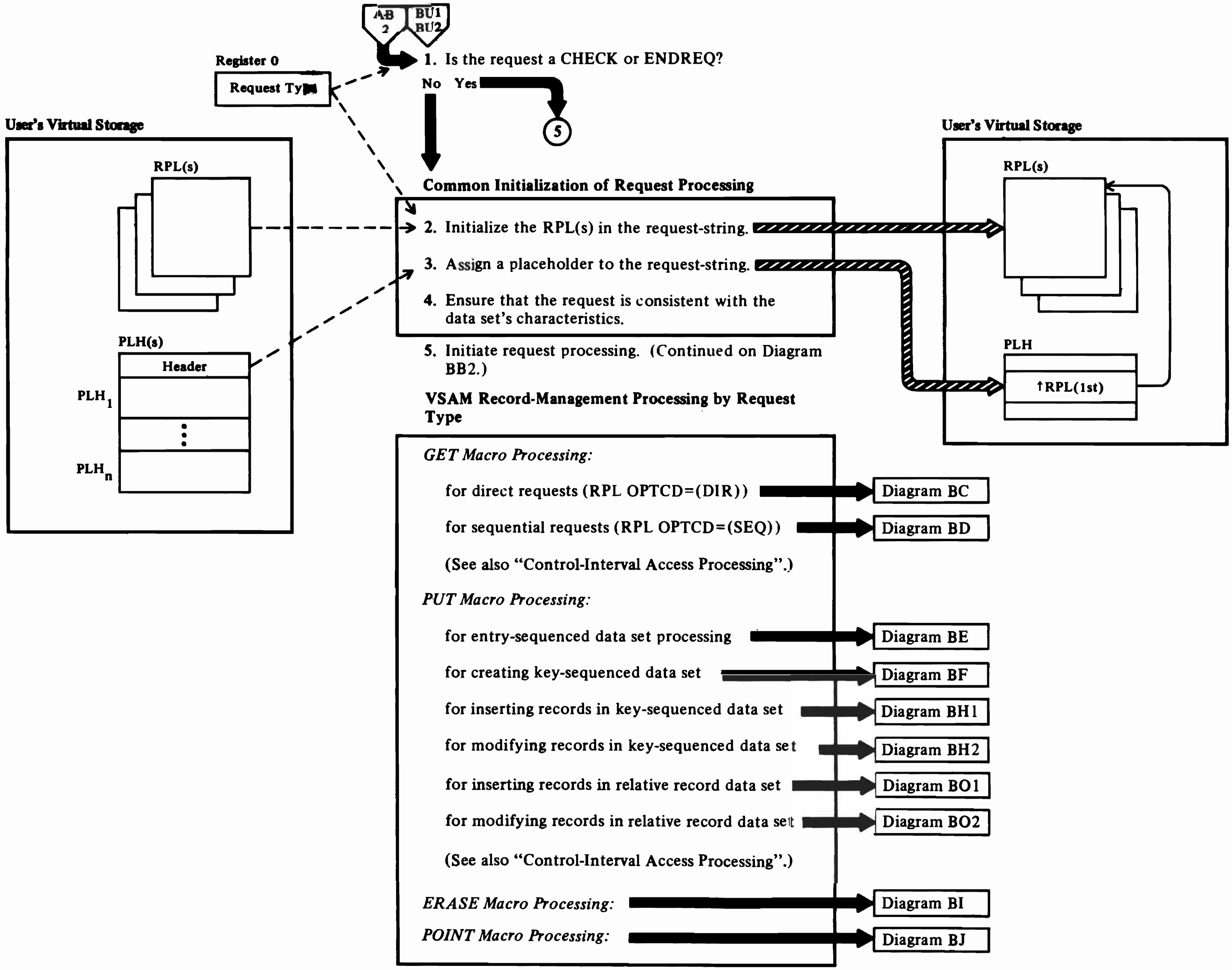

VSAM Overview Diagram AB
ISAM Interface Diagram BU
VSAM Record Management Overview Diagram BB
GET Macro Processing (Direct) Diagram BC
GET Macro Processing (Sequential) Diagram BD
PUT Macro Processing (Entry Sequenced) Diagram BE
ERASE Macro Processing Diagram BI
POINT Macro Processing Diagram BJ
ENDREQ Macro Processing Diagram BK
CHECK Macro Processing Diagram BL
VERIFY Processing Diagram BM
Creating a Key-Sequenced Data Set
PUT Macro Processing (Key Sequenced) Diagram BF1
Getting a New Control Interval Diagram BG1
Creating Index Records Diagram BG3
Getting a New Control Area Diagram BG2
Updating an Index Structure Diagram BG4
Creating or Modifying a Relative Record Data Set
PUT Macro Processing (Insert) Diagram BO1
PUT or ERASE Macro Processing (Modify) Diagram BO2
Modifying a Key-Sequenced Data Set
PUT Macro Processing (Insert) Diagram BH1
PUT Macro Processing (Modify) Diagram BH2
Creating Space for Insertions Diagram BH3
Inserting an Index Entry Diagram BH6
Splitting a Control Area Diagram BH4
Updating the Index Structure Diagram BH8
Splitting an Index Record Diagram BH9
Processing by Control Interval
GET or GETIX Macro Processing Diagram BN1
PUT Macro Processing (Create) Diagram BN2
PUT or PUTIX Macro Processing (Update) Diagram BN3
Processing with Shared Resources
MRKBFR Macro Processing Diagram BP1
WRTBFR Macro Processing Diagram BP2
SCHBFR Macro Processing Diagram BP3
Major Subroutines
Processing a Path Diagram BQ
Upgrading an Alternate Index Diagram BR
Buffer Management Diagram BS
End of Volume
Obtaining the VSAM Object's Next Volume Diagram BT1
Allocating Additional Space to a VSAM Object Diagram BT2

Diagram BB1. VSAM Record Management Overview



1. Is the request a CHECK or ENDREQ?

Common Initialization of Request Processing

2. Initialize the RPL(s) in the request-string.
3. Assign a placeholder to the request-string.
4. Ensure that the request is consistent with the data set's characteristics.
5. Initiate request processing. (Continued on Diagram BB2.)

VSAM Record-Management Processing by Request Type

GET Macro Processing:

for direct requests (RPL OPTCD=(DIR))

for sequential requests (RPL OPTCD=(SEQ))

(See also "Control-Interval Access Processing".)

PUT Macro Processing:

for entry-sequenced data set processing

for creating key-sequenced data set

for inserting records in key-sequenced data set

for modifying records in key-sequenced data set

for inserting records in relative record data set

for modifying records in relative record data se!t

(See also "Control-Interval Access Processing".)

ERASE Macro Processing:

POINT Macro Processing:

Notes for Diagram BB1

2

Several RPLs may be chained together to process more than one record with a single macro request. For example, a GET request associated with a chain of three RPLs returns three records to the user's problem program.

3

The number of placeholders is based on the STRNO parameter in the ACB control block.

Each placeholder is examined to determine whether it is available for assignment to the request string. (Note: Once a placeholder is assigned to a request string, this association is fixed until an ENDREQ macro or a direct request that doesn't require placeholder retention is issued against the RPL at the head of the request string. After the ENDREQ or direct processing is completed, the placeholder is available for reassignment to another request-string.)

When no placeholder is available in the list of placeholders for assignment to a request or request string and resources are being shared in a VSAM resource pool or a previously empty data set is being loaded, an error code is set and a return is made to the caller. Otherwise, IDA019R1 calls IDA019RU (IDAXGLPH) to obtain additional placeholders. If a placeholder is available, its identifier is placed in the RPL associated with the user's macro request.

4

If any of the following restrictions are violated, an error code is set in the associated RPL and the remaining RPLs (if any) in the request string are posted as incomplete:

Keyed Request Errors

Keyed requests against an entry-sequenced data set are not allowed.

Requests based on a generic key must include a specified key-length value.

Specified key lengths may not exceed the maximum key length value defined for a data set.

Addressed Request Errors

An addressed PUT-add request against a key-sequenced data set is not allowed.

An ERASE request against an entry-sequenced data set is not allowed.

An addressed request against a relative record data set is not allowed.

Control Interval Request Errors

Control interval requests may not be issued against a data set unless the data set was opened for control interval processing.

Diagram BB2. VSAM Record Management Overview

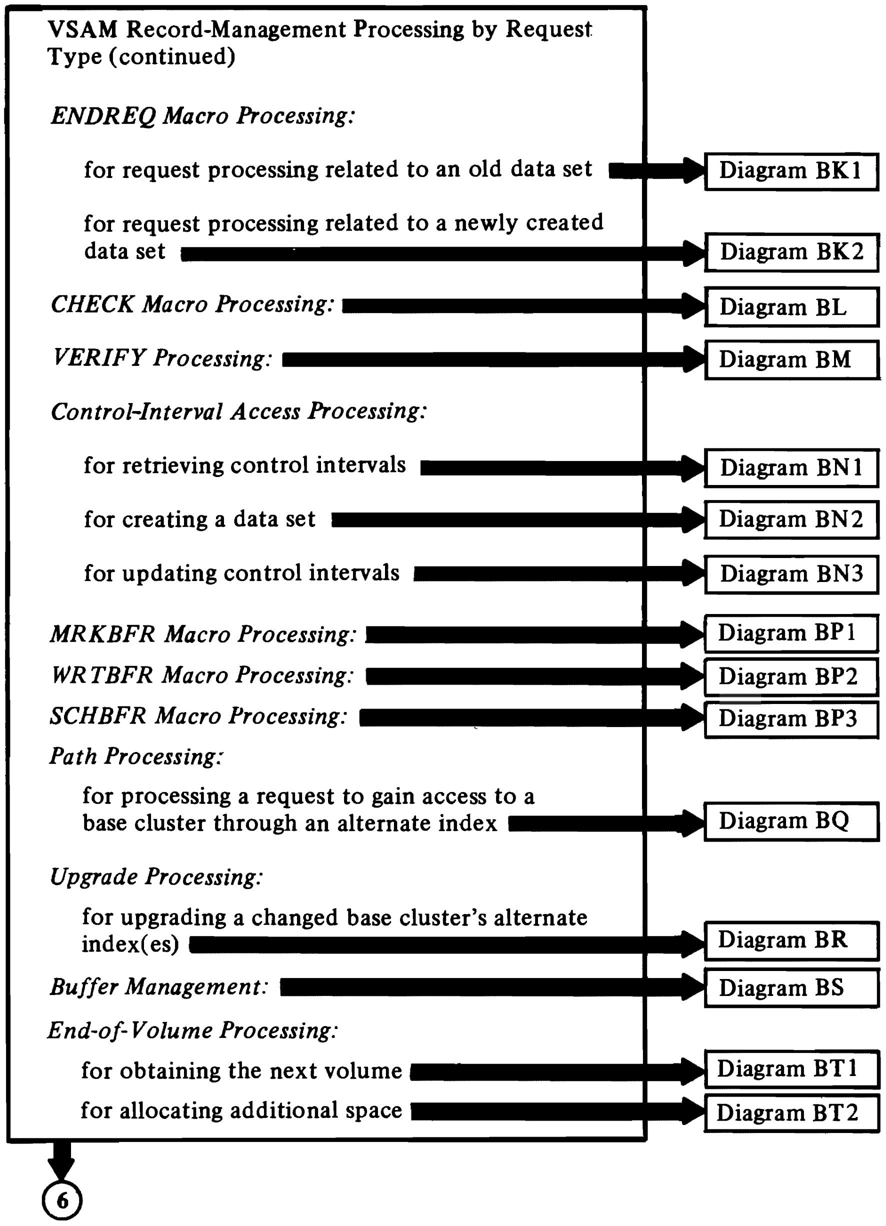
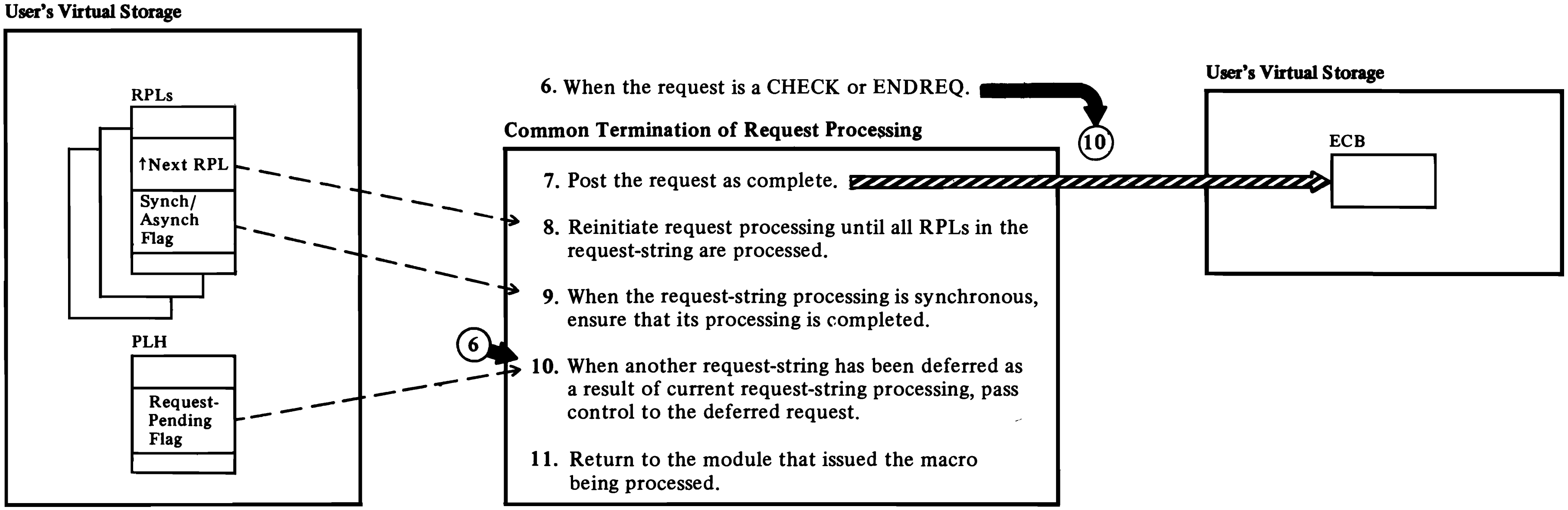

VSAM Record-Management Processing by Request Type (continued)
ENDREQ Macro Processing:
for request processing related to an old data set
Diagram BK1
for request processing related to a newly created data set
Diagram BK2
CHECK Macro Processing:
Diagram BL
VERIFY Processing:
Diagram BM
Control-Interval Access Processing:
for retrieving control intervals
Diagram BN1
for creating a data set
Diagram BN2
for updating control intervals
Diagram BN3
MRKBFR Macro Processing:
Diagram BP1
WRTBFR Macro Processing:
Diagram BP2
SCHBFR Macro Processing:
Diagram BP3
Path Processing:
for processing a request to gain access to a base cluster through an alternate index
Diagram BQ
Upgrade Processing:
for upgrading a changed base cluster's alternate index(es)
Diagram BR
Buffer Management:
Diagram BS
End-of-Volume Processing:
for obtaining the next volume
Diagram BT1
for allocating additional space
Diagram BT2
6

Diagram BB3. VSAM Record Management Overview



6. When the request is a CHECK or ENDREQ.

Common Termination of Request Processing

7. Post the request as complete.
8. Reinitiate request processing until all RPLs in the request-string are processed.
9. When the request-string processing is synchronous, ensure that its processing is completed.
10. When another request-string has been deferred as a result of current request-string processing, pass control to the deferred request.
11. Return to the module that issued the macro being processed.

Notes for Diagram BB3

10

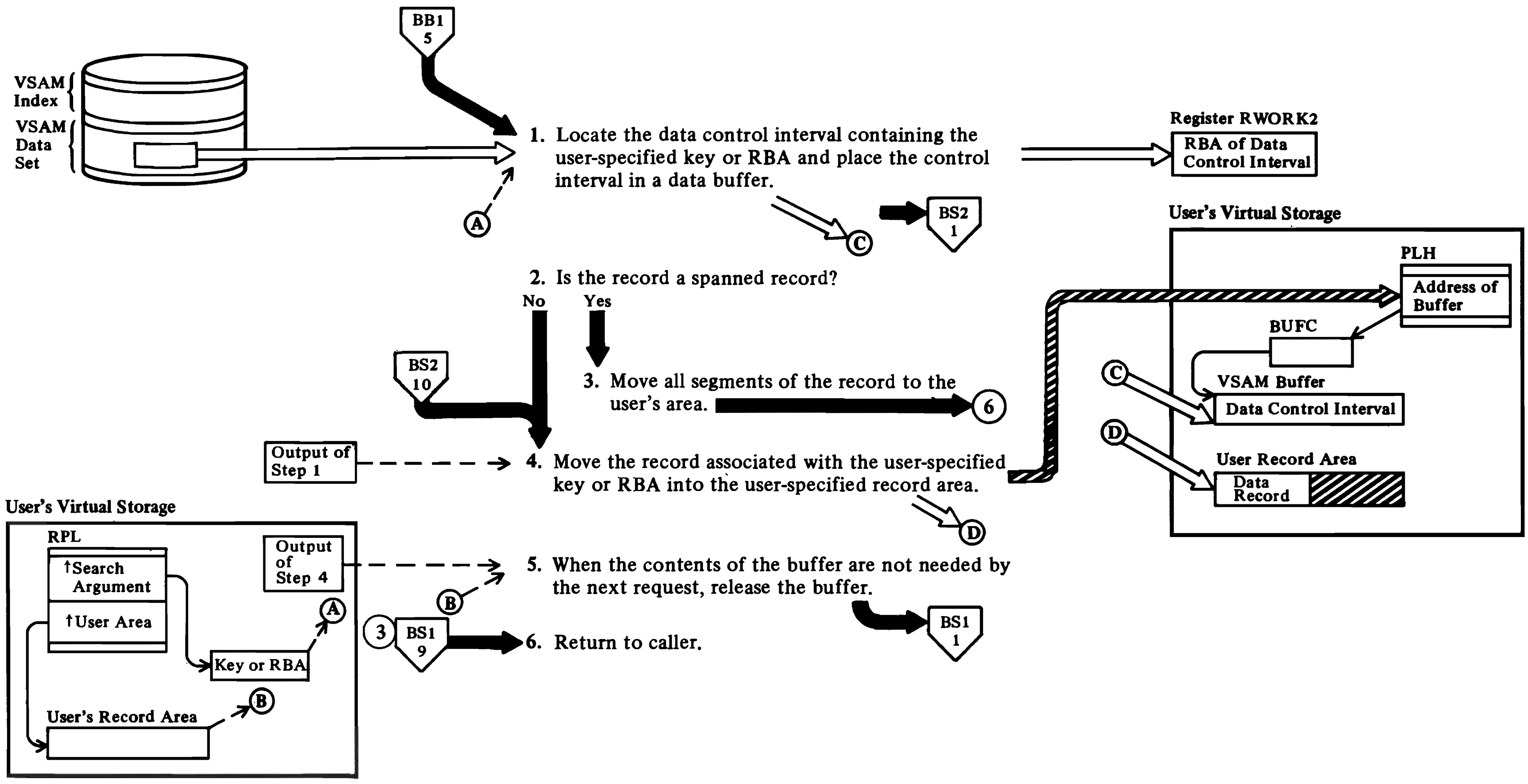
When two request-strings are competing concurrently for a serially reusable resource, the second string is deferred.

When the deferred request is synchronous, a WAIT macro will have been issued against its ECB. When the DIWA is released by another request string, control is returned to a synchronous request at the point at which it issued the WAIT by module IDA019R5.

It posts the request-string's ECB to eliminate the wait condition.

If an asynchronous request is deferred, a return address will have been placed in its placeholder, and when the serially reusable resource becomes available, a branch is made to that address.

Diagram BC. GET-Direct Processing: Direct Retrieval



1. Locate the data control interval containing the user-specified key or RBA and place the control interval in a data buffer.

2. Is the record a spanned record?

3. Move all segments of the record to the user's area.

4. Move the record associated with the user-specified key or RBA into the user-specified record area.

5. When the contents of the buffer are not needed by the next request, release the buffer.

6. Return to caller.

Notes for Diagram BC

1 **Keyed Processing—Key-Sequenced Data Set**

IDA019RA

When the request is keyed, an index search must be performed. The index level where the search begins is based on the following considerations:

- For skip-sequential processing, the index search starts at the sequence set—normally at the index record pointed to by the current PLH. If the PLH is invalid, the search starts at the first record in the sequence set.
- For direct processing, the search starts at the highest level of the index.

IDA019RA calls **IDA019RB,** which calls **IDA019RZ (IDAGRB)**

The index record at which the search is to start is moved into an index buffer.

IDA019RB calls **IDA019RC**

The index record is searched for an entry that is greater than or equal to the search key.

IDA019RB

When the search is unsuccessful, the next record in logical sequence is searched. If the search is successful and a lower index level exists, the search is performed on the index records in the lower level.

Keyed Processing—Relative Record Data Set

IDA019RR

The relative record number that is specified as a search argument is converted into the RBA of the control interval that contains the record, plus the offset of the record in the control interval.

IDA019RR calls **IDA019RZ (IDAGRB)**

The control interval is read in by RBA.

Addressed Processing

IDA019RA

The RBA that is specified as a search argument is converted into the RBA of the boundary of the control interval within which it falls.

2 This step doesn't apply to a relative record data set.

3 **IDA019R4** calls **IDA019RT (IDADARTV)**

A spanned record is retrieved.

IDADARTV calls **IDA019RZ (IDAFREEB)**

A segment is moved to the user's area. The buffer is freed.

IDADARTV calls **IDA019RZ (IDAGNXT)**

The next segment is obtained.

4 **IDA019R4**

If the user is performing locate processing, the address of the record is moved into the user area.

If the request is for update and an upgrade set exists, IDA019RU is called to save the LLOR (least length of the record that contains all key fields). (See Diagram BR.)

Relative Record Processing

IDA019RR

If the user is performing locate processing, the address of the record is moved into the user's area.

5 **IDA019R4: RLSEBUFS** (calls **IDA019RZ)**

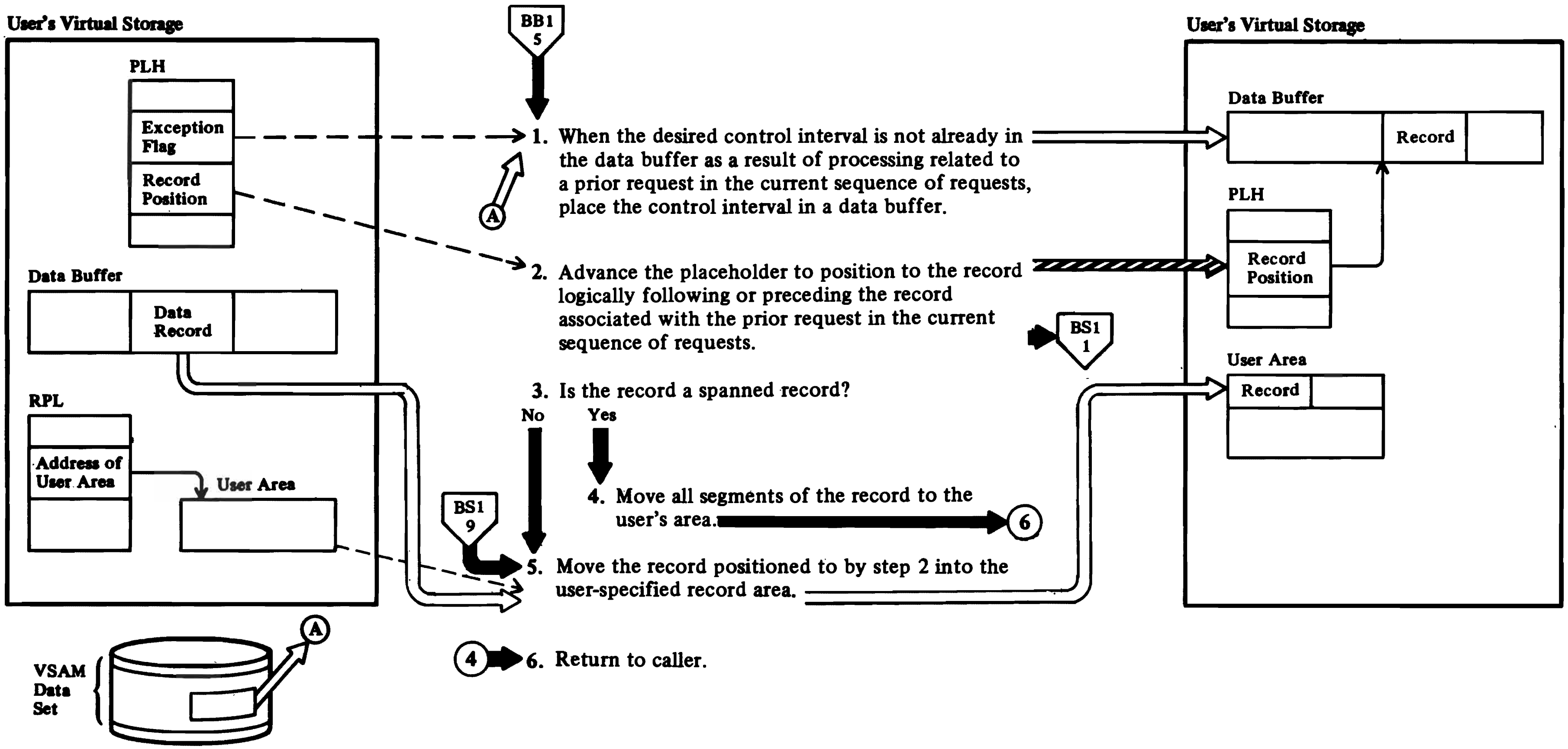
If the request is direct and neither update, note-string-position, nor locate mode is specified, the contents of the buffer are not logically needed by the next request and the buffer is released. If the user is processing with shared resources, any index buffer is freed.

Relative Record Processing

IDA019RR calls **IDA019RZ (IDAFREEB)**

If the request is direct and neither update, note-string-position, nor locate mode is specified, the buffer is released.

Diagram BD. GET-Sequential Processing: Sequential Retrieval



1. When the desired control interval is not already in the data buffer as a result of processing related to a prior request in the current sequence of requests, place the control interval in a data buffer.
2. Advance the placeholder to position to the record logically following or preceding the record associated with the prior request in the current sequence of requests.
3. Is the record a spanned record?
4. Move all segments of the record to the user's area.
5. Move the record positioned to by step 2 into the user-specified record area.
6. Return to caller.

Notes for Diagram BD

Key-Sequenced or Entry-Sequenced Data Set

1 **IDA019R4**

2 **Forward Processing**

IDA019R4: ADVPLH

Normal GET-sequential processing advances the record pointer to the next record in physical sequence in the data buffer.

If the advance positions the record pointer to the end of a control interval, the following processing is performed:

IDA019R4 calls **IDA019RZ (IDAFREEB)**

The current buffer is released.

IDA019R4 calls **IDA019RZ (IDAGNXT)**

The next control interval is retrieved. If the next control interval contains all free space, the retrieval process continues until a control interval containing data is acquired.

Backward Processing

IDA019R4 calls **IDA019RV (IDAADVPH)**

Normal processing advances the record pointer to the preceding record in RBA sequence in the data buffer.

If the record pointer points to the beginning of a control interval, the following processing takes place:

IDAADVPH calls **IDA019RZ (IDAFREEB)**

The current control interval is released.

IDAADVPH calls **IDA019RZ (IDAGNXT)**

The preceding control interval is retrieved.

4 **IDA019R4** calls **IDA019RT (IDADARTV)**

A spanned record is retrieved.

IDADARTV calls **IDA019RZ (IDAFREEB)**

A segment is moved to the user's area. The buffer is freed.

IDADARTV calls **IDA019RZ (IDAGNXT)**

The next segment is obtained.

5 **IDA019R4: DATARTV**

If the request is for update and an upgrade set exists, IDA019RU is called to save the LLOR (least length of the record that contains all key fields). (See Diagram BR.)

Relative Record Data Set

1 **IDA019RR**

The data buffer contains the current control interval.

2 **IDA019RR: ADVPLH**

The record pointer is advanced for normal sequential processing or backed up for backward sequential processing. If the record pointer points to the end of the control interval for normal processing or the beginning of the control interval for backward processing, the following processing takes place:

IDA019RR calls **IDA019RZ (IDAFREEB)**

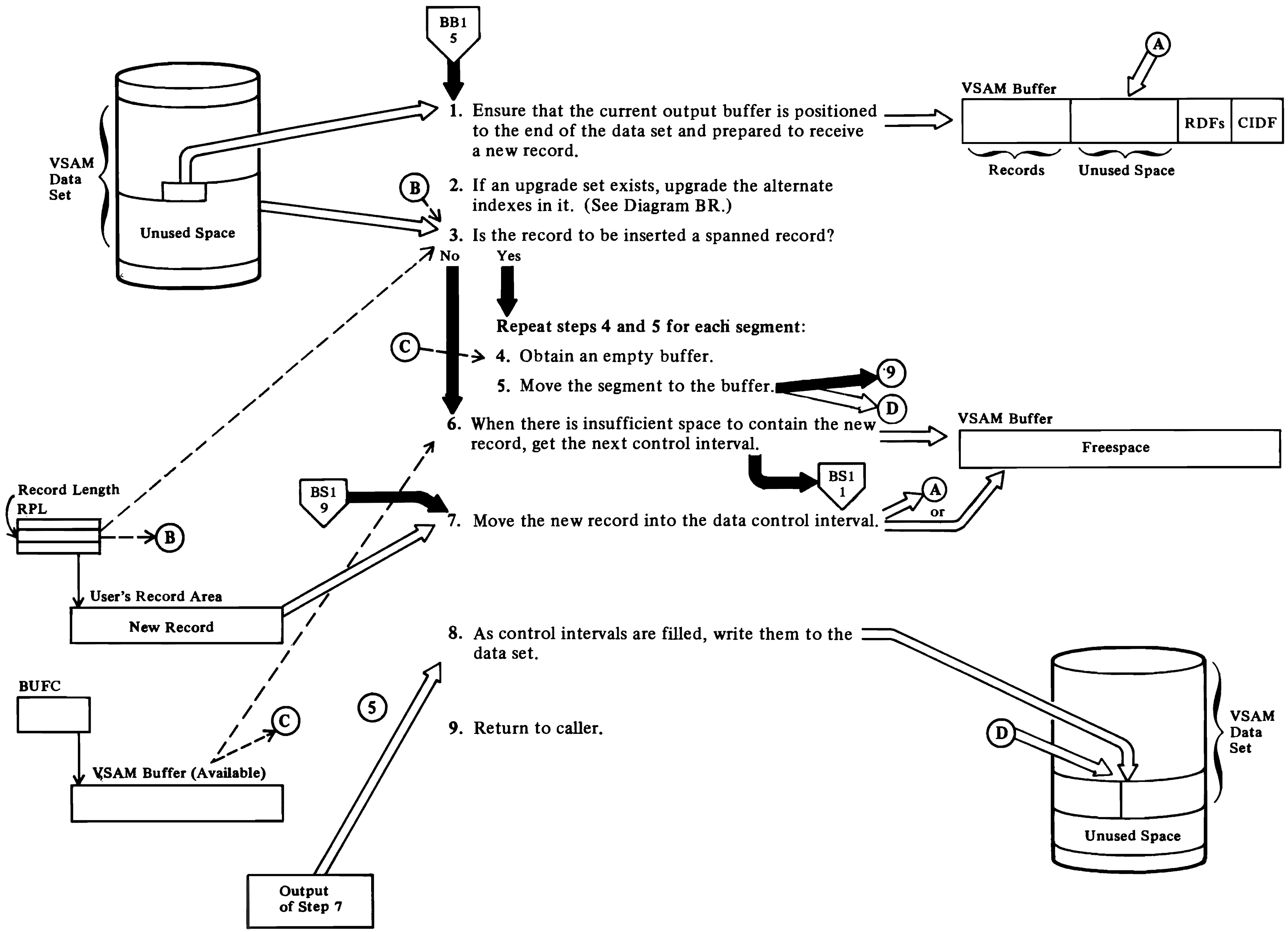
The current buffer is released.

IDA019RR calls **IDA019RZ (IDAGNXT)**

For normal processing, the next sequential control interval is retrieved, and the record pointer is set to the first record. For backward processing, the preceding sequential control interval is retrieved, and the record pointer is set to the last record.

5 **IDA019RR**

Diagram BE. PUT-Entry-Sequenced Processing: Create or Mass Insert



1. Ensure that the current output buffer is positioned to the end of the data set and prepared to receive a new record.
2. If an upgrade set exists, upgrade the alternate indexes in it. (See Diagram BR.)
3. Is the record to be inserted a spanned record?

No Yes

Repeat steps 4 and 5 for each segment:

4. Obtain an empty buffer.
5. Move the segment to the buffer.

6. When there is insufficient space to contain the new record, get the next control interval.
7. Move the new record into the data control interval.
8. As control intervals are filled, write them to the data set.
9. Return to caller.

Notes for Diagram BE

1 **Create Mode Processing**

IDA019R4: SQICHECK (calls **IDA019RZ (IDAGNNFL)**)

When processing is in create mode and the current request is the first request after opening the data set, a buffer is assigned to the request.

IDA019R4: SQICHECK

The buffer is initialized and buffer output is positioned to the first control interval associated with the data set.

Add-to-End or Mass Insert (Noncreate) Processing

IDA019R4: GETINCI (calls **IDA019RA**)

The address of the desired control interval is established by GETINCI, and IDA019RA determines whether the control interval in the current data buffer has that address. When it does not, excess buffers are released (IDA019RA calls IDA019RZ (IDASBF)) and the desired control interval is moved into the buffer (IDA019RA calls IDA019RZ (IDAGRB)).

2 **IDA019R4** calls **IDA019RU**

4 **IDA019RM** calls **IDA019RT**

If the buffer is not empty, IDA019RT calls IDA019SA to obtain an empty buffer.

5 **IDA019RT**

The record segment is moved to the buffer, and the CIDF and RDFs are built.

6 **IDA019R4** calls **IDA019RM**

When there is insufficient space to contain the new record, IDA019RM calls IDA019SA and the following processing is performed:

IDA019SA calls **IDA019RZ (IDAFREEB)**

The current data buffer is released to be written.

IDA019SA: EOCA

When no more control intervals in the current control area can be used, IDA019SA calls IDA019RZ (IDAWRBFR) to ensure that all output to the current control area is completed. Then, after positioning to the next control area boundary, a test is made to determine whether the new control area address exceeds the limits of the data space allocated to the data set. If the data space is exceeded, IDA019SA (EOCA) calls IDA019R5 (IDAEOVIF) to issue an SVC 55 in order to allocate additional extents to the data set.

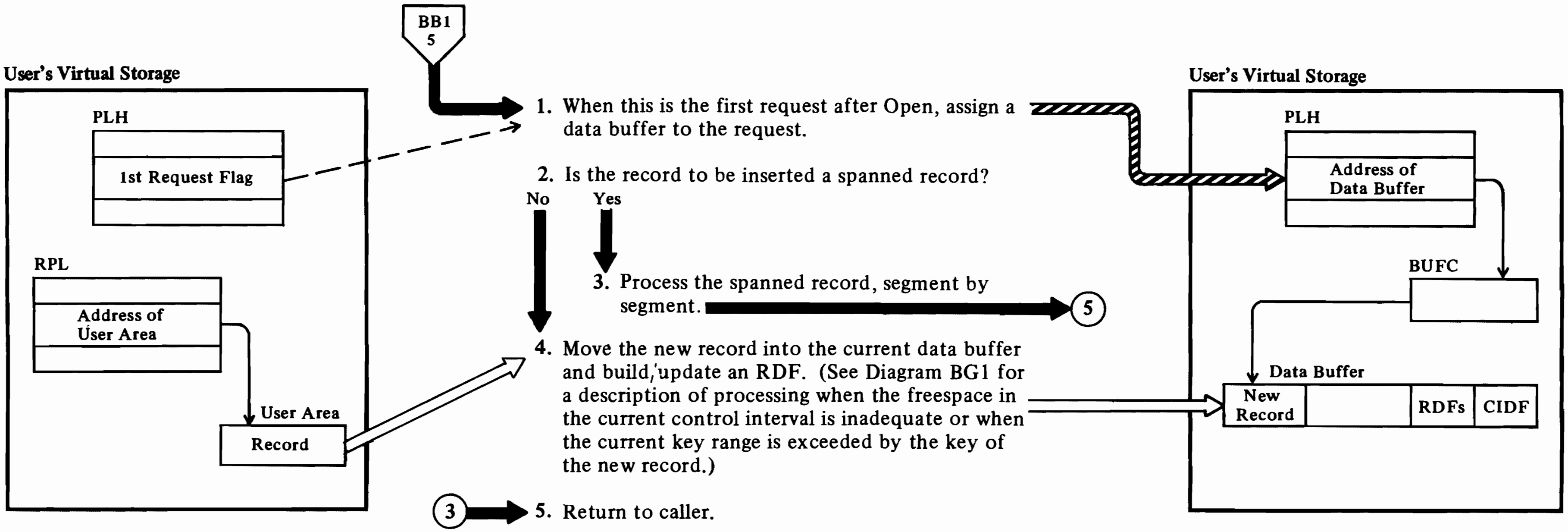
7 **IDA019RM**

Before moving the record into the control interval, an RDF is created for the new record.

8

Actually, this process occurs at step 6. It is not determined that a control interval is full until an attempt is made to insert the next new record.

Diagram BF. PUT-Key-Sequenced Processing: Create



1. When this is the first request after Open, assign a data buffer to the request.

2. Is the record to be inserted a spanned record?

3. Process the spanned record, segment by segment.

4. Move the new record into the current data buffer and build,'update an RDF. (See Diagram BG1 for a description of processing when the freespace in the current control interval is inadequate or when the current key range is exceeded by the key of the new record.)

5. Return to caller.

Notes for Diagram BF

1 **IDA019R4** calls **IDA019RZ (IDAGNNFL)**

The buffer control block entries are searched for an unassigned entry. The first unassigned entry found is assigned to the current request.

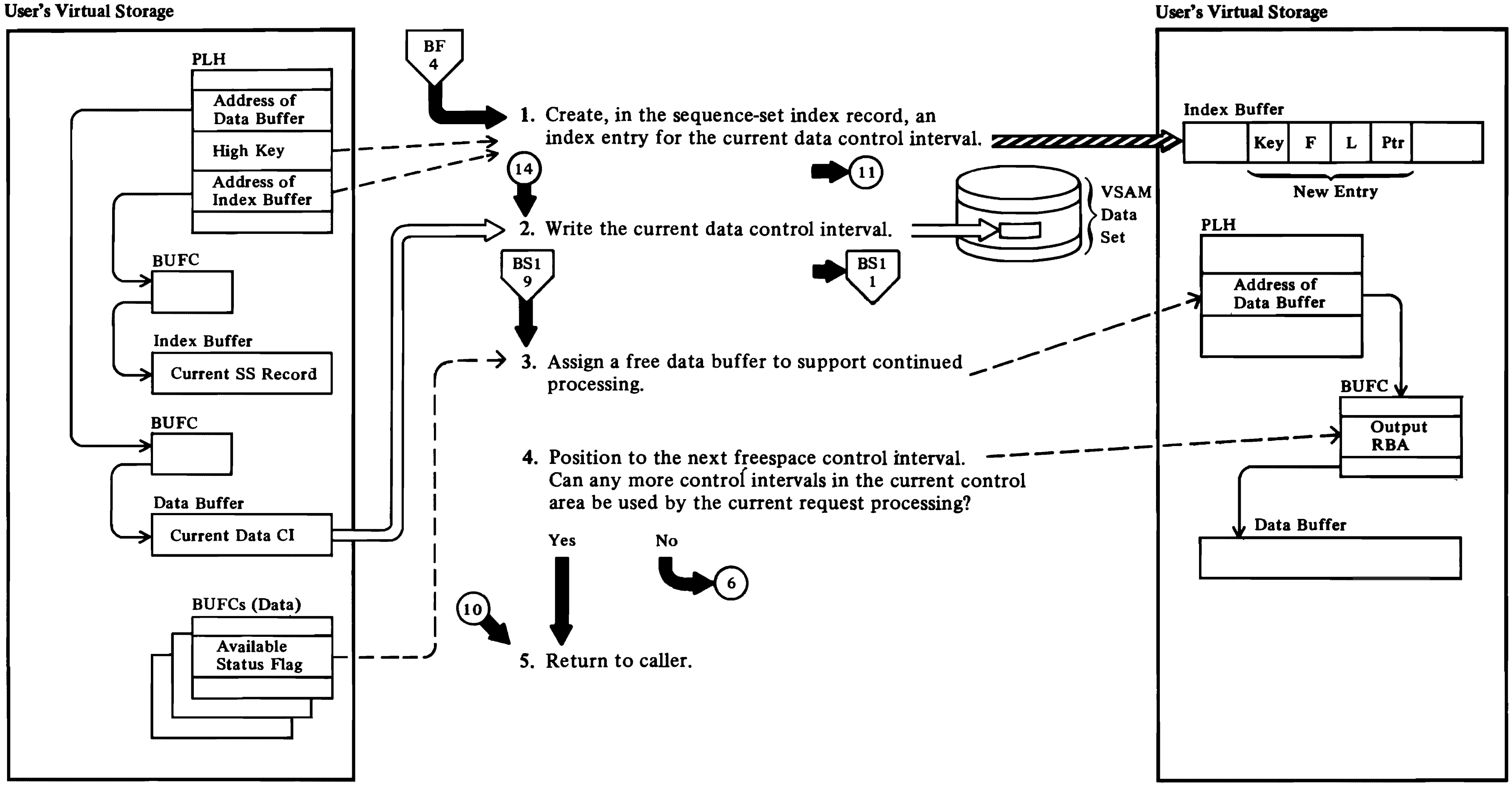
3 **IDA019RM** calls **IDA019RT,** which calls **IDA019SA**

IDA019SA gets an empty buffer. IDA019RT moves a segment to the empty buffer.

4 **IDA019R4** calls **IDA019RM**

Diagram BG1. Creating a Key-Sequenced Data Set

Get a New Freespace Control Interval



1. Create, in the sequence-set index record, an index entry for the current data control interval.
2. Write the current data control interval.
3. Assign a free data buffer to support continued processing.
4. Position to the next freespace control interval. Can any more controſ intervals in the current control area be used by the current request processing?
5. Return to caller.

Notes for Diagram BG1

1 **IDA019SA** calls **IDA019RG**

2 **IDA019SA** calls **IDA019RZ (IDAFREEB)**

The buffer is made available for assignment to another request; however, the next request that attempts to use the buffer must first write the contents to the data set.

3 **IDA019SA** calls **IDA019RZ (IDAGNNFL)**

The BUFC for the next available buffer must be written before it can be used. If the buffer must be written, Buffer Management calls the I/O Manager, IDAM19R3, to perform the write operation, and a wait is performed to ensure that the I/O is completed. (Note: the IDAGNNFL procedure is called when processing in create mode or when adding to the end of an entry-sequenced data set in update mode. Write operations for PUT-sequential processing are initiated only by IDAGNNFL.)

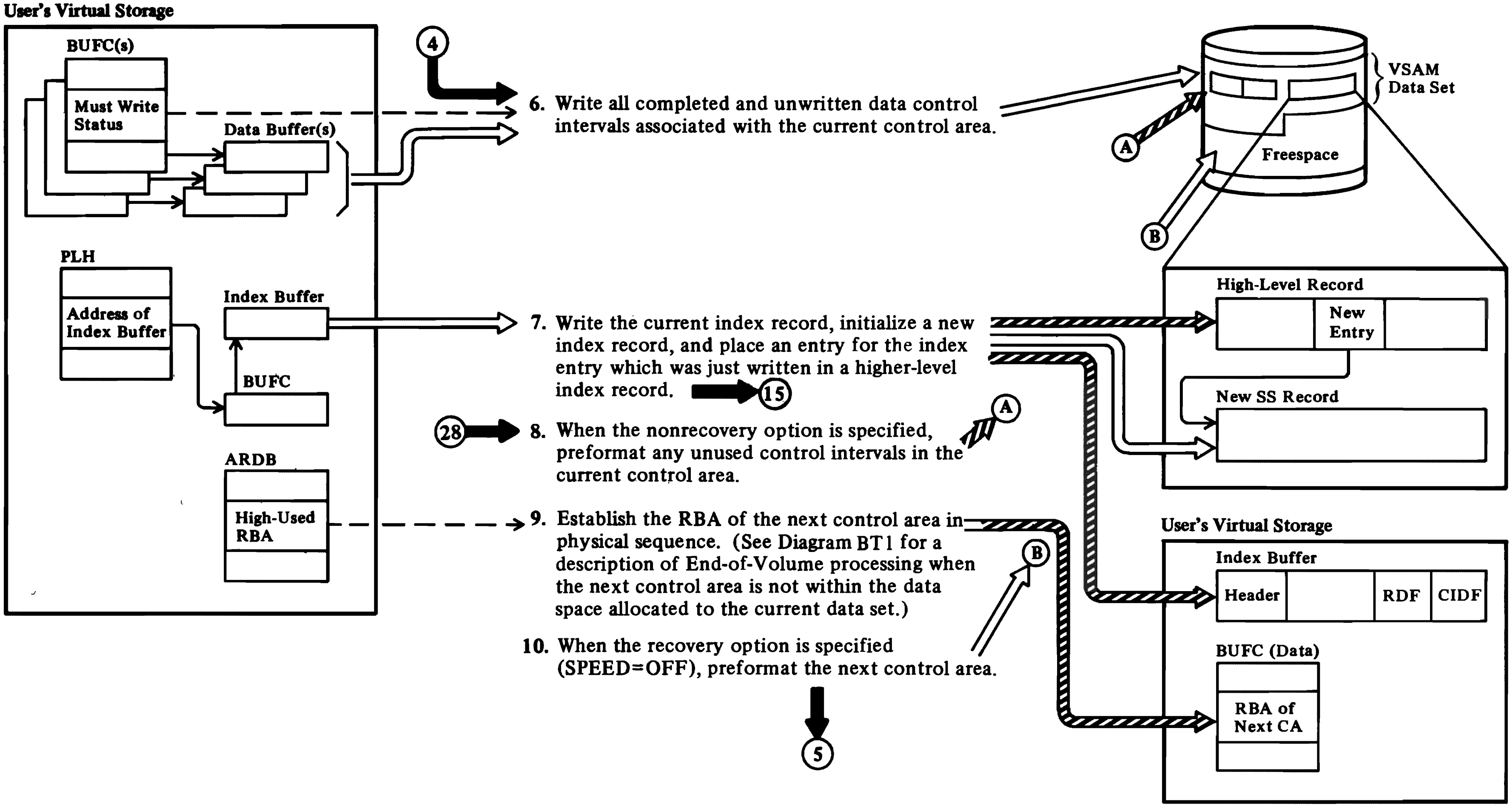
4 **IDA019SA**

IDA019SA: EOCA

More control intervals cannot be added to the current control area if the key of the last record in the last data control interval equals the high key of the current or only key range or if there aren't enough freespace control intervals remaining in the control area to hold the new record and to maintain freespace requirements (that is, to maintain the number of freespace control intervals per control area specified by the user).

Diagram BG2. Creating a Key-Sequenced Data Set

Get a New Freespace Control Area



6. Write all completed and unwritten data control intervals associated with the current control area.

7. Write the current index record, initialize a new index record, and place an entry for the index entry which was just written in a higher-level index record.

8. When the nonrecovery option is specified, preformat any unused control intervals in the current control area.

9. Establish the RBA of the next control area in physical sequence. (See Diagram BT1 for a description of End-of-Volume processing when the next control area is not within the data space allocated to the current data set.)

10. When the recovery option is specified (SPEED=OFF), preformat the next control area.

Notes for Diagram BG2

6 IDA019SA: EOCA (calls **IDA019RZ (IDAWRBFR)**)

Other than the current data buffer, all of the data buffers that have not been previously written are written to the current control area.

7 IDA019SA calls **IDA019RG**

8 IDA019SA calls **IDA019RK**

9 IDA019SA

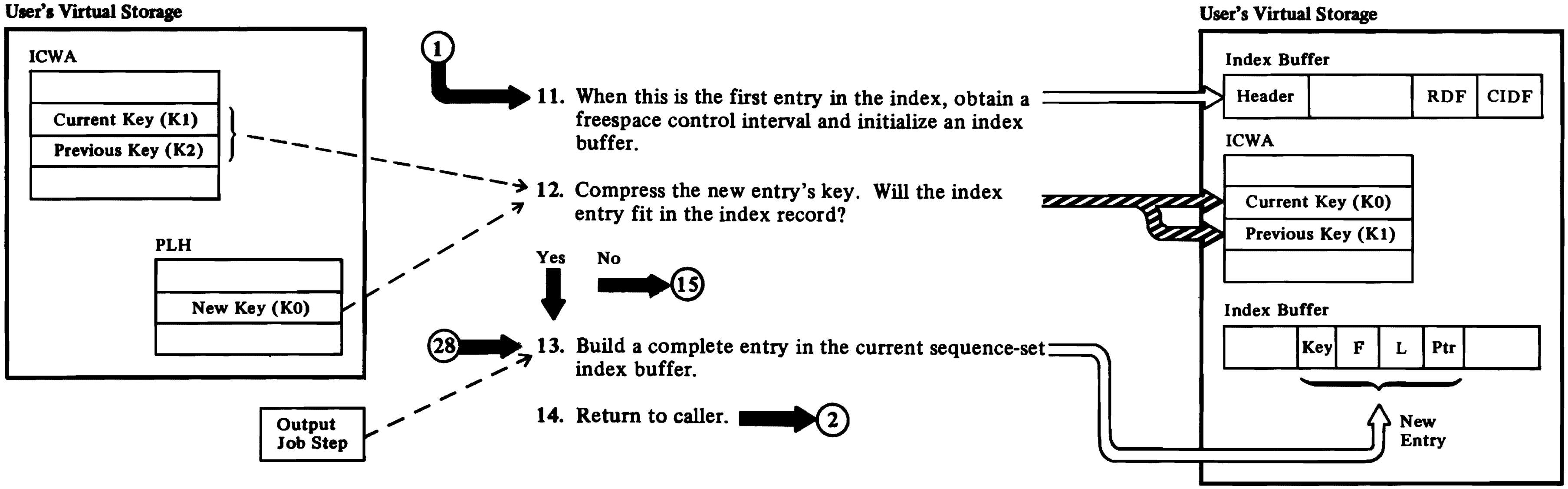
IDA019SA calls **IDA019R5 (IDAEOVIF)**

The end-of-volume processor is called to allocate additional extent(s) to the data set if necessary.

10 IDA019SA calls **IDA019RK**

Diagram BG3. Creating a Key-Sequenced Data Set

Make an Index Entry for a Completed Data Control Interval



11. When this is the first entry in the index, obtain a freespace control interval and initialize an index buffer.

12. Compress the new entry's key. Will the index entry fit in the index record?

13. Build a complete entry in the current sequence-set index buffer.

14. Return to caller.

Notes for Diagram BG3

11 IDA019RG calls **IDA019RN (IDAAQR)**

The index address-range-definition block (ARDB), which governs the range of keys that include the new index entry's key, is located. The field in the ARDB that contains the address of the next available freespace control interval is placed in the index create work area (ICWA).

IDA019RG calls **IDA019RZ (IDAGNFL)**

An index buffer is assigned to the request.

IDA019RG: INTNEWRC

The contents of the index buffer are set to 0 and the following items in the buffer are initialized to form an index record: header, dummy entry, CIDF, RDF, and freespace data control interval pointers (if the request is for a sequence-set record).

12 IDA019RG: IDAIST

Before the new entry's key is compressed, the current, previous, and section key values in the ICWA are updated; the current key becomes the previous key, the new key becomes the current key, and the section key is updated if a new section entry has been built.

The new key is compared with the previous section key, and a count of the common leading characters in the keys is set as a front compression value. (Note: The new key is front-compressed as if it were for a section entry even though it may not be. Because they front-compress less, section entries are slightly larger than normal entries.)

When the current index record is a sequence-set index record, the current key is rear-compressed relative to the next data-record key, that is, the key of the first data record in the next data control interval. The next data-record key is in the record located by the RPLAREA field.

The characters in the keys are compared from left to right until two corresponding characters in the respective keys differ in value. The current key is then truncated at this point.

The length of the new entry is established, based on the compressed key and section pointer, F, L, and normal pointer field lengths. When there is inadequate unused space in the current index record to contain the new entry, a return is made to the caller, IDA019SA, to obtain a new control area.
(Note: IDA019SA recalls IDA019RG to write the current index record and to create an entry for the newly completed index record in a higher-level index record.)

13 Section Entry Processing

IDA019RG: IDAIST

Move the F, L, and key values into the dummy entry, which becomes the new section entry. Then set the offset to the new dummy's F field in the new section entry's LL field. (Note: The offset in the LL field is incremented by the displacement to each succeeding new dummy entry's F field until a new section entry is established. The process then repeats for each succeeding section entry until the record is filled.)

When a previous section entry exists, it is linked to the new section entry by setting the displacement between the F fields of the new and previous section entries in the previous section entry's LL field.

When the insertion is to a sequence-set record or when an index-record split was just performed on the index record to receive the new entry, the next freespace control interval pointer in the index record is moved into the dummy record. (Note: A dummy record is always maintained as the highest possible key in the index during create processing in order to make the index complete and searchable even while it is being created.)

When the new section entry is made in a high-level index record, the RBA of the current index record in the next lower index level is converted to an index entry pointer and placed in the dummy entry.
(Note: There is an ICWA for each level of the index. Each ICWA has a field containing the RBA of the current index record at its particular index level.)
When the current index record in the next lower level is completed, its high key will be placed in the dummy entry and this cycle continues.

14 Normal (or Nonsection) Entry Processing

IDA019RG: IDAIST

The current key is front compressed relative to the previous key. The front compression performed in step 12 is based on the assumption that the new entry is a section entry. Only the rear compression performed for step 12 is valid in this normal, or nonsection, entry case.

The key length is calculated and the F, L, and key values are moved into the new entry.

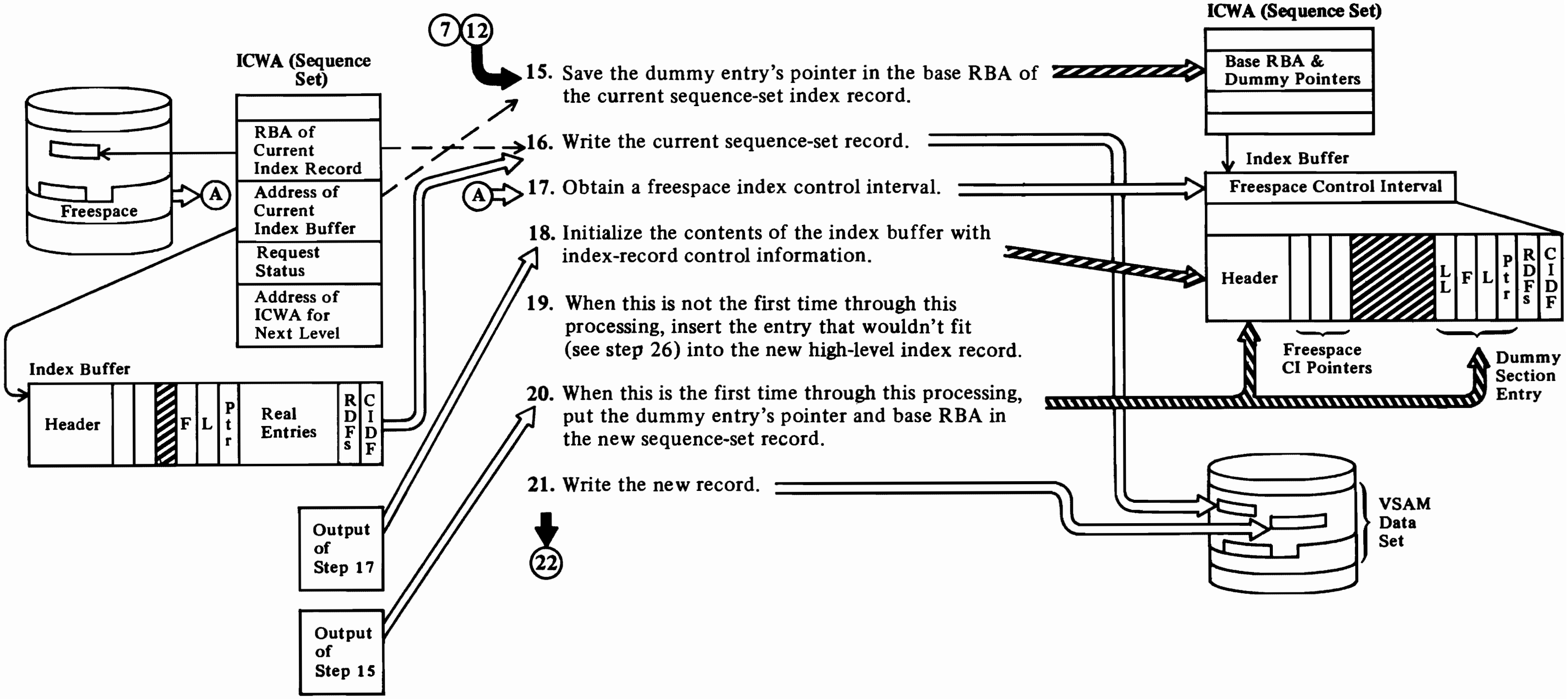
When a section entry has not been built, the section entry pointer in the index record header is advanced to point to the F field in the new dummy entry.

When a section entry has been built, the LL field is incremented by the displacement between the new entry's F field and the new dummy entry's F field (see also note 13, "Section Entry Processing,").

See note 13, "Section Entry Processing," for a description of how the dummy entry's pointer is derived.

Diagram BG4. Creating a Key-Sequenced Data Set

Insert an Index Entry for a New Index Record in an Index Record at the Next Higher Level



15. Save the dummy entry's pointer in the base RBA of the current sequence-set index record.

16. Write the current sequence-set record.

17. Obtain a freespace index control interval.

18. Initialize the contents of the index buffer with index-record control information.

19. When this is not the first time through this processing, insert the entry that wouldn't fit (see step 26) into the new high-level index record.

20. When this is the first time through this processing, put the dummy entry's pointer and base RBA in the new sequence-set record.

21. Write the new record.

Notes for Diagram BG4

15 IDA019RG

The base RBA is the RBA of the data control area controlled by the index record. During index create, the dummy entry points to the freespace control interval following the last control interval in the control area in which data records were inserted. At the end of index-create processing, the dummy points to the control interval containing the high-key record of the data set.

16 IDA019RG calls **IDA019RJ (IDAWR)**

This operation overlays the index record that was generated by step 21 when this procedure was previously entered.

17 IDA019RG calls **IDA019RN (IDAAQR)**

The index address range definition block (ARDB) that governs the range of keys that includes the new index entry's key is located. The contents of the field in the ARDB that contains the address of the next available freespace control interval is placed in the ICWA.

18 IDA019RG: INTNEWRC (calls **IDA019RZ (IDAGNFL)**)

An index buffer is obtained, the buffer is cleared, and then it is initialized as a sequenced-set or a high-level index record.

When the index record is high level (see note 19), a pointer to the lower-level index record just written (see note 21) is moved into the new higher-level index record as the dummy entry representing the highest key of the current level of the index.

19

Steps 17 through 27 represent a repeating sequence of operations that retain control until an index entry is successfully inserted in an index record on the index level above the level on which a new index record is created. The first time through this code, processing is directed at the sequence-set level of the index. Subsequent iterations are directed at successively higher levels of the index.

IDA019RG: IDAIST

The high key of the new lower-level index record is moved into the new higher-level index record built by step 18.

20

Dummy entries are maintained in all levels of the index as the highest possible key in each level in order to ensure that the index is complete, or searchable, even when it is being created. If the index is accessed while it is being created, an index search, no matter how high the key of the search argument, is always satisfied.

For high-level index records (see note 18), the dummy entry points to the incomplete index record at the next lower level, and for sequence-set records, it points to a data control interval.

21 IDA019RJ: IDAWR

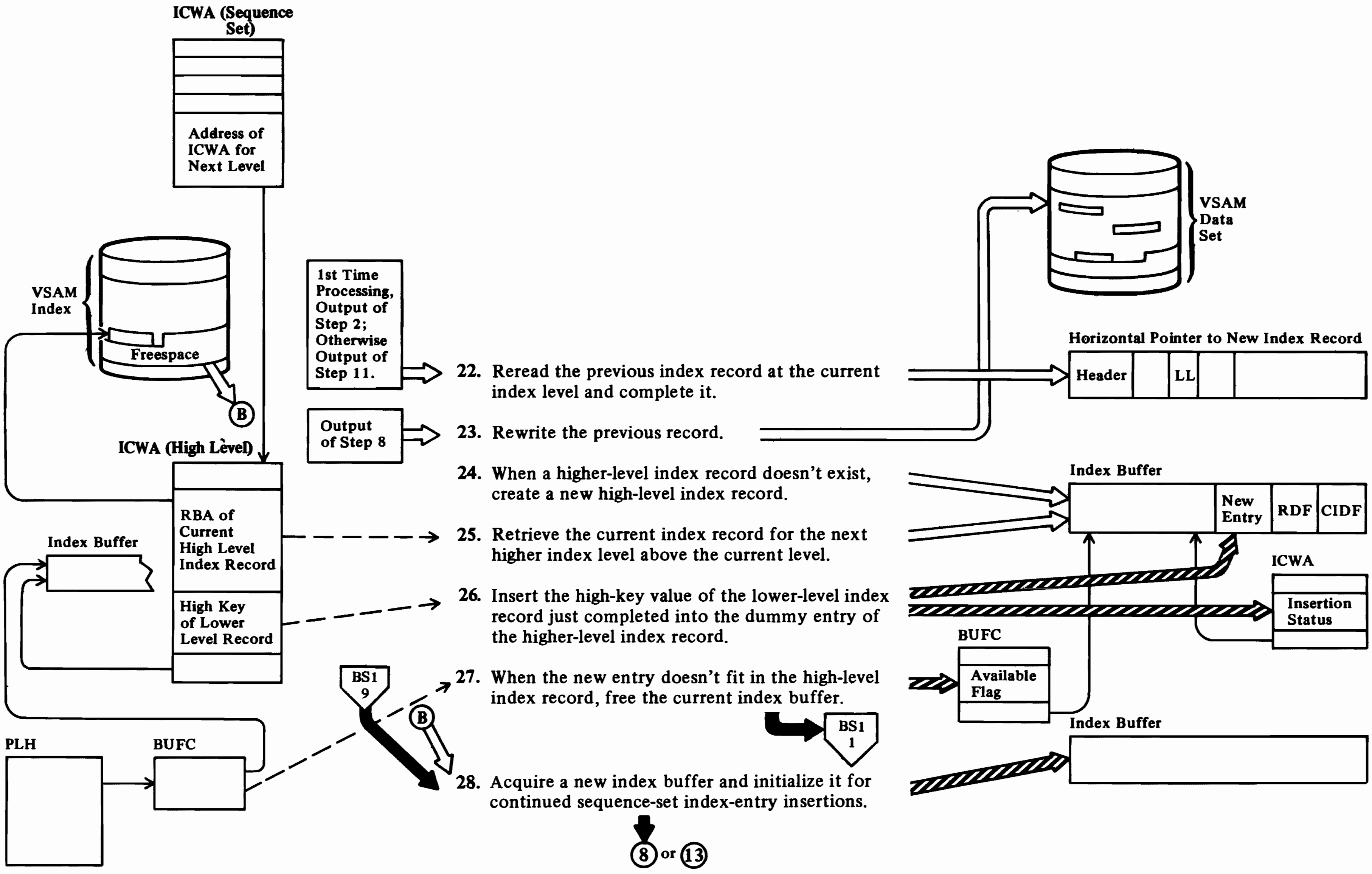
The new sequence-set or high-level index record is written to the data set.

On a sequence-set level, this record points back to the data control interval in the control area belonging to the previous (just completed) sequence-set record and is maintained only to make the index complete. It is destroyed when the next sequence-set index record is completed and written to the data set (see note 16).

On a higher level, this new record has an entry for the index record just completed on the next lower level and a dummy entry for the new incomplete record at that level.

Diagram BG5. Creating a Key-Sequenced Data Set

Insert an Index Entry for a New Index Record in an Index Record at the Next Higher Level (continued)



22. Reread the previous index record at the current index level and complete it.

23. Rewrite the previous record.

24. When a higher-level index record doesn't exist, create a new high-level index record.

25. Retrieve the current index record for the next higher index level above the current level.

26. Insert the high-key value of the lower-level index record just completed into the dummy entry of the higher-level index record.

27. When the new entry doesn't fit in the high-level index record, free the current index buffer.

28. Acquire a new index buffer and initialize it for continued sequence-set index-entry insertions.

Notes for Diagram BG5

22 IDA019RJ: IDAR

The previous index record at the current index level is reread.

IDA019RG

A horizontal pointer to the new record on the current index level is set in the previous index record.

IDA019RG calls **IDA019RN (IDAER)**

The dummy entry in the index record is erased, and the last (high-key) entry, or entry preceding the erased dummy entry, is converted to a section entry. The dummy entry is removed without detracting from the completeness of the index because a new dummy entry has been created by steps 20 and 22 (for high-level and sequence-set records, respectively) and because the horizontal pointer in the previous record makes the dummy entry accessible.

23 IDA019RJ: IDAWR

24 IDA019RG calls **IDA019RN (IDAAQR)**

The index address range definition block (ARDB) that governs the range of keys that includes the new index entry's key is located. An ARDB field contains information about the next available freespace control interval; it is placed in the ICWA.

IDA019RG: INTNEWRC

The buffer is initialized as a high-level index record. A pointer to the lower-level index record just completed (see note 19 or 22) is moved into the new higher-level index record as the dummy entry representing the highest key of the current level of the index.

25 IDA019RJ: IDAR

26 IDA019RG

The current key in the ICWA for the current index level is moved into the current-key field in the next higher level's ICWA.

IDA019RG: IDAIST

The value in the higher-level's ICWA is then inserted in the current higher-level record.

27 IDA019RZ: IDAFREEB

When a new entry will not fit in the higher-level record, a new higher-level record is built to contain the new entry.

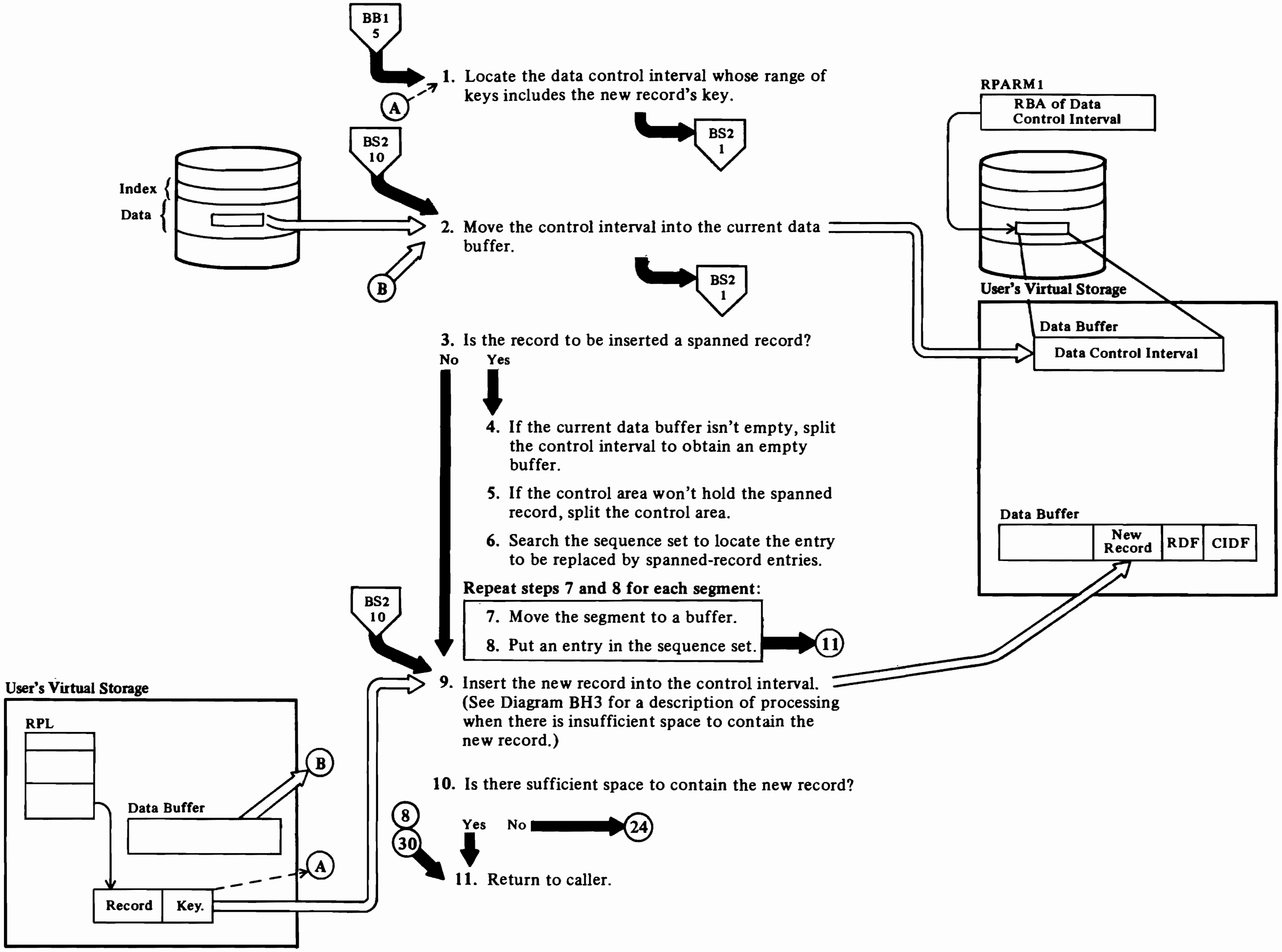
The processing of steps 17 through 27 is repeated until an index entry is successfully inserted in an index record on the index level above the level on which a new index record is created.

28

When this sequence-set record is completed and this routine is reentered, this record will be written at step 16, overlaying the dummy sequence-set record written at step 23.

Diagram BH1. Modifying a Key-Sequenced Data Set

PUT-Insert Processing (Single or Multiple Record Insertion)



1. Locate the data control interval whose range of keys includes the new record's key.

2. Move the control interval into the current data buffer.

3. Is the record to be inserted a spanned record?

No Yes

4. If the current data buffer isn't empty, split the control interval to obtain an empty buffer.

5. If the control area won't hold the spanned record, split the control area.

6. Search the sequence set to locate the entry to be replaced by spanned-record entries.

Repeat steps 7 and 8 for each segment:

7. Move the segment to a buffer.

8. Put an entry in the sequence set.

9. Insert the new record into the control interval. (See Diagram BH3 for a description of processing when there is insufficient space to contain the new record.)

10. Is there sufficient space to contain the new record?

Yes No

11. Return to caller.

Notes for Diagram BH1

1 **IDA019RA**

An index search must be performed. The index level where the search begins is based on the following considerations:

- For skip-sequential processing, the index search starts at the sequence set. The search normally starts at the index record pointed to by the current PLH. If the PLH is invalid, the search starts at the first record in the sequence set.
- For direct processing, the search starts at the highest level of the index.

IDA019RA calls **IDA019RB,** which calls **IDA019RZ (IDAGRB)**

The index record at which the search is to start is moved into an index buffer.

IDA019RB calls **IDA019RC**

The index record is searched for an entry that is greater than or equal to the search key.

IDA019RB

When the search is unsuccessful, the next record in logical sequence is searched. If the search is successful and a lower index level exists, the search is performed on the index records in the lower level.

IDA019RU

If an upgrade set exists, upgrade the alternate indexes in it. (See Diagram BR.)

2 **IDA019RA** calls **IDA019RZ (IDAGRB)**

3 **IDA019RM** calls **IDA019RT**

For spanned-record processing.

4 **IDA019RT** calls **IDA019RE**

IDA019RE is called until the current buffer, whose address is given in PLHDBUFC, is empty.

5 **IDA019RT** calls **IDA019RF**

The control area is split too, when the sequence-set record won't hold enough entries for the spanned-record insertion.

6 **IDA019RT** calls **IDA019RC**

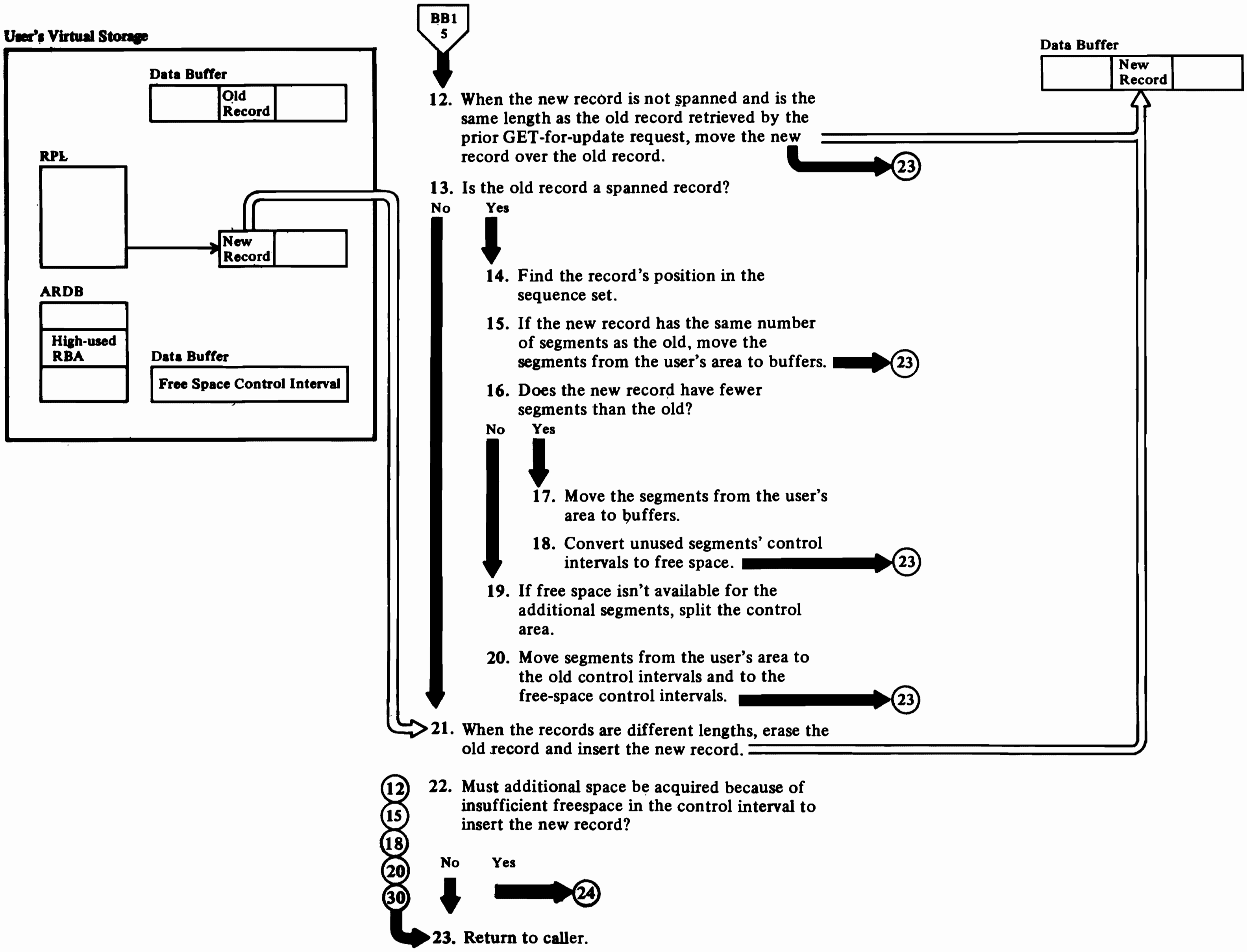
7 **IDA019RT** calls **IDA019RS (IDAMVSEG)**

8 **IDA019RT** calls **IDA019RS (IDAADSEG)**

9 **IDA019R4** calls **IDA019RM**

Diagram BH2. Modifying a Key-Sequenced Data Set

PUT-Update Processing (Modify Existing Record)



12. When the new record is not spanned and is the same length as the old record retrieved by the prior GET-for-update request, move the new record over the old record.

13. Is the old record a spanned record?

14. Find the record's position in the sequence set.

15. If the new record has the same number of segments as the old, move the segments from the user's area to buffers.

16. Does the new record have fewer segments than the old?

17. Move the segments from the user's area to buffers.

18. Convert unused segments' control intervals to free space.

19. If free space isn't available for the additional segments, split the control area.

20. Move segments from the user's area to the old control intervals and to the free-space control intervals.

21. When the records are different lengths, erase the old record and insert the new record.

22. Must additional space be acquired because of insufficient freespace in the control interval to insert the new record?

23. Return to caller.

Notes for Diagram BH2

12 IDA019RL

13 IDA019RL calls **IDA019RS**

IDA019RS gets control only when the old record is a spanned record.

14 IDA019RS calls **IDA019RC**

15 IDA019RS: IDAMVSEG

A CIDF and RDFs are built for each control interval that contains a segment.

17 See note for step 15.

18 IDA019RS: CLEARSEG

An unused buffer is gotten and filled with binary zeros and a free-space CIDF. It is written for each freed segment.

IDA019RS: DELSEG

Entries for unused segments are removed, and free-data-control-interval pointers are set up.

19 IDA019RS calls **IDA019RF**

20 See note for step 15.

IDA019RS: IDAADSEG

Entries for the additional segments are set up in the sequence set.

21 IDA019RL

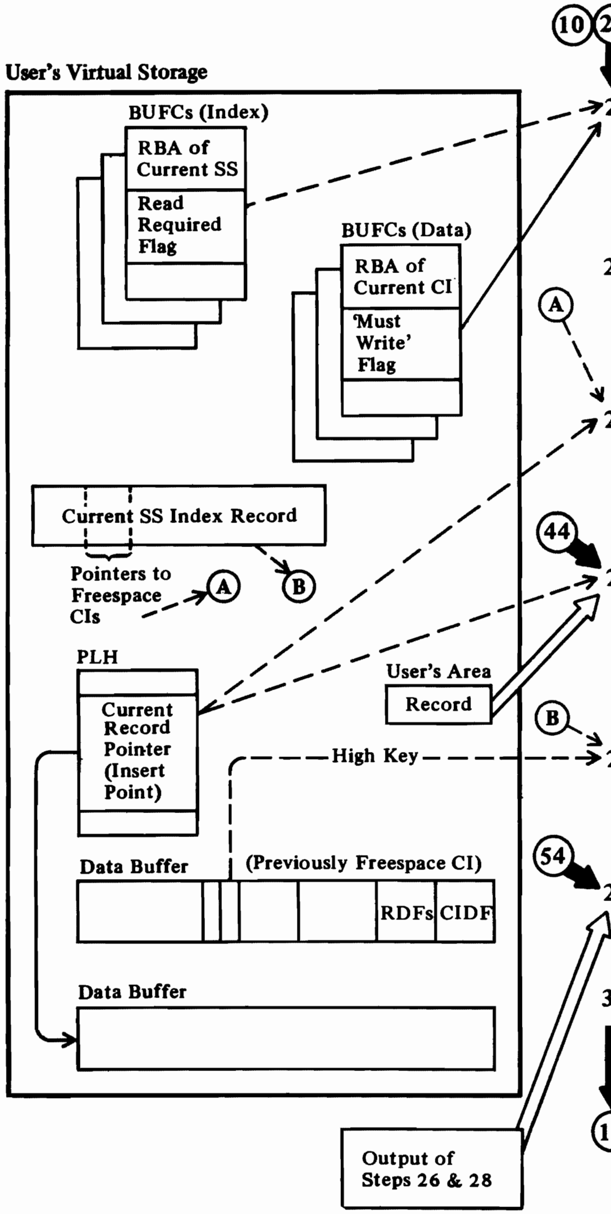
The old (unspanned) record is erased by overlaying it with records to its right. If the record is the last record in the control interval, it is cleared with zeros. IDA019RL then calls IDA019RM to insert the new (unspanned) record.

IDA019RM

If an upgrade set exists, the alternate indexes in it are upgraded. (See Diagram BR.)

Diagram BH3. Modifying a Key-Sequenced Data Set

Create Space to Insert a New or Modified Record in a Data Control Interval



24. Ensure that the current sequence-set and the current data control interval reflect any changes made by other requests strings (if any).

BS2 1 BS1 1

25. When the current control interval must be split, move records from the current control interval into the next freespace control interval in the current control area and adjust RDFs to reflect the new distribution of records.

26. When there isn't a freespace control interval in the current control area, move some control intervals to the next freespace control area to create free-space.

31

27. Insert the new record into the appropriate control interval.

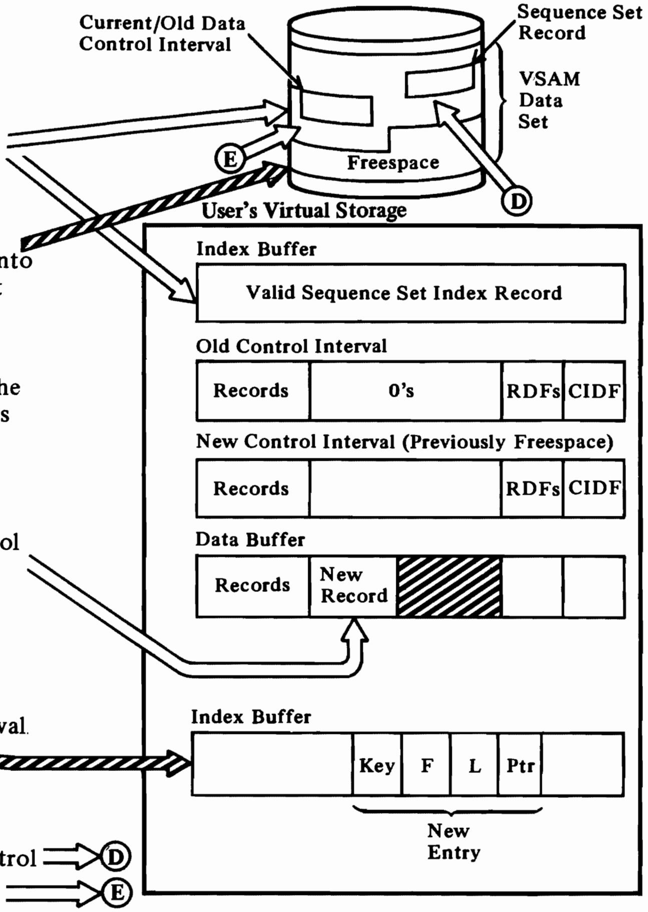
28. Create an index entry for the new control interval in the sequence set index record.

45

29. Write the updated index record, and when a control interval split has occurred, write the old data control interval.

30. When record insertion performed by step 27 is unsuccessful, repeat steps 25 through 29.

11 or 23



Notes for Diagram BH3

24 IDA019RE calls **IDA019RZ (IDAGRB)**

When the current sequence-set index record has been updated by another request since it was last read, it must be reread.

IDA019RE calls **IDA019RZ (IDAFREEB)**

When the current data control interval has been updated by another request since it was last written, it must be rewritten to preserve those updates from possible loss.

25

If the record is to be inserted at the end of a control interval or if it is one of a sequence of records to be inserted at the beginning of a control interval, the control interval is not split and the record is placed in the next control interval currently containing freespace.

If the request is a direct request to insert a record at the beginning of the control interval or if it is either a direct or sequential request to insert a record at some point other than the beginning or end of the control interval, the control interval must be split.

If the request is a sequential request, the control interval is split at the point where the data record is to be inserted.

If the request is a direct request, the record boundary nearest to the midpoint of the control interval is used as the split point.

The RDFs are divided among the control intervals so that they remain associated with their respective records.

IDA019RE calls **IDA019RZ (IDAGNFL)** and **IDA019RE (BUILDFS)**

A work buffer is obtained, converted to freespace, and attached to the data insert work area (DIWA). The work buffer is used to perform the record insertion processing.

IDA019RE

Records to the right of the split point in the old control interval are moved into the new freespace control interval. Then the moved records are zeroed-out in the old control interval and the freespace pointers in each control interval's CIDF are adjusted.

26 IDA019RE calls **IDA019RF**

27 IDA019RE calls **IDA019RM**

28 IDA019RE calls **IDA019RH**

The new index entry reflects the high key of the data records within the new data control interval. If the new index entry fits in the index record, the buffer that contains the record is not written to the index until the new data control interval is written to the data set.

29 IDA019RE calls **IDA019RZ (IDAWRBFR)**

The new data control interval residing in the work buffer associated with the DIWA is written.

IDA019RE calls **IDA019RH (IXIDAWR)**

The updated index record residing in the index buffer associated with the current placeholder is written.

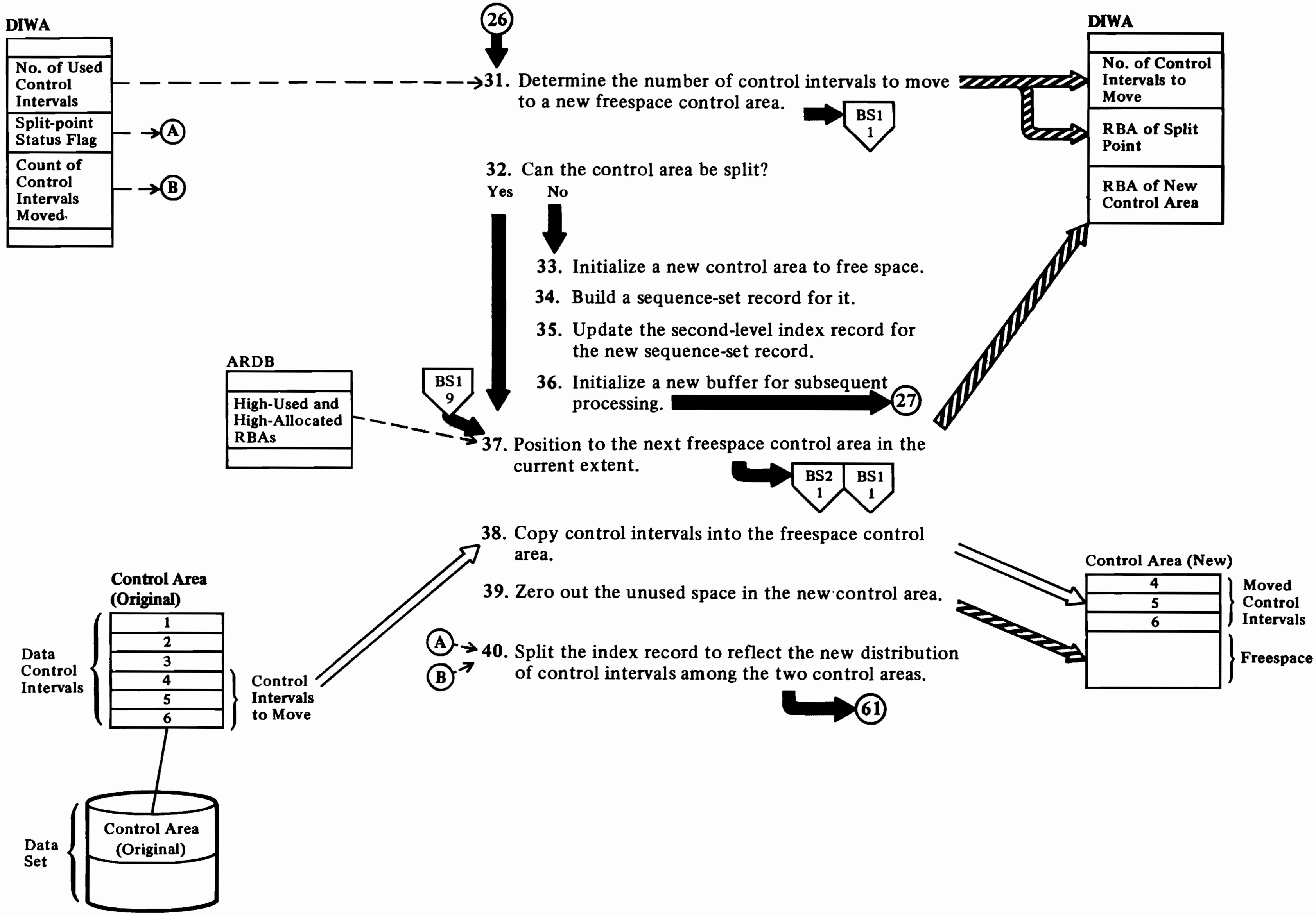
IDA019RE calls **IDA019RZ (IDAWRBFR)**

When a control interval split occurs (see note 25), the old data control interval associated with the current placeholder is written.

30 If the record insertion is unsuccessful after the control interval has been split, a second pass results in a successful insertion—IDA019R4 has verified that the record fits in a control interval.

Diagram BH4. Modifying a Key-Sequenced Data Set

Split a Control Area to Create Freespace and to Generate a New Index Record



31. Determine the number of control intervals to move to a new freespace control area.

32. Can the control area be split?

33. Initialize a new control area to free space.

34. Build a sequence-set record for it.

35. Update the second-level index record for the new sequence-set record.

36. Initialize a new buffer for subsequent processing.

37. Position to the next freespace control area in the current extent.

38. Copy control intervals into the freespace control area.

39. Zero out the unused space in the new control area.

40. Split the index record to reflect the new distribution of control intervals among the two control areas.

Notes for Diagram BH4

When the process involves adding a record to the end of a key range or to the end of the data set, there is no data transfer between control areas. Steps 37 through 41 and 42 are the only steps performed for add-to-end and end-of-key-range processing.

31 **IDA019RF**

The number of control intervals to be moved to the new control area from the control area being split is calculated:

- If the request is a sequential insert request (RPLSEQ=ON), all data control intervals to the right of the insert point are moved to the new control area.
- If the request is a direct request, one half of the data control intervals are moved to the new control area.

IDA019RF calls **IDA019RW (IDAABF)**

Buffers are added to the placeholder's buffer chain until there is a buffer in the chain for each control interval to be moved or until there are no more data buffers in the buffer pool.

32 The control area can't be split if it contains only one (spanned) record (each control interval contains a segment).

33 **IDA019RF** calls **IDA019RK**

34 **IDA019RF** calls **IDA019SF,** which calls **IDA019RI (IDANEWRD)**

The header of the index record is initialized.

User's Key Less Than Old Key

The new sequence-set record is pointed horizontally to the sequence-set record of the old control area. The sequence-set record preceding the old control area's sequence-set record is located.

IDA019SF calls **IDA019RZ (IDAGRB)**

This preceding sequence-set record is read and pointed horizontally to the new sequence-set record.

IDA019SF calls **IDA019RZ (IDAWRBFR)**

The preceding sequence-set record is written.

User's Key Greater Than Old Key

IDA019SF calls **IDA019RZ (IDAWRBFR)**

The new sequence-set record is pointed horizontally to the sequence-set record that the sequence-set record of the old control area pointed to and is written.

IDA019SF calls **IDA019RZ (IDAGRB)**

The sequence-set record of the old control area is read and pointed horizontally to the new sequence-set record.

IDA019SF calls **IDA019RZ (IDAWRBFR)**

The sequence-set record of the old control area is written.

IDA019SF calls **IDA019RI (IDAHLINS)**

IDA019SF calls **IDA019RZ (IDAGNNFL)**

37 **IDA019RF**

Before acquiring a freespace control area, the data buffer control block (BUFC) chain is examined to determine whether any of them have an RBA under exclusive control within the range of RBAs for the control area being split. If there is an exclusive control conflict, an error code is set and a return is made to the caller.

If the boundary of the next freespace control area exceeds the boundary of the current extent, that is, the high-allocated RBA, VSAM End-of-Volume is called via an SVC 55 to attempt to acquire more space (see Diagram BT1, VSAM End of Volume: Obtain the VSAM Object's Next Volume). If space is unavailable, an error code is set in the RPL and a return is made to the caller.

38 **IDA019RF** calls **IDA019RZ (IDAGRB)**

The first control interval is retrieved as a direct request.

IDA019RF calls **IDA019RZ (IDAGNXT)**

Subsequent control intervals are retrieved on a sequential basis.

IDA019RF calls **IDA019RZ (IDAFREEB)**

As each buffer is filled, its must-write flag is set (BUFCMW=ON), and then it is released (BUFCAVL=ON).

IDA019RF calls **IDA019RZ (IDAWRBFR)**

When all of the control intervals eligible for the move have been read into buffers, the buffers are written to the data set.

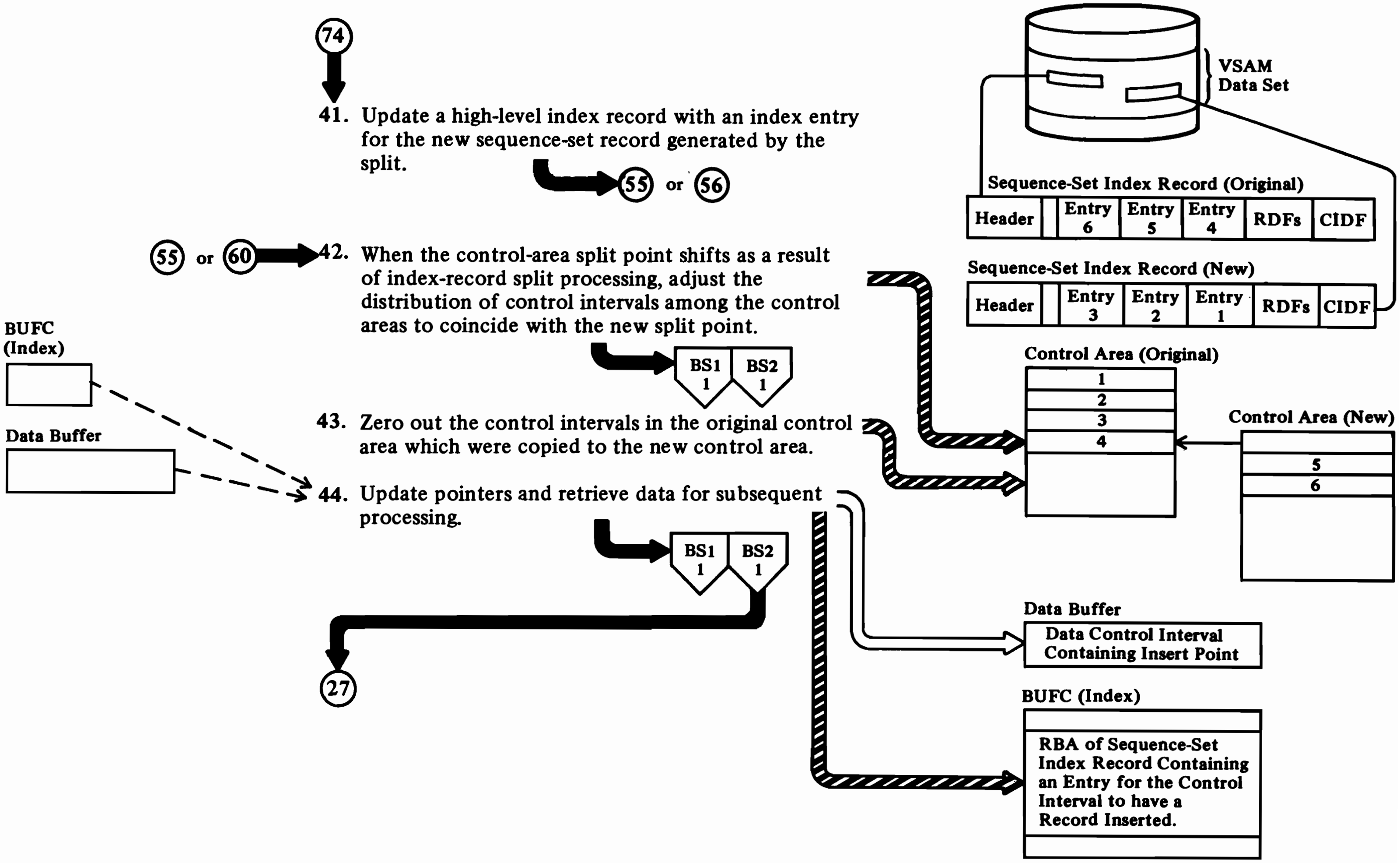
39 **IDA019RF** calls **IDA019RK**

40 **IDA019RF** calls **IDA019RI,** which calls **IDA019RJ**

For add-to-end processing, only a new sequence-set record is created. For other processing, the original control area's sequence-set record is split, thereby creating a new sequence-set record with index entries for the control intervals that were moved to the new control area.

Diagram BH5. Modifying a Key-Sequenced Data Set

Split a Control Area to Create Freespace and to Generate a New Index Record (continued)



41. Update a high-level index record with an index entry for the new sequence-set record generated by the split.

42. When the control-area split point shifts as a result of index-record split processing, adjust the distribution of control intervals among the control areas to coincide with the new split point.

43. Zero out the control intervals in the original control area which were copied to the new control area.

44. Update pointers and retrieve data for subsequent processing.

74
55
or
56
55
or
60
BS1 1
BS2 1
BS1 1
BS2 1
27
BUFC (Index)
Data Buffer
VSAM Data Set
Sequence-Set Index Record (Original)
Header
Entry 6
Entry 5
Entry 4
RDFs
CIDF
Sequence-Set Index Record (New)
Header
Entry 3
Entry 2
Entry 1
RDFs
CIDF
Control Area (Original)
1
2
3
4
Control Area (New)
5
6
Data Buffer
Data Control Interval Containing Insert Point
BUFC (Index)
RBA of Sequence-Set Index Record Containing an Entry for the Control Interval to have a Record Inserted.

Notes for Diagram BH5

41 IDA019RI

42

Any control intervals that were copied into the new control area and that are no longer validly associated with that control area as a result of distribution changes effected by the sequence-set split process are zeroed out in the new control area. The following procedures effect this change:

All of the buffers in the placeholder's buffer chain are zeroed out.

IDA019RF calls **IDA019RZ (IDAGNNFL)**

A buffer in the placeholder's buffer chain is assigned as a work buffer.

IDA019RF calls **IDA019RZ (IDAFREEB)**

The work buffer's must-write flag is set on, and it is freed. (Note: It is written when the next request for a free buffer examines its must-write status and causes it to be written before reassigning it.)

IDA019RF calls **IDA019RZ (IDAWRBFR)**

The previous two steps are repeated until all invalid control intervals in the new control area have been erased. All of the work buffers are then written to the data set.

IDA019RF calls **IDA019RZ (IDAGRB)**

The sequence set of the original control area is then read into an index buffer.

If the control interval containing the insert-point address is returned to the old control area by the process outlined by the previous four steps, the insert point must be recalculated.

IDA019RF calls **IDA019RZ (IDAWRBFR)**

The buffers are written to the data set.

43 IDA019RF

44 IDA019RF

Ensure that the PLH points to the sequence-set record containing an index entry for the data control interval that contains the new record's insert point.

IDA019RZ (IDAFREEB)

If it does not, the index buffer containing the sequence-set record for the old control area is released.

IDA019RZ: IDAGRB

The sequence-set record for the new control area is brought into the index buffer.

IDA019RZ: IDASBF

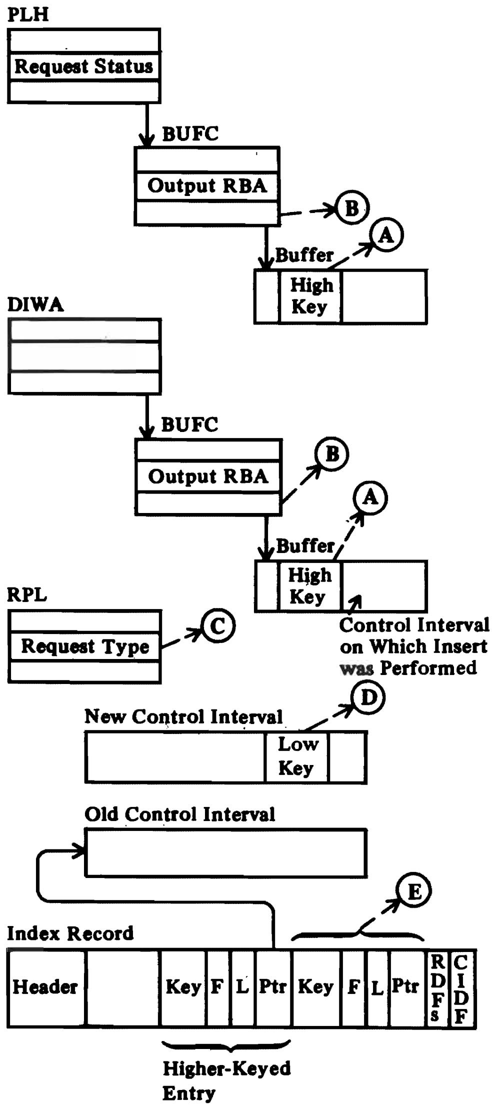
The buffers that were added to the placeholder's buffer chain to support the control-area-split process (see note 31) are released from the chain.

IDA019RZ: IDAGRB

The control interval that contains the insert point is retrieved from the data set and placed in a data buffer.

Diagram BH6. Modifying a Key-Sequenced Data Set

Build an Index Entry and Insert It in an Index Record



(28)

Processing for Insertion of an Entry in the Sequence Set.

(A) 45. Determine which of the data control intervals passed has the highest key.

(B) 46. Establish the high key of the old data control interval for insertion into the sequence-set index record and compute the index-entry pointer values for both the new and the old data control intervals.

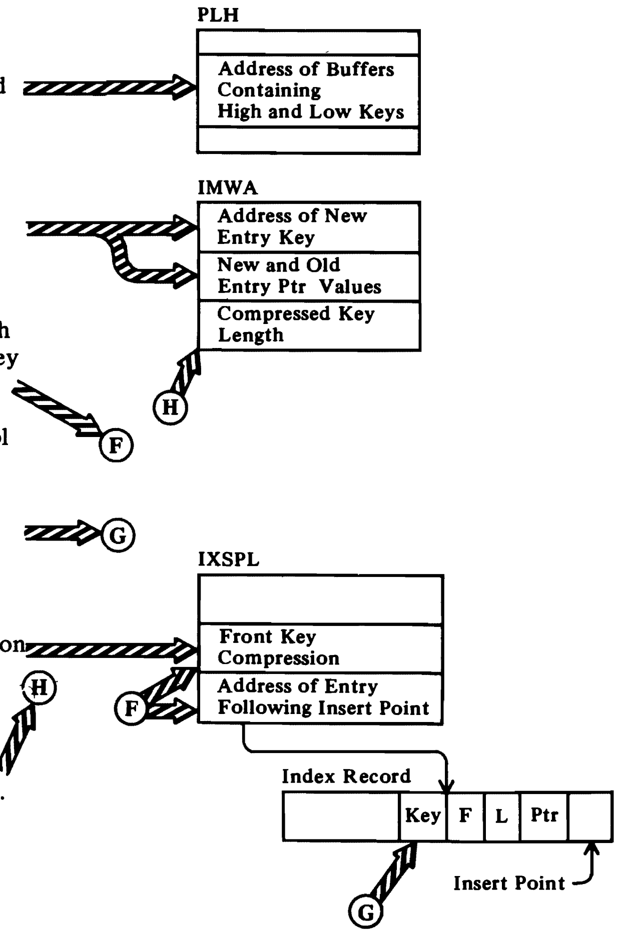
47. Locate the insertion point for the new entry for the old control interval in the index record, and establish a front key-compression value for the new entry's key relative to the lower-keyed entry which it follows.

(C) 48. When the current processing is the result of a control interval split due to a sequential, or mass, insert to the old control interval, do the following:

- Decrement the low key of the new control interval by 1 and use this as the high key of the old control interval to be inserted into the index record. (D)
- Reestablish (see step 47) the front key-compression value of the new entry's key relative to the lower key of the entry which it will follow.

(D) (E) 49. Rear compress the new entry's key relative to the low key in the new data control interval and establish the key length of the fully compressed key.

(50)



Notes for Diagram BH6

45 IDA019RH

46 IDA019RH

47 IDA019RH calls **IDA019RC**

The index-record search begins with a search of the section entries. After a section entry whose key is equal to or greater than the key being sought is located, the individual entries governed by the section entry are examined until a key that is greater than the search key is found.

During the nonsection entry search process, a count of the common leading characters of each entry relative to the search key is maintained. When control is returned to IDA019RH (index insert), this value is sometimes used as the front key-compression value of the new entry's key, or the search key, relative to the previous, or lower-keyed, entry in the index record.

48

Basing the high key of the new control interval on the low key (minus 1) of the next control interval enables the sequential insertion process to continue without having to update the index record for each record in the group of records that are added to the data control interval as a mass insert; otherwise, a relatively small group of records could establish multiple new high keys for the control interval receiving the records.

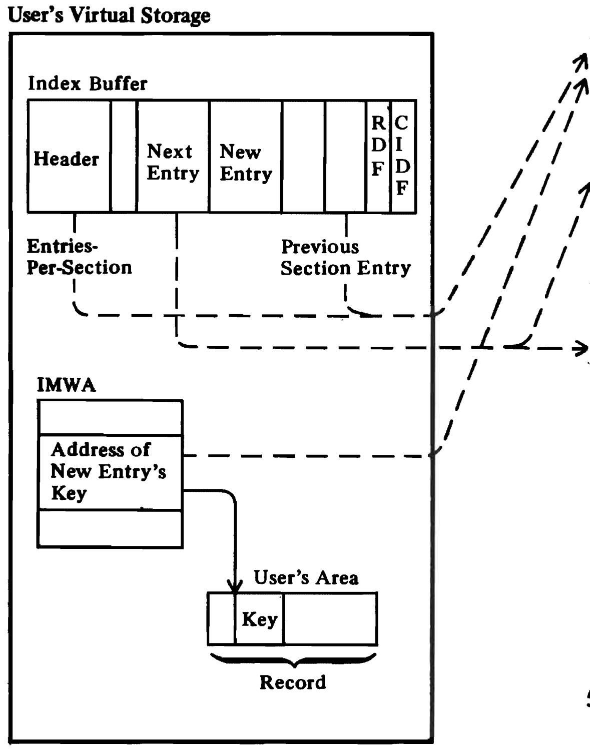
49 IDA019RH: COMPRS

The leading characters of the two keys are compared until the first unlike character is found. The like characters are dropped from the new key when it is compressed.

The front and rear compression values are then used to determine the length of the compressed key.

Diagram BH7. Modifying a Key-Sequenced Data Set

Build an Index Entry and Insert It in an Index Record (continued)



Common Processing for High-Level and Sequence-Set Insertions

50. When the new entry should be a section entry, establish a front key-compression value for the new section entry relative to previous section entries and repeat step 49 for sequence-set records.

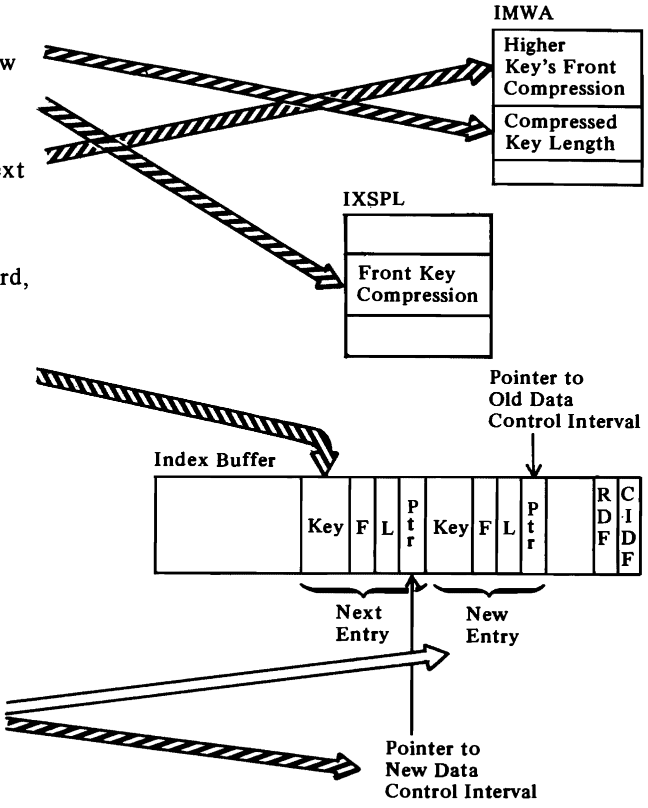
51. Establish a front key-compression value for the next higher key following the new entry relative to the new entry's key.

52. When the new entry does not fit in the index record, return to the caller.

53. Front compress the key field of the next, higher-keyed, index entry.

54. Build the new entry and put a new pointer in the next entry.

29



Notes for Diagram BH7

50 IDA019RH

For section-entry key compression, the new section key is compared against each succeeding section entry, starting with the first, in establishing the front compression value.

51 IDA019RH: HLINSERT

Before establishing a front-compression value, the front key compression, or F value, in the high-keyed index entry is compared against the front-key compression value combined with the key length of the new index entry. If the F value in the high-keyed index entry is not greater than the other combined values, or if the key length, or L value, of the new index entry is 0, compression is not performed.

52 IDA019RH

The length of the new entry's key (L value) plus the standard F, L, and pointer field lengths are compared to the amount of freespace in the current index record combined with the front-compression value established by step 51. (If the entry is a section entry, the length of the section entry pointer (LL field) is also included in these calculations.) If there is insufficient space for the new index entry, control is returned to IDA019RJ (index split), by way of IDA019RI (index update), to split the index record.

53 IDA019RH

The higher-keyed entry is moved to the left, overlaying the front characters in its key which are to be compressed.

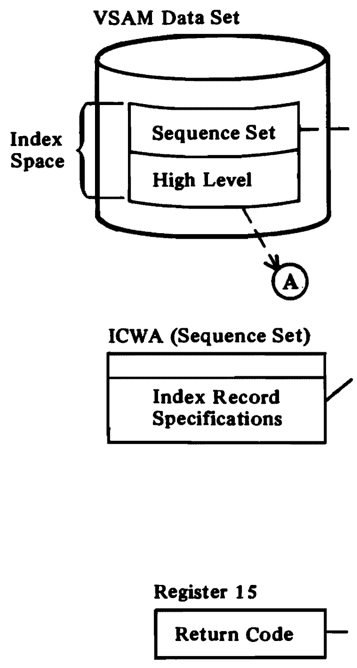
54 IDA019RH

The entries following (to the left of) the insert point are moved to the left, overlaying the freespace to the left of the high-keyed entry in the record, until sufficient space exists at the insert point to contain the new index entry.

The following higher-keyed index entry contains the key of the new data or index control interval generated by IDA019RE (control interval split) or IDA019RJ (index split). Accordingly, its pointer must be replaced with a pointer to the new control interval.

Diagram BH8. Modifying a Key-Sequenced Data Set

Update a High Level of the Index with an Entry for the New Sequence-Set Record.



55. When there is only one index level in the data set, build a high-level index record as follows:
- Obtain the RBA of a freespace index control interval for the high-level record.
- Assign an index buffer to the request and build the high-level index record.
- Write the high-level record.
- Return to the caller.

Index-Entry Processing When a High-Level Index Record Exists

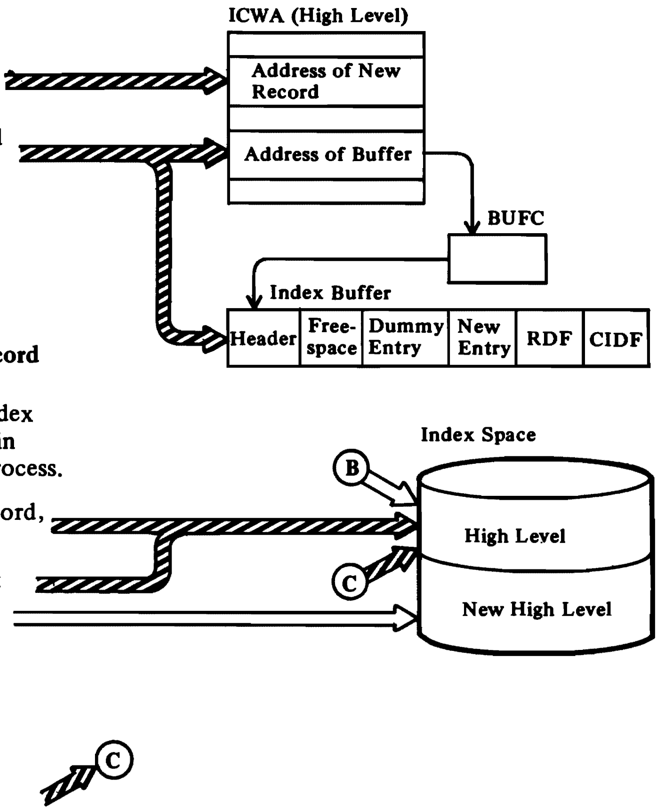
56. Search the next higher level of the index for an index record whose range of keys includes the high key in the new index record created by the index split process.

57. Insert an entry, which points to the new index record, in the higher-level index record.

58. When the insertion is unsuccessful, locate the split point and split the higher-level index record to create space for the new entry.

59. After the record is split following an unsuccessful attempt to insert an entry, insert the new entry in the unused space generated by the split process (see step 58).

60. Repeat steps 56 through 58 until the index level above the level on which the last split was performed contains a new entry for the new record created by the split process.



Notes for Diagram BH8

55 IDA019RI calls **IDA019RN (IDAAQR)**

The index address range definition block (ARDB) that governs the range of keys that includes the new index entry's key is located. The contents of the field in the ARDB that contains the address of the next available freespace control interval is placed in the ICWA.

IDA019RJ calls **IDA019RK**

If this is the first time that space governed by the ARDB located above has been used and if sequence-set-with-data is specified, the new index record requires preformatting. Starting at the address established above, software end-of-file control intervals (zeros) are built until the end of the track on which replication is to occur is reached.

IDA019RI calls **IDA019RZ (IDAGNFL)**

A buffer is assigned to the request.

IDA019RI calls **IDA019RH,** which calls **IDA019RZ (IDAWRBFR)**

The high level index record is written.

56 IDA019RI calls **IDA019RB**

57 IDA019RI calls **IDA019RH**

58

If there was insufficient space in the index buffer to support the index-split process, an attempt is made to provide more space.

IDA019RI: FINDSP

The offset to the section entry containing the split point is established by tracing along the chain of section entries until a section entry is reached whose displacement from the start of the index record is less than the displacement of the split point used in the prior unsuccessful split operation.

IDA019RI: LNEXTE

Using this information, a new split point is established for the next attempt to split the index record.

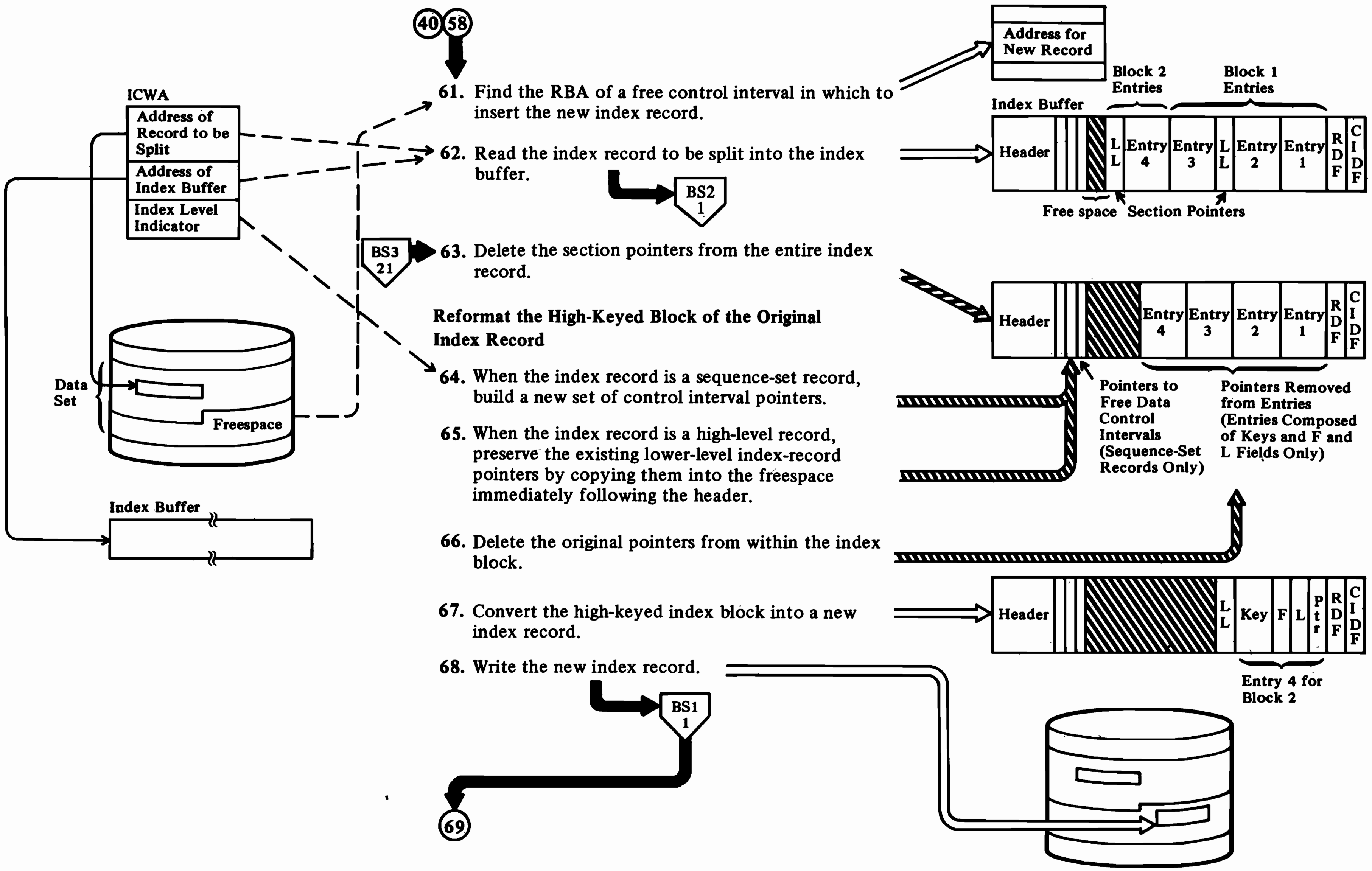
IDA019RI calls **IDA019RJ**

The index record is split to create space for the index entry associated with the new index record created by the split process.

59 IDA019RI calls **IDA019RH**

Diagram BH9. Modifying a Key-Sequenced Data Set

Split an Index Record to Create Space for a New Index Entry



61. Find the RBA of a free control interval in which to insert the new index record.

62. Read the index record to be split into the index buffer.

63. Delete the section pointers from the entire index record.

Reformat the High-Keyed Block of the Original Index Record

64. When the index record is a sequence-set record, build a new set of control interval pointers.

65. When the index record is a high-level record, preserve the existing lower-level index-record pointers by copying them into the freespace immediately following the header.

66. Delete the original pointers from within the index block.

67. Convert the high-keyed index block into a new index record.

68. Write the new index record.

Notes for Diagram BH9

61 IDA019RN: IDAAQR

The index address range definition block (ARDB) that governs the range of keys that includes the new index entry's key is located. The contents of the field in the ARDB that contains the address of the next available freespace control interval is placed in the ICWA.

IDA019RJ calls **IDA019RK**

If this is the first time that space governed by the ARDB located above has been used and if sequence-set-with-data is specified, the new index record requires preformatting. Starting at the address established above, software end-of-file control intervals (zeros) are built until the end of the track on which replication is to occur is reached.

62 IDA019RJ: IDAR (calls **IDA019RZ (IDAGRB)**)

The appropriate index record is in the index buffer when IDA019RJ is entered. However, the index is freed by IDA019RJ to provide for the contingency that preformatting of succeeding index records will be required (see note 61). Accordingly, the index record must be reread.

63 IDA019RJ: DELSECT

Starting with the rightmost, or low-keyed, section entry, each section entry is moved to the left by the length necessary to eliminate the section entry's section-chaining pointer (LL field). This operation continues until the last section entry is reached. The last section entry is identified by a section chaining pointer containing zeros.

64 IDA019RJ

For sequence-set index records, a complete set of 1-byte or 2-byte pointers is built adjacent to the index header. The number of pointers equals the number of control intervals per control area.

65 IDA019RJ: MOVEPTR

For high-level index records, each index pointer in the index block is moved into the freespace between the index header and the index block, moving from left to right. The pointers within the block are not altered by this procedure.

66 IDA019RJ: DELPTR

The pointers in the index entries are eliminated by moving each index entry to the left so that it overlays the pointer field of the next higher-keyed entry.

67 IDA019RJ: BUILDREC

The following operations are performed to recreate an index record from a compressed block established by the preceding steps:

a The right end of the buffer that contains the section of the index record to the right of the split point is set to zeros.

b The first (rightmost) pointer in the group of pointers adjacent to the header is moved to the end of the index record adjoining the RDF. This becomes a dummy entry with F and L fields set to zero.

c IDA019RJ: RJE

The first (rightmost, or low-keyed) entry in the index block is eliminated. This is done to provide additional space for the Insert routine. The key was previously saved in the ICWA.

d IDA019RJ calls **IDA019RG (IDAIST)**

The key that was placed in the ICWA is front compressed (if necessary) and real values are established in the dummy entry's F and L index-entry fields.

e If there is insufficient space preceding the dummy index entry for the Insert routine to insert the key and if there is freespace to the left of the index block, the index block is moved to the left to overlay any freespace that is available. If there is no freespace available, or if after acquiring all available space there is still insufficient space to contain the key, control is returned to the caller, IDA019RI, the split point is adjusted to the left, and IDA019RI calls IDA019RJ to begin the split process again.

f If there are two or more keys remaining to be moved or if the last entry is not a dummy entry, the ICWA is adjusted for use by the Insert routine as follows:

The current key is moved into the previous key field.

The current key length is moved into the previous key length field.

The next key to the left in the index record is uncompressed and placed in the current key field.

The key length is placed in the key-length field.

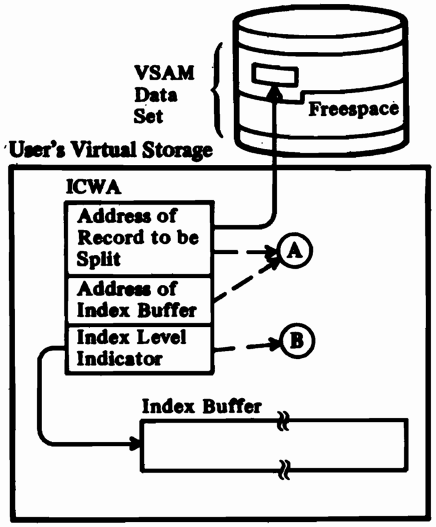
g Steps 67d, e, and f are repeated until the test in step 67f is not satisfied.

68 IDA019RJ calls **IDA019RZ (IDAWRBFR)**

The index buffer containing the new index record is written to the data set and then freed after it has been written.

Diagram BH10. Modifying a Key-Sequenced Data Set

Split an Index Record to Create Space for a New Index Entry (continued)



Reformat the Low-Keyed Block of the Original Index Record

69. Reread the original index record into the index buffer.

70. Delete the section pointers from the entire index record.

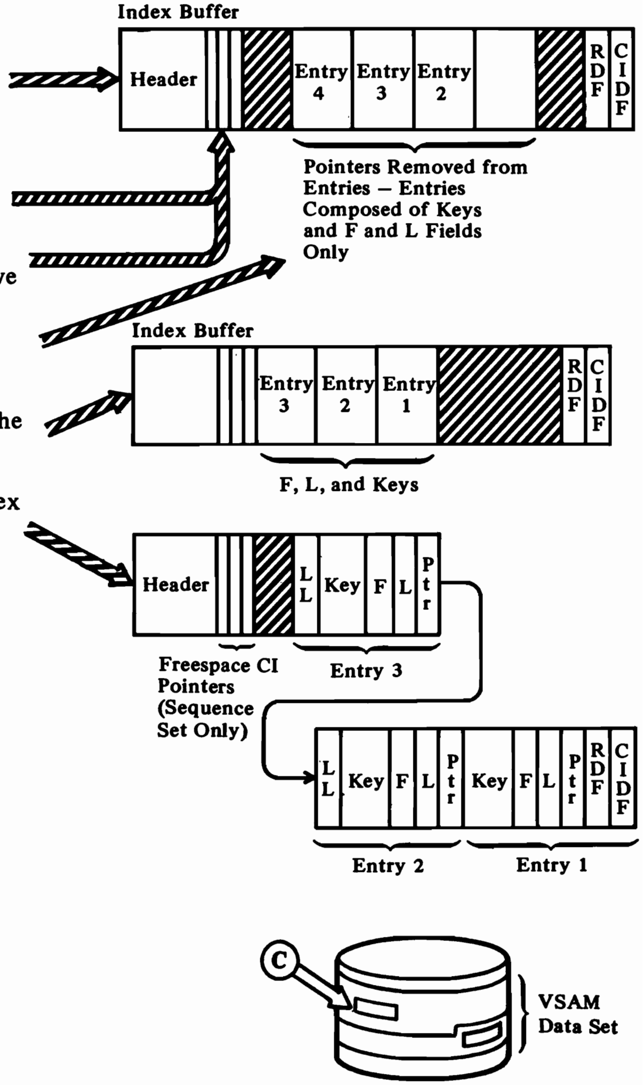
71. Compress the low-keyed index block as follows:
- Count the entries to the left of the split point and preserve their pointers.
- When the record is a sequence-set record, count entries to the right of the split point and preserve their pointers.
- Delete the entry pointers in both blocks of the index record.

72. Move the entries to the right of the split point to the left to adjoin the pointers.

73. Convert the low-keyed index block into a new index record.

74. Rewrite the original index record.

(41) or (59)



Notes for Diagram BH10

69 IDA019RJ: IDAR (calls **IDA019RZ (IDAGRB)**)

The original index record is reread.

70 IDA019RJ: DELSECT

See note 63.

71 IDA019RJ: COUNT

The number of index entries between and including the first entry to the left of the split point and the leftmost (high-keyed) entry in the index record are counted.

IDA019RJ: MOVEPTRR

If enough space exists between the header and the leftmost index entry for the entry for the entry pointer established by the count above, each index pointer in the index block is moved into the freespace, moving from left to right.

IDA019RJ: MOVEPTRL

If there is not enough space for the entry pointer, the pointers are moved by placing the leftmost pointer in the index block into the leftmost location in the freespace, and by placing the next pointer to the right into the next position to the right in the freespace until all of the pointers established by the count are moved.

High-level index record processing is not concerned with pointers that have been moved out of the index record by the split process. Sequence-set records must maintain pointers for control intervals that are freed by a control-area-split operation and retain pointers to the data control intervals that remain in the control area being split; whereas, high-level index records have pointers only to lower-level index records that are not moved by these processes.

The steps performed by MOVEPTRR and MOVEPTRL are repeated; however, in this case, the process is directed against the pointers that are contained in the index entries to the right of the split point, instead to the left.

IDA019RJ: DELPTR

See note 66.

72

Starting with the entry to the right of the split point, the index block is moved to the left until it reaches the pointers that were established by prior steps.

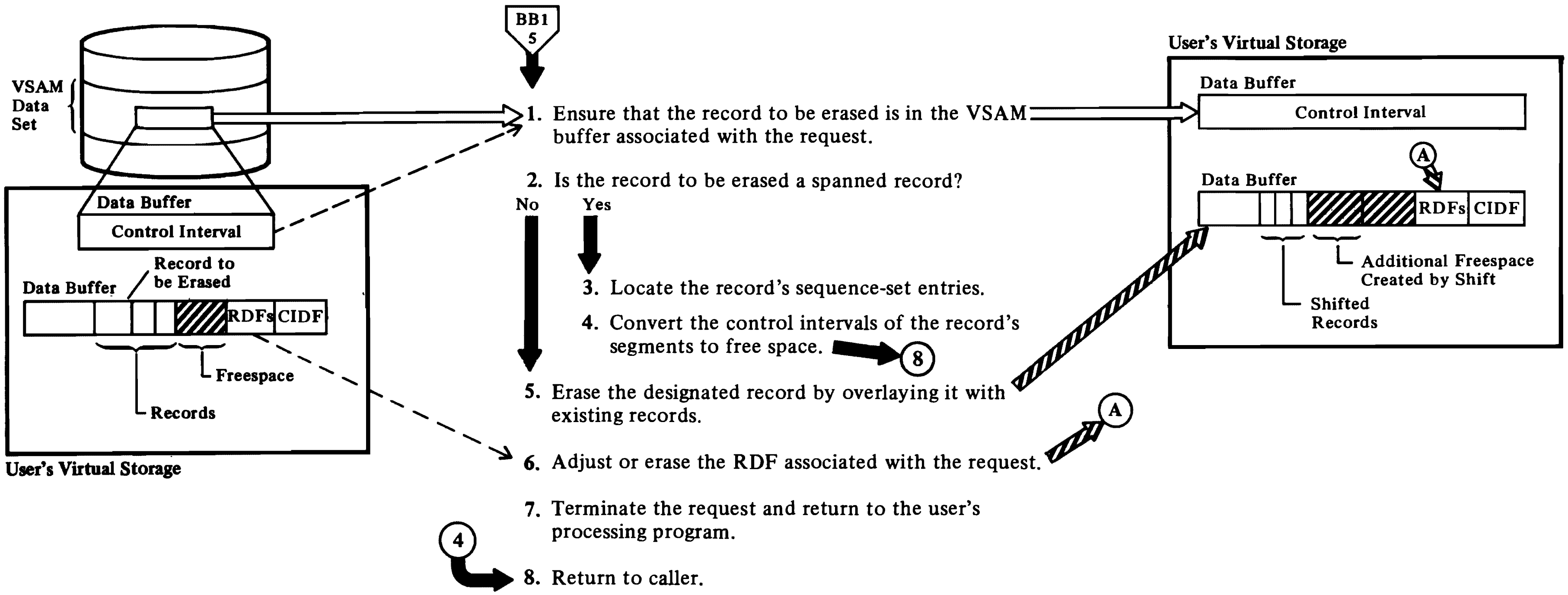
73 IDA019RJ: BUILDREC

See note 67.

74 IDA019RJ: IDAWR

The index buffer containing the revised index record is written to the data set, overlaying the original index record.

Diagram BI. ERASE Processing: Key-Sequenced



1. Ensure that the record to be erased is in the VSAM buffer associated with the request.

2. Is the record to be erased a spanned record?

No Yes

3. Locate the record's sequence-set entries.

4. Convert the control intervals of the record's segments to free space.

5. Erase the designated record by overlaying it with existing records.

6. Adjust or erase the RDF associated with the request.

7. Terminate the request and return to the user's processing program.

8. Return to caller.

Notes for Diagram BI

1 **IDA019RL**

An ERASE request must be preceded by a GET-for-update request that moves the data control interval containing the desired record into a buffer.

IDA019RU

If an upgrade set exists, the alternate indexes in it are upgraded. (See Diagram BR.)

2 **IDA019RL** calls **IDA019RS**

For spanned-record processing.

3 **IDA019RS** calls **IDA019RC**

4 **IDA019RS: CLEARSEG**

An unused buffer is obtained and filled with binary zeros and a free-space CIDF. The RBA of each segment is calculated from the index and placed in the BUFC. The buffer is written for each segment.

IDA019RS: DELSEG

Entries for all segments except the first are removed, and free-data-control-interval pointers are set up. The entry for the first segment is converted to indicate an unspanned record.

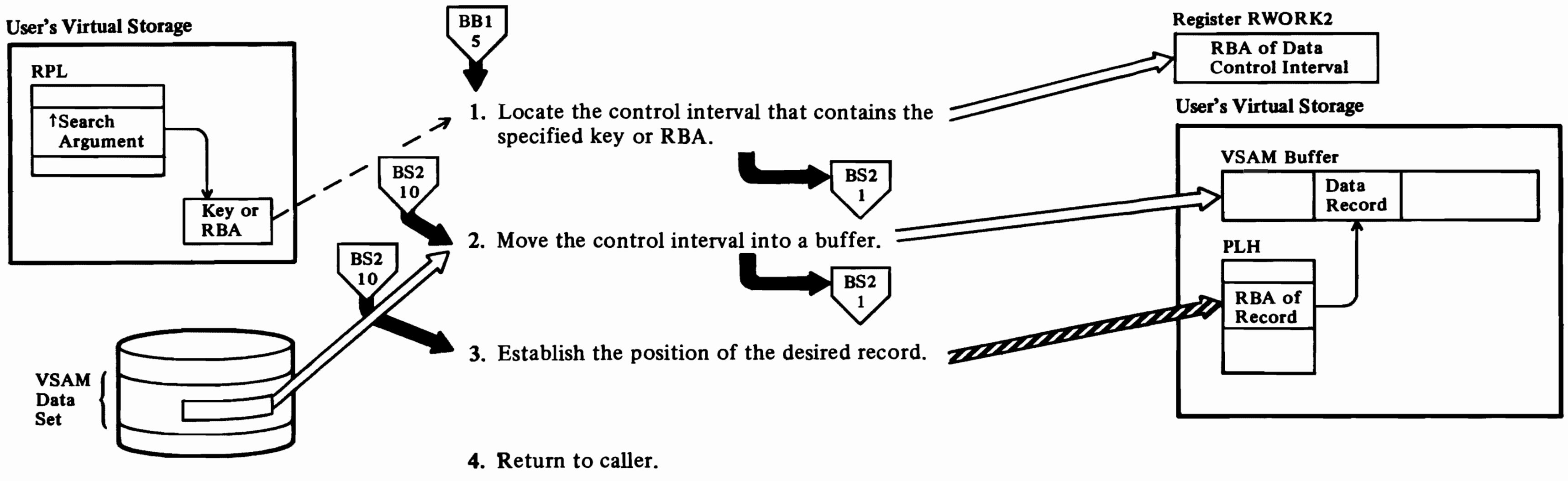
5 **IDA019RL**

6 **IDA019RL**

When the RDF is a single RDF, it is erased. When the RDF is a group RDF (that is, two RDFs are combined to refer to two or more data records of equal length), the following processing occurs:

- If the count of the records related to the group RDF is greater than two, the count is reduced by one.
- If the count of the records is equal to one (which should not occur), the two RDFs are eliminated and the CIDF is adjusted to reflect the increase in freespace in the control interval.
- If the count of the records is two, one of the two RDFs is eliminated and the CIDF is adjusted.

Diagram BJ. POINT Processing



1. Locate the control interval that contains the specified key or RBA.
2. Move the control interval into a buffer.
3. Establish the position of the desired record.
4. Return to caller.

Notes for Diagram BJ

1 **Keyed Processing—Key-Sequenced Data Set**

IDA019RA

When the request is keyed, an index search must be performed. The index level where the search begins is determined as follows:

- For skip-sequential processing, the index search starts at the sequence set. The search normally starts at the index record pointed to by the current PLH. If the PLH is invalid, the search starts at the first record in the sequence set.
- For direct processing, the search starts at the highest level of the index.

IDA019RA calls **IDA019RB** which calls **IDA019RZ (IDAGRB)**

The index record at which the search is to start is moved into an index buffer.

IDA019RB calls **IDA019RC**

The index record is searched for an entry that is greater than or equal to the search key.

IDA019RB

When the search is unsuccessful, the next record in logical sequence is searched. If the search is successful and a lower index level exists, the search is performed on the index records in the lower level.

Keyed Processing—Relative Record Data Set

IDA019RR

The relative record number that is specified as a search argument is converted to the RBA of the control interval that contains the record, plus the offset of the record in the control interval.

IDA019RR calls **IDA019RR (IDARRDRL)**

If the RBA is within the data set, the control interval's contents are retrieved. If the RBA is not within the data set, then:

- With KGE, end-of-data is indicated and positioning is established at the end of the data set
- Without KGE, no-record-found is indicated

Addressed Processing

2 **IDA019RA**

The RBA that is specified as a search argument is converted into the RBA of the boundary of the control interval that it falls within.

2 **IDA019RA** calls **IDA019RZ (IDAGRB)**

Relative Record Processing

IDARRDRL calls **IDA019RZ (IDAGRB)**

The control interval is read by RBA.

3 **IDA019RA**

The control interval is scanned to determine whether the key or RBA provided as a search argument is within the retrieved control interval. (Note: The RBA must represent a valid record boundary within the control interval.)

When the key search is unsuccessful, a test is made to determine whether a control interval split has been performed by another request-string operating concurrently with the current request. If a split has occurred, processing returns to step 1 to perform a new index search.

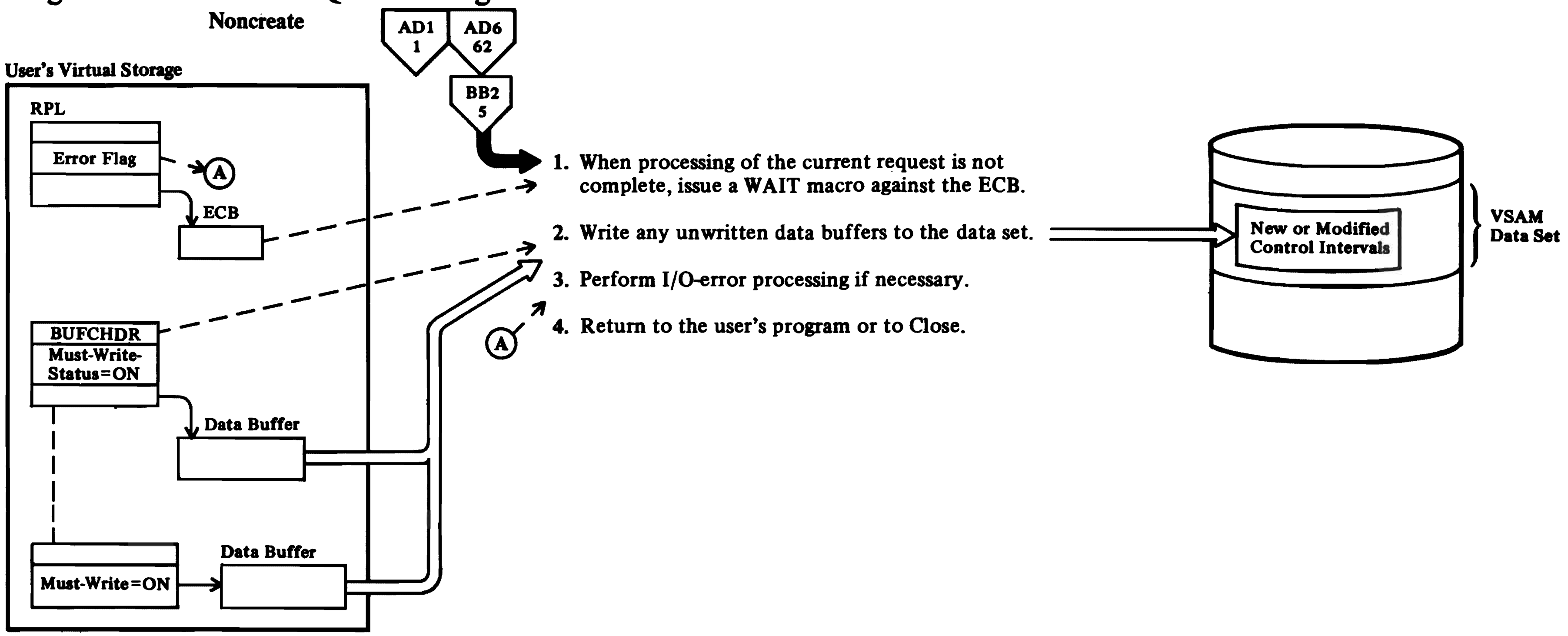
Relative Record Processing

IDA019RR: IDARRDRL

Positioning is established by saving in the PLH pointers to the record and its RDF and the RBA of the control interval.

Diagram BK1. ENDREQ Processing

Noncreate



1. When processing of the current request is not complete, issue a WAIT macro against the ECB.
2. Write any unwritten data buffers to the data set.
3. Perform I/O-error processing if necessary.
4. Return to the user's program or to Close.

Notes for Diagram BK1

1 **IDA019R1: FINDOPLH**

The placeholder (PLH) for the request string associated with the ENDREQ request is located by searching the placeholder list for a placeholder that points to the RPL identified by the ENDREQ.

IDA019RP: IDAENDRQ

Other RPLs (if any) in the request string are prevented from being processed by setting a flag in the placeholder that indicates that an ENDREQ request is being processed. (Note: Once a request-string starts processing, it continues until all of the RPLs in the string are processed or until an ENDREQ is issued. When an ENDREQ is issued, processing against the request-string is terminated when processing of the current RPL in the string has completed.) If the current request is not complete, the WAIT is issued to ensure completion.

2 **IDA019RP: IDAENDRQ**

Before performing any I/O, the processing is forced into synchronous mode to ensure that control is not returned to the user until I/O associated with the ENDREQ request is completed. When I/O is completed, asynchronous processing is restored if the processing was previously asynchronous.

IDA019RP: IDAENDRQ (calls **IDA019RZ (IDAWRBFR)**)

All unwritten data buffers associated with the current placeholder are written.

3 **IDA019RP:** calls **IDA019R5**

The buffer control block (BUFC) chain for the I/O-Management block (IOMB) in error is searched for a BUFC with an error indicator.

IDA019R1: R1ENDREQ (calls **IDA019R5)**

Error conditions are analyzed and an error message is built.

IDA019RP calls **IDA019R5 (IDAEXITR)**

For processing if a SYNAD routine exists.

4 **IDA019RP: IDAENDRQ** (calls **IDA019RZ (IDASBF)**)

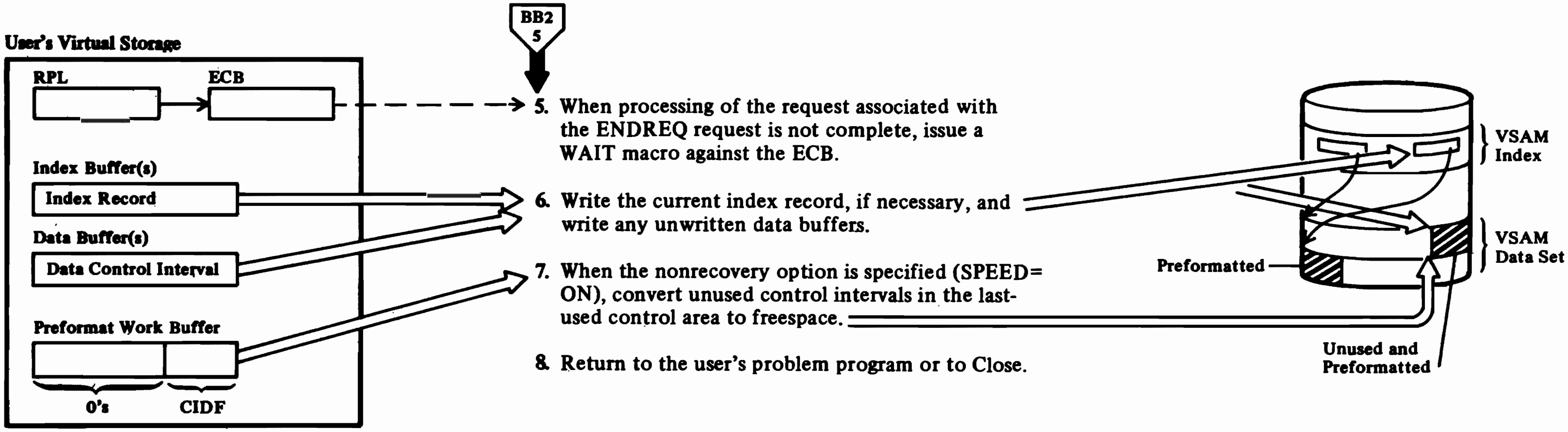
Excess data buffers are released from the current placeholder.

IDA019RP: IDAENDRQ

The placeholder is released from the current request string.

Diagram BK2. ENDREQ Processing

Create



5. When processing of the request associated with the ENDREQ request is not complete, issue a WAIT macro against the ECB.

6. Write the current index record, if necessary, and write any unwritten data buffers.

7. When the nonrecovery option is specified (SPEED=ON), convert unused control intervals in the last-used control area to freespace.

8. Return to the user's problem program or to Close.

Notes for Diagram BK2

5 IDA019R1: FINDOPLH

The placeholder for the request string associated with the ENDREQ request is located by searching the placeholder list for a placeholder that points to the RPL identified by the ENDREQ.

IDA019RP: IDAENDRQ

Other RPLs (if any) in the request string are prevented from being processed by setting a flag in the placeholder that indicates that an ENDREQ request is being processed. (Note: Once processing for a request-string starts, it continues until all of the RPLs in the string are processed or until an ENDREQ is issued. When an ENDREQ is issued, processing against the request-string is terminated when processing of the current RPL in the string has completed.) If the current request is not complete, the WAIT is issued to ensure completion.

6

The processing for step 6 ensures that the index entry for the last data control interval in the current data buffer for the current control area will fit in the index record for the current control area. Otherwise, when processing is resumed and when the dummy entry in the index record does not have space for the key, the data control interval would have to be moved to a new control area and have its index entry placed in the index record for the new control area.

IDA019RP calls **IDA019RG**

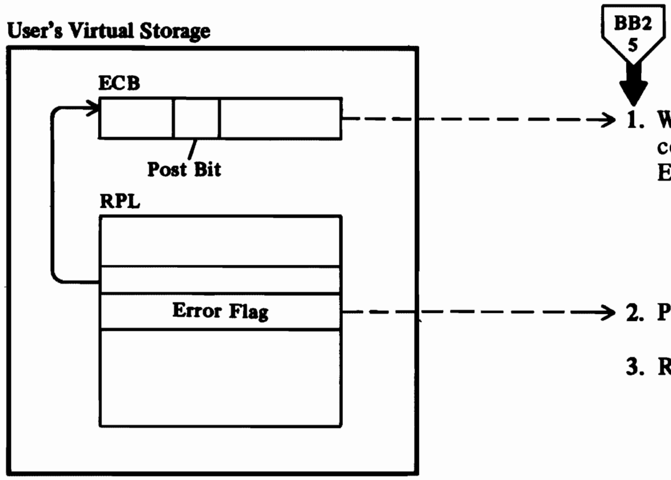
Before writing the index buffer, the following processing is performed: IDA019RG checks the leftmost entry, a dummy entry for the current control interval, in the index record to determine whether a maximum length key will fit in the remaining index record freespace. If there is adequate space to insert a key, IDA019RG writes out the current index record and frees the index-create work area(s) (ICWAs).

If there is inadequate space to contain a key for the control interval in the current data buffer, IDA019RP calls IDA019SA, which recalls IDA019RG, in order to have the entry inserted into the index record. IDA019RG returns a no-fit indicator to IDA019SA, which forces an end-of-control-area situation for IDA019SA (EOCA) processing. In response to the no-fit indicator, IDA019SA (EOCA) writes out any full data buffers (less the current data buffer) to the data set and acquires a new control area.

7 IDA019RP calls **IDA019RZ (IDAWRBFR)**

8 IDA019RP calls **IDA019RK**

Diagram BL. CHECK Processing



1. When the request's ECB is not posted as being complete, a WAIT macro is issued against the ECB.

2. Perform error processing if necessary.

3. Return to the user's processing program.

Notes for Diagram BL

1 IDA019R1: FINDOPLH

The placeholder for the request-string associated with the CHECK request is located by searching the placeholder list for a placeholder that points to the RPL identified by the ENDREQ.

IDA019R1: R1CHECK

A WAIT macro is issued to ensure that the asynchronous request, for which the CHECK was issued, has completed.

2 IDA019R1 calls **IDA019R5**

The buffer control block (BUFC) chain for the I/O block (IOB) in error is searched for a BUFC with an error indicator.

Error conditions are analyzed and an error message is built.

IDA019R1 calls **IDA019R5 (IDAEXITR)**

For processing if a SYNAD routine exists.

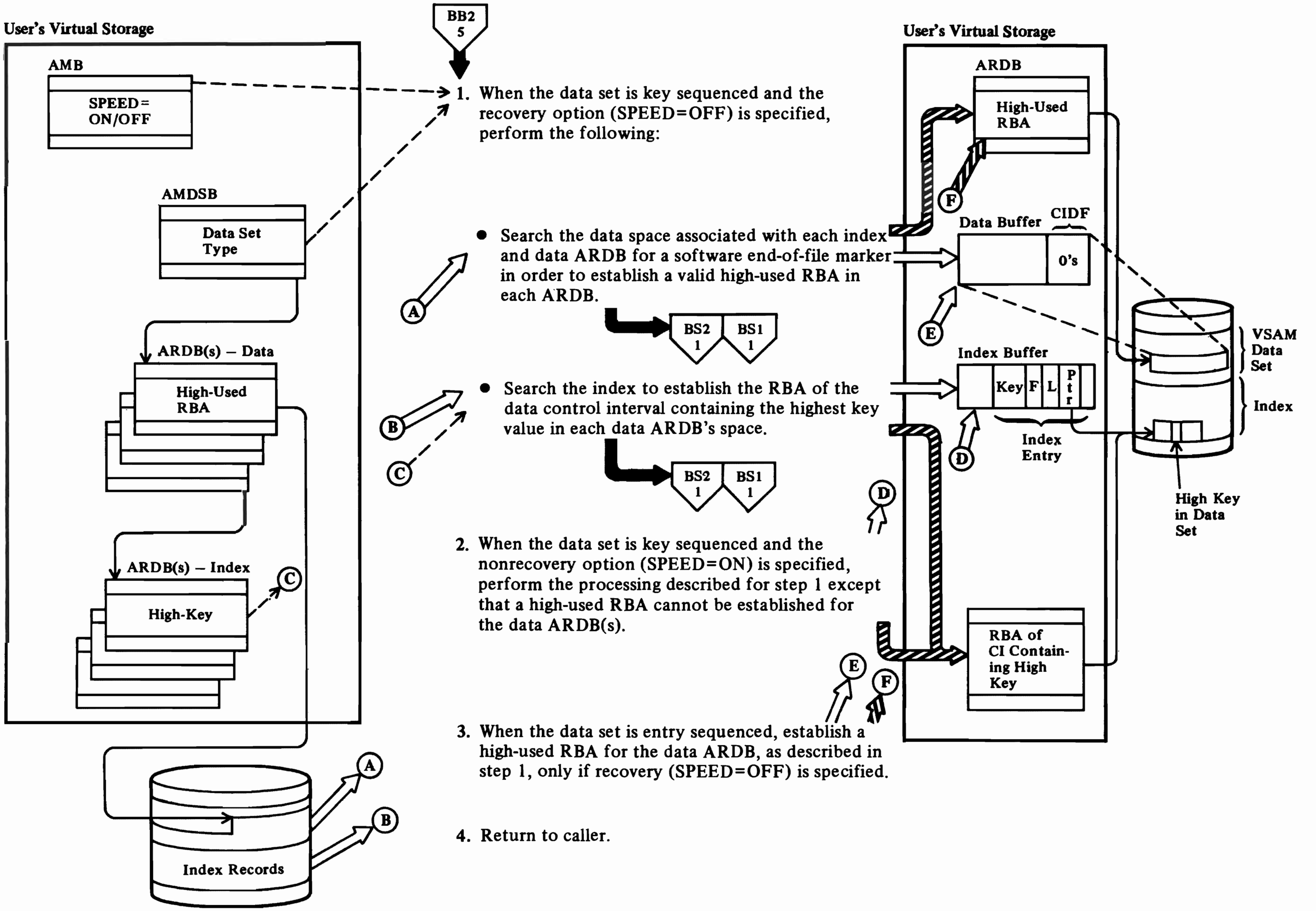
3 IDA019R1: R1CHECK

The check process is repeated for each RPL (if any) in the RPL-string associated with the RPL that the CHECK was originally issued against.

The placeholder is released if necessary.

The placeholder remains associated with the current request-string unless the processing is direct. For direct processing, the next request must be repositioned to an address in the data set. For sequential or skip-sequential processing, the positioning information established by a prior request is used by the succeeding request.

Diagram BM. VERIFY Processing



1. When the data set is key sequenced and the recovery option (SPEED=OFF) is specified, perform the following:

- Search the data space associated with each index and data ARDB for a software end-of-file marker in order to establish a valid high-used RBA in each ARDB.

- Search the index to establish the RBA of the data control interval containing the highest key value in each data ARDB's space.

2. When the data set is key sequenced and the nonrecovery option (SPEED=ON) is specified, perform the processing described for step 1 except that a high-used RBA cannot be established for the data ARDB(s).

3. When the data set is entry sequenced, establish a high-used RBA for the data ARDB, as described in step 1, only if recovery (SPEED=OFF) is specified.

4. Return to caller.

Notes for Diagram BM

1

- **IDA019R8** calls **IDA019RO**

  Other requests are prevented from adding records into the data space controlled by the ARDB that is being examined by Verify.

  IDA019RO calls **IDA019RZ (IDAGRB)** and **IDA019RZ (IDAFREEB)**

  Starting with the high-used key in an ARDB, retrieve and release successive control intervals until a software end-of-file marker, that is, a CIDF set to zeros, is found. The RBA of the control interval containing the software end-of-file marker is used to update the high-used RBA in the ARDB.

- **IDA019RO** calls **IDA019RB,** which calls **IDA019RZ (IDAGRB)**

  An index record is moved into a buffer. (Note: The search starts at the highest level of the index.)

  IDA019RB calls **IDA019RC**

  The index record is searched for a key that is greater than or equal to the search key.

  IDA019RB calls **IDA019RZ (IDAFREEB)**

  If the search is not satisfied or if lower-level index records exist (that is, the current level is not the sequence set), the current buffer is released. (IDAO19RB then calls IDAO19RZ (IDAGRB) to retrieve another index record and the search process repeats itself.)

  IDA019RO

  When the search is successful, the pointer in the index entry is converted into a valid RBA and moved into the ARDB.

2

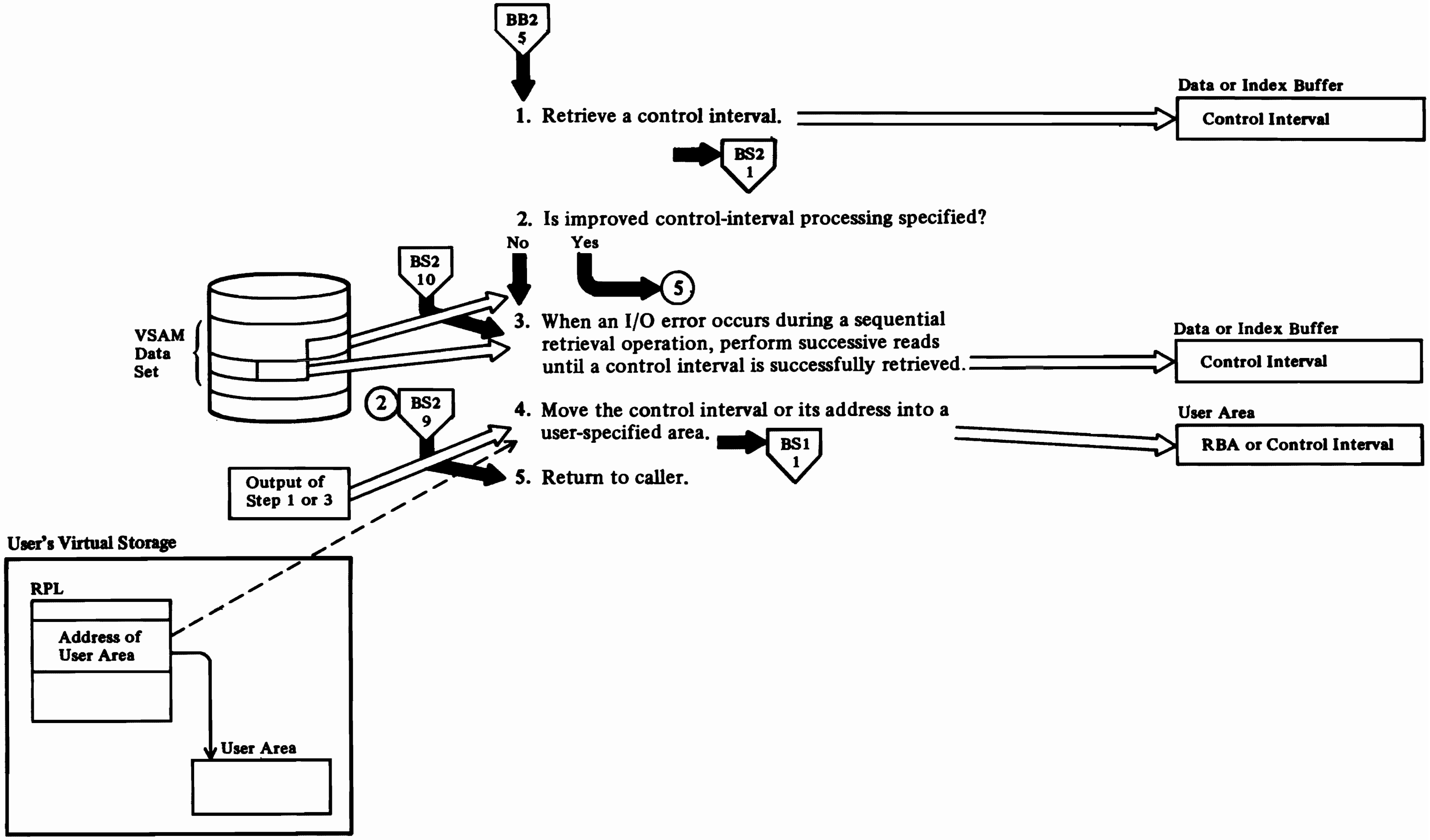
See note 1.

3

See note 1.

Diagram BN1. Processing by Control Interval

GET or GETIX Processing (Control Interval Retrieval)



1. Retrieve a control interval.

2. Is improved control-interval processing specified?

3. When an I/O error occurs during a sequential retrieval operation, perform successive reads until a control interval is successfully retrieved.

4. Move the control interval or its address into a user-specified area.

5. Return to caller.

Notes for Diagram BN1

1 **Normal Control-Interval Processing (NCI)**

Direct Request Processing

IDA019R8 calls **IDA019RZ (IDASBF)**

When the prior request was sequential, excess buffers in the chain of buffers associated with the current placeholder (PLH) are released.

IDA019R8 calls **IDA019RZ (IDAGRB)**

The control interval at a user-specified address is retrieved.

Sequential Retrieval (GET) Processing Only

IDA019R8 calls **IDA019RZ (IDAGRB)**

When this is the first request after Open, the control interval at a user-specified address is retrieved. Subsequent control intervals are retrieved sequentially by IDA019RZ (IDAGNXT).

Improved Control-Interval Processing (ICI)

IDA019S1

The request is decoded. A placeholder is obtained. If the request is for update, exclusive control of the control interval is obtained.

IDA019S1 calls **IDA019S3**

The control interval at a user-specified address is retrieved.

3 **IDA019R8** calls **IDA019RZ (IDAGNXT)**

4 **IDA019R8** calls **IDA019RP (IDATJXIT)**

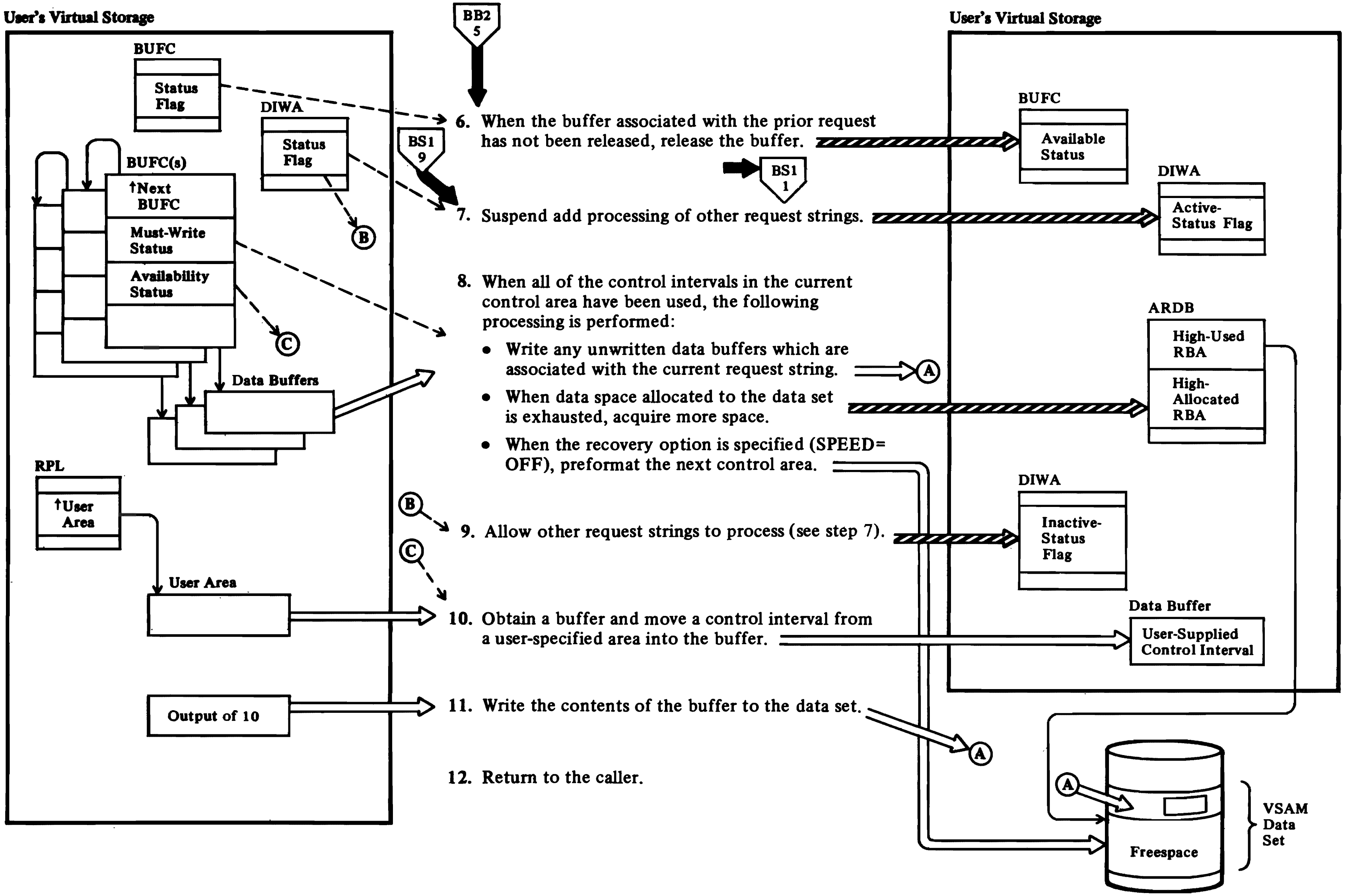
Journalling is performed when a journal exit routine exists.

IDA019R8 calls **IDA019RZ (IDAFREEB)**

For normal direct requests, the buffer associated with the request is released before returning to the caller.

Diagram BN2. Processing by Control Interval

PUT-Create Processing (Add a New Control Interval)



6. When the buffer associated with the prior request has not been released, release the buffer.

7. Suspend add processing of other request strings.

8. When all of the control intervals in the current control area have been used, the following processing is performed:
   - Write any unwritten data buffers which are associated with the current request string.
   - When data space allocated to the data set is exhausted, acquire more space.
   - When the recovery option is specified (SPEED=OFF), preformat the next control area.

9. Allow other request strings to process (see step 7).

10. Obtain a buffer and move a control interval from a user-specified area into the buffer.

11. Write the contents of the buffer to the data set.

12. Return to the caller.

Notes for Diagram BN2

6 **IDA019R8** calls **IDA019RZ (IDAFREEB)**

7 **IDA019R8**

The DIWA, a serially reuseable resource, is examined to determine whether another request string is in control. When the DIWA is active, processing of the current request is deferred. When the DIWA is inactive, it is given an active status, which effectively defers processing of other requests that may be competing for this resource.

8 **IDA019R8** calls **IDA019RZ (IDASBF)**

IDA019R8 calls **IDA019R5 (IDAEOVIF)**

IDA019R8 calls **IDA019RK**

9 **IDA019R8**

See note 7.

10 **IDA019R8** calls **IDA019RZ (IDAGNNFL)**

An available buffer is assigned to the request and written if necessary.

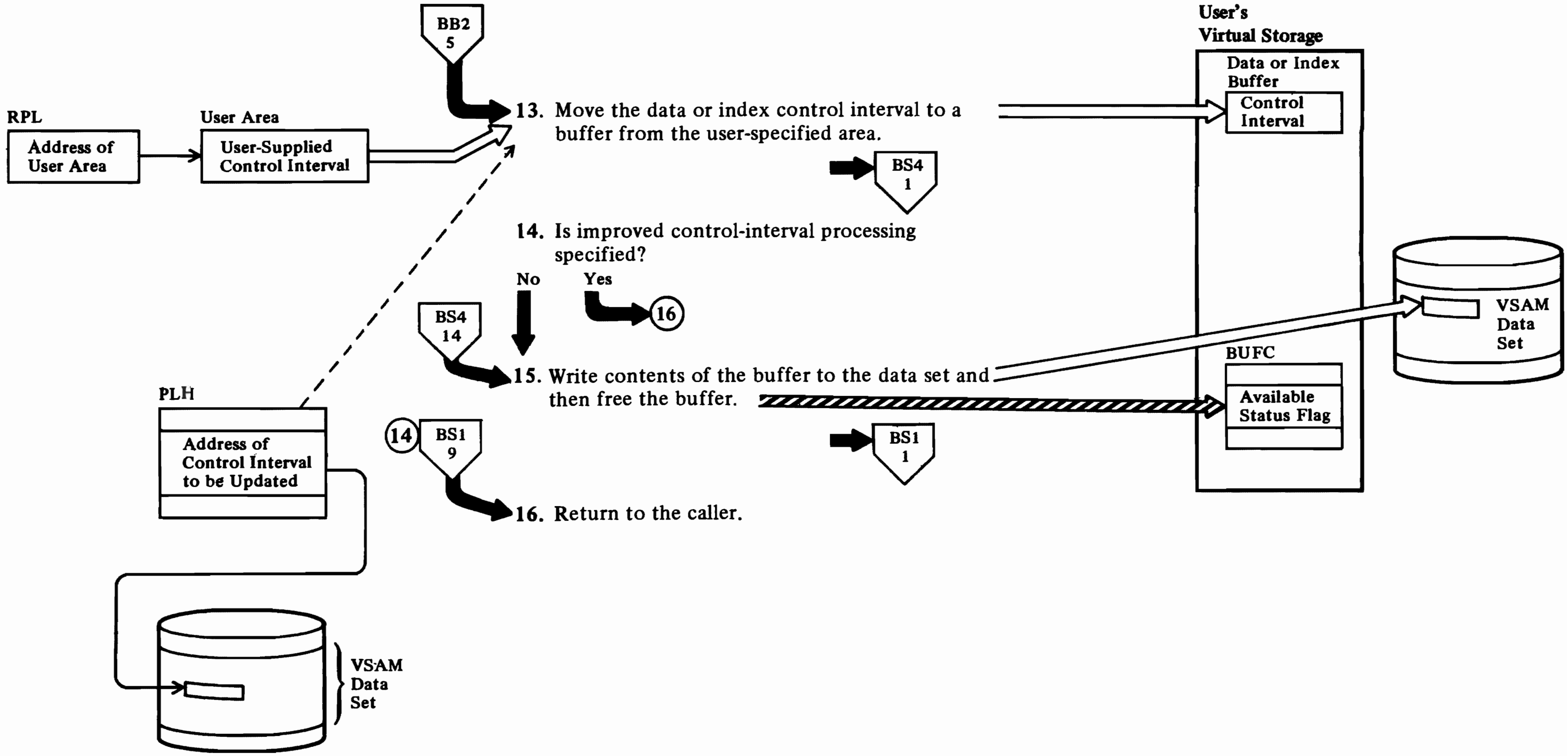
IDA019R8

The user-specified control interval is moved into the buffer.

11 **IDA019R8** calls **IDA019RZ (IDAWRBFR)**

Diagram BN3. Processing by Control Interval

PUT- or PUTIX-Update Processing (Update a Control Interval)



13. Move the data or index control interval to a buffer from the user-specified area.

14. Is improved control-interval processing specified?

15. Write contents of the buffer to the data set and then free the buffer.

16. Return to the caller.

Notes to Diagram BN3

13 **Normal Control-Interval Processing** (NCI)

The request is invalid if any of the following conditions exist:

- The record length is not equal to control interval size.
- A PUT request specifies LOCATE mode.
- A PUTIX request doesn't specify update.
- A stand-alone PUT-for-update is issued without specifying user buffering. (Note:"Stand-alone" implies that the PUT-for-update is *not* preceded by a GET-for-update.)

The address of the control interval to be updated is established as follows:

IDA019R8 calls **IDA019RW (IDAFRBA)**

For sequential requests, the new address calculation is based on information in the placeholder.

For direct requests, the address is taken from the RPL.

Improved Control-Interval Processing (ICI)

IDA019S1

The request is decoded. A placeholder is obtained.

IDA019S1 calls **IDA019S3**

The control interval specified by the RPL is written.

15 **IDA019R8** calls **IDA019RP (IDATJXIT)**

Before writing the new control interval, journaling is performed if a journal exit routine exists.

IDA019R8 calls **IDA019RZ (IDAWRBFR)**

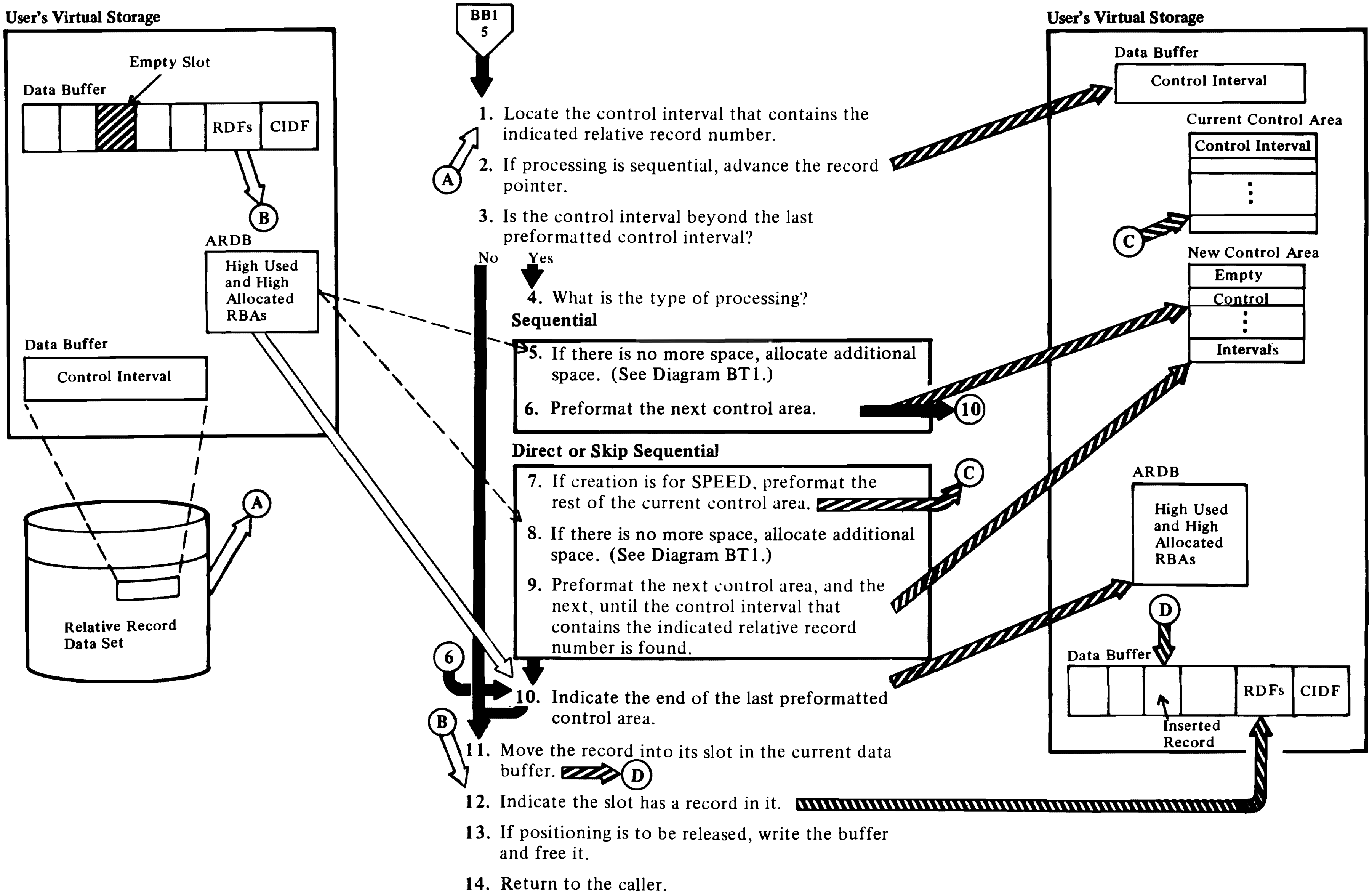
The new control interval is written to the data set.

IDA019R8 calls **IDA019RZ (IDAFREEB)**

The buffer is released.

Diagram BO1. Creating or Modifying a Relative Record Data Set

PUT-Insert Processing



1. Locate the control interval that contains the indicated relative record number.
2. If processing is sequential, advance the record pointer.
3. Is the control interval beyond the last preformatted control interval?
4. What is the type of processing?

Sequential

5. If there is no more space, allocate additional space. (See Diagram BT1.)
6. Preformat the next control area.

Direct or Skip Sequential

7. If creation is for SPEED, preformat the rest of the current control area.
8. If there is no more space, allocate additional space. (See Diagram BT1.)
9. Preformat the next control area, and the next, until the control interval that contains the indicated relative record number is found.
10. Indicate the end of the last preformatted control area.
11. Move the record into its slot in the current data buffer.
12. Indicate the slot has a record in it.
13. If positioning is to be released, write the buffer and free it.
14. Return to the caller.

Notes for Diagram BO1

1 **Direct or Skip Sequential Processing**

IDA019RQ calls **IDA019RR (IDARRDRL)**

If the data set is not being created, or it is being created and the control interval is in an existing control area, the control interval is read and the record pointer is set in the PLH.

Sequential Creation

IDA019RR calls **IDA019RZ (IDAFREEB, IDAGNXT)**

If there are no more slots in the current control interval and the next control interval has already been written, the next control interval is read into a buffer.

IDA019RQ calls **IDA019RZ (IDAGRB)**

If there are no more slots in the current control interval and the next control interval has already been written, the next control interval is read into an insert buffer.

Sequential Insertion

IDA019RQ calls **IDA019RR (IDARRDRL)**

If the previous request was a POINT for KGE (key greater than or equal), its search argument is used to retrieve the control interval as though for a direct request.

IDA019RQ calls **IDA019RZ (IDAFREEB, IDAGNXT)**

Otherwise, if there are no more slots in the current control interval, the next control interval is read with read-ahead buffering.

3 **Direct or Skip Sequential Creation**

IDA019RQ calls **IDA019RZ (IDAFREEB, IDAGNNFL)**

If the control interval is not in an existing control area, a buffer is obtained and formatted with empty slots.

Sequential Creation

IDA019RQ calls **IDA019RZ (IDAGNNFL)**

If there are no more slots in the current control interval and the next control interval is not in an existing control area, a buffer is obtained and formatted with empty slots.

5 **IDA019RQ** calls **IDA019R5 (IDAEOVIF)**

End of Volume does the allocation.

6 **IDA019RQ** calls **IDA019RK**

Each control interval is formatted with empty slots.

7 See note for step 6. If the requested control interval is among those formatted, processing continues at step 10.

8 See note for step 5.

9 See note for step 6. During preformatting of control areas, End of Volume might have to be called to allocate additional space. (See Diagram BT1.)

10 **IDA019RQ**

The high used RBA is at the beginning of the next control area—except for creation with the SPEED option, for which it is at the beginning of the next control interval.

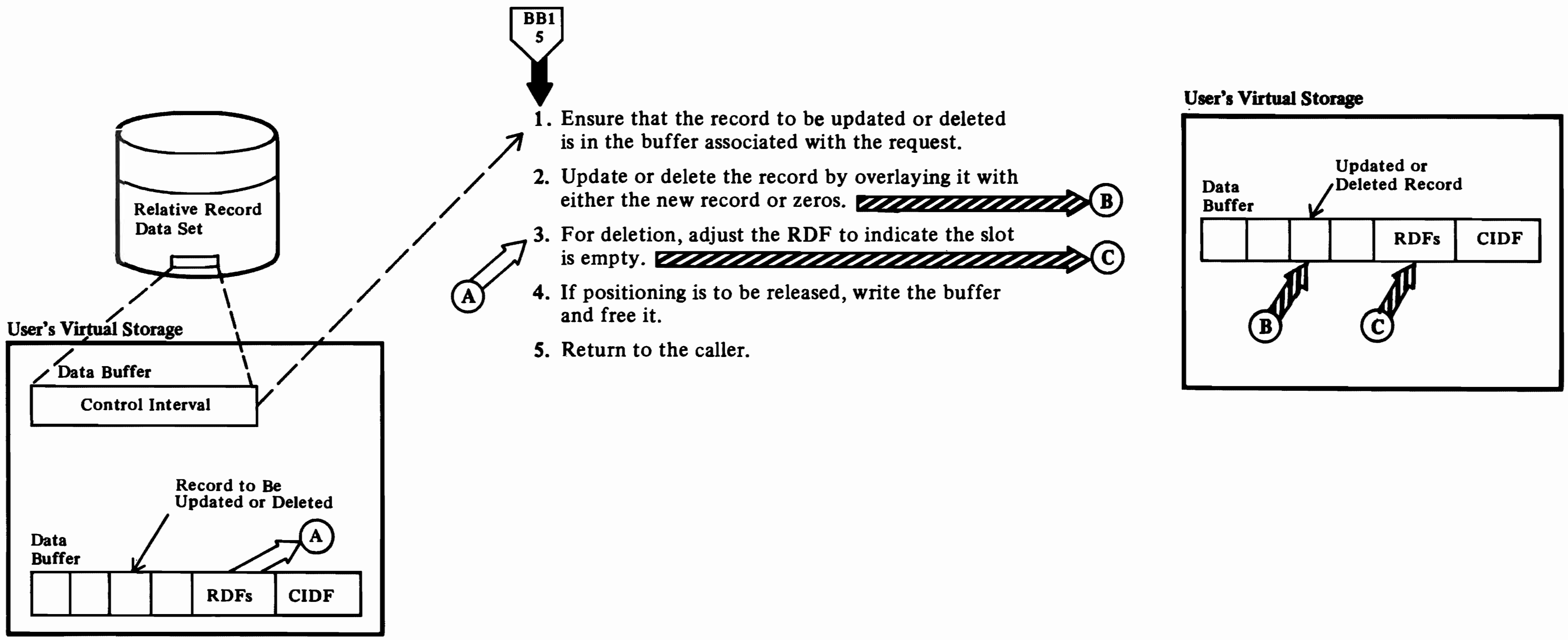
11 **IDA019RQ**

If the slot into which the record is to be moved isn't empty, a duplicate-record error is indicated.

12 The codes that indicate whether a slot is empty or filled are given under "VSAM Data Set Format" in "Data Areas."

13 **IDA019RQ** calls **IDA019RZ (IDAWRBFR, IDAFREEB)**

Diagram BO2. Modifying a Relative Record Data Set



1. Ensure that the record to be updated or deleted is in the buffer associated with the request.
2. Update or delete the record by overlaying it with either the new record or zeros.
3. For deletion, adjust the RDF to indicate the slot is empty.
4. If positioning is to be released, write the buffer and free it.
5. Return to the caller.

Notes for Diagram BO2

1 **IDA019RQ**

A PUT-update or ERASE request must be preceded by a GET-update request.

2 **IDA019RQ**

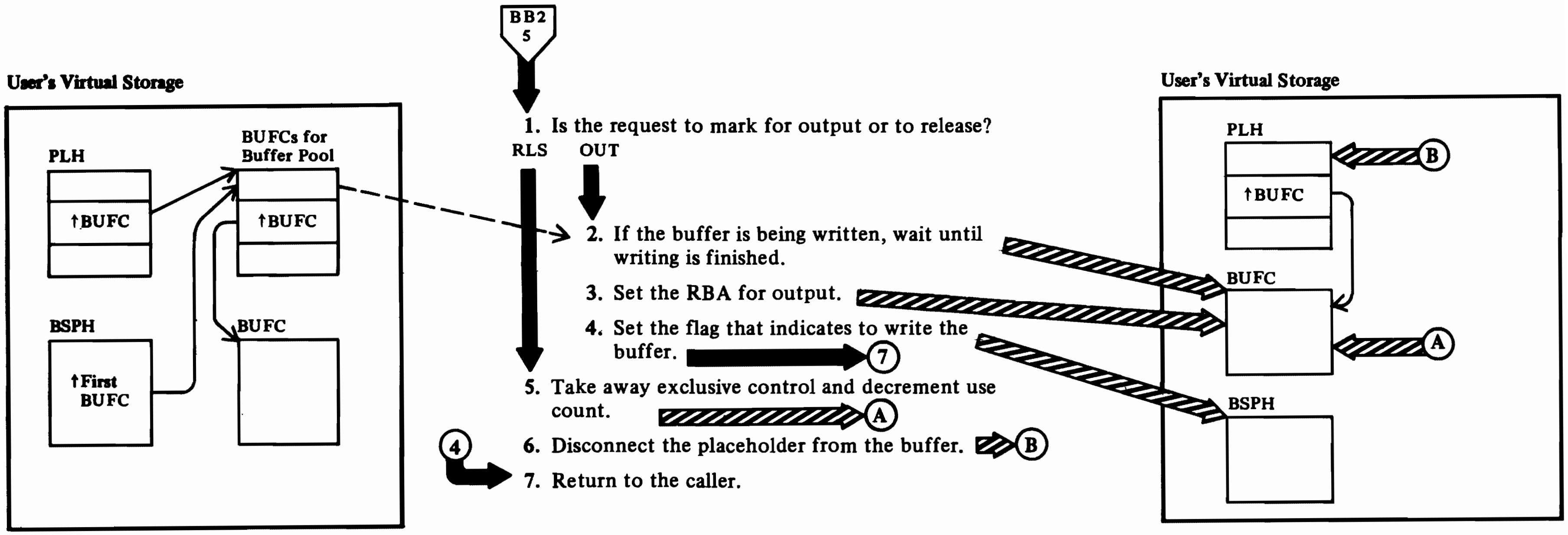
For PUT-update processing, the length of the updated record must be the same as that of the original.

3 **IDA019RQ**

The codes that indicate whether a slot is empty or filled are given under "VSAM Data Set Format" in "Data Areas."

4 **IDA019RQ** calls **IDA019RZ (IDAWRBFR, IDAFREEB)**

Diagram BP1. MRKBFR: Marking a Buffer in the Buffer Pool with Shared Resources



1. Is the request to mark for output or to release?
2. If the buffer is being written, wait until writing is finished.
3. Set the RBA for output.
4. Set the flag that indicates to write the buffer.
5. Take away exclusive control and decrement use count.
6. Disconnect the placeholder from the buffer.
7. Return to the caller.

Notes for Diagram BP1

1 **IDA019RY: MRKBF**

2 **IDA019RY** calls **IDA019R5 (IDADRQ)**

The request is deferred until the buffer has been written.

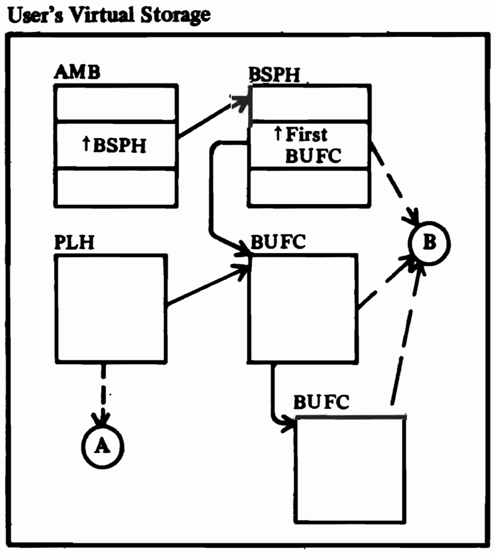
3 **IDA019RY**

The RBA of the control interval to be written is assigned to the buffer that contains the control interval.

6 **IDA019RY**

The placeholder is marked invalid.

Diagram BP2. WRTBFR: Writing a Buffer in the Buffer Pool (With Shared Resources)



BB2 5

1. If a placeholder holds a position in the buffer, free the placeholder.
2. Determine the type of request.

TYPE=DS

3. Write all buffers marked for output for the specified AMB.
4. Indicate that each buffer is empty.

TYPE=CHK

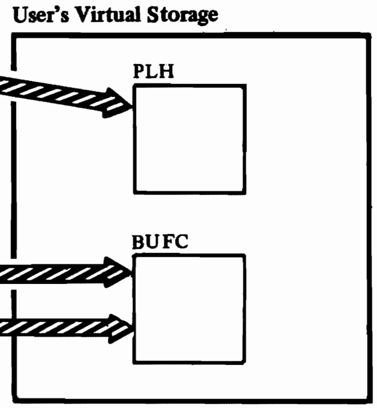
5. Write all buffers marked for output for the specified transaction ID.
6. Ignore errors.

TYPE=TRN

7. Write all buffers marked for output for the specified transaction ID.

TYPE=ALL

8. Write all buffers marked for output.
9. Return to the caller.



Notes for Diagram BP2

3 **IDA019RY: WRTBF** calls **IDA019RY (WRBFR)**

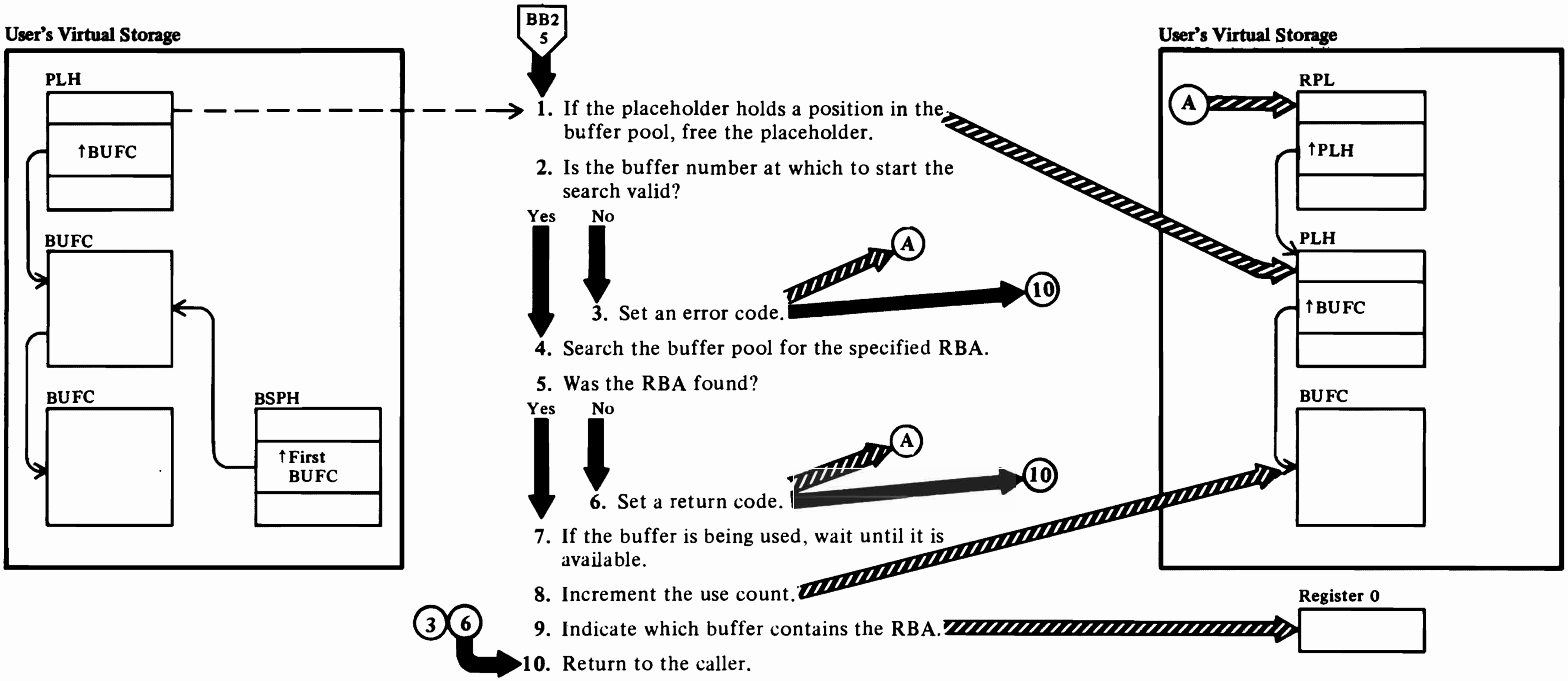
WRBFR writes the buffers associated with the BUFCs indicated by the request.

5 Same as note for step 3.

7 Same as note for step 3.

8 Same as note for step 3.

Diagram BP3. SCHBFR: Searching the Buffer Pool (With Shared Resources)



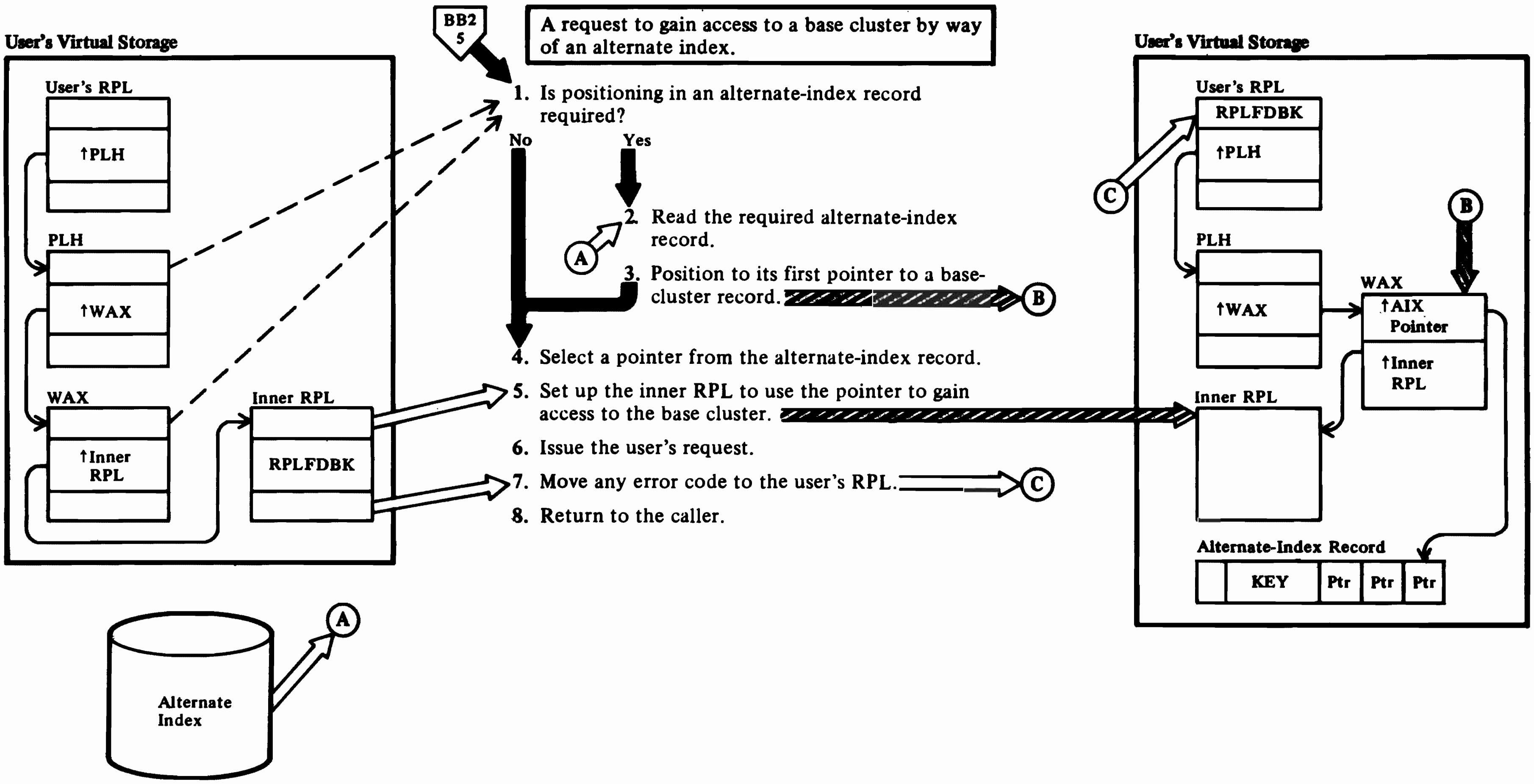
1. If the placeholder holds a position in the buffer pool, free the placeholder.
2. Is the buffer number at which to start the search valid?
3. Set an error code.
4. Search the buffer pool for the specified RBA.
5. Was the RBA found?
6. Set a return code.
7. If the buffer is being used, wait until it is available.
8. Increment the use count.
9. Indicate which buffer contains the RBA.
10. Return to the caller.

Notes for Diagram BP3

7 **IDA019RY** calls **IDA019R5 (IDADRQ)**

The request is deferred until the buffer has been processed.

Diagram BQ. Processing a Path



A request to gain access to a base cluster by way of an alternate index.

1. Is positioning in an alternate-index record required?

   No / Yes

2. Read the required alternate-index record.
3. Position to its first pointer to a base-cluster record.
4. Select a pointer from the alternate-index record.
5. Set up the inner RPL to use the pointer to gain access to the base cluster.
6. Issue the user's request.
7. Move any error code to the user's RPL.
8. Return to the caller.

Notes for Diagram BQ

1 **IDA019RX**

If the request is a PUT or a POINT, no positioning is required. If the request is a GET, positioning could already have been established by a previous GET.

2 **IDA019RX** calls **IDA019R4**

3 **IDA019RX**

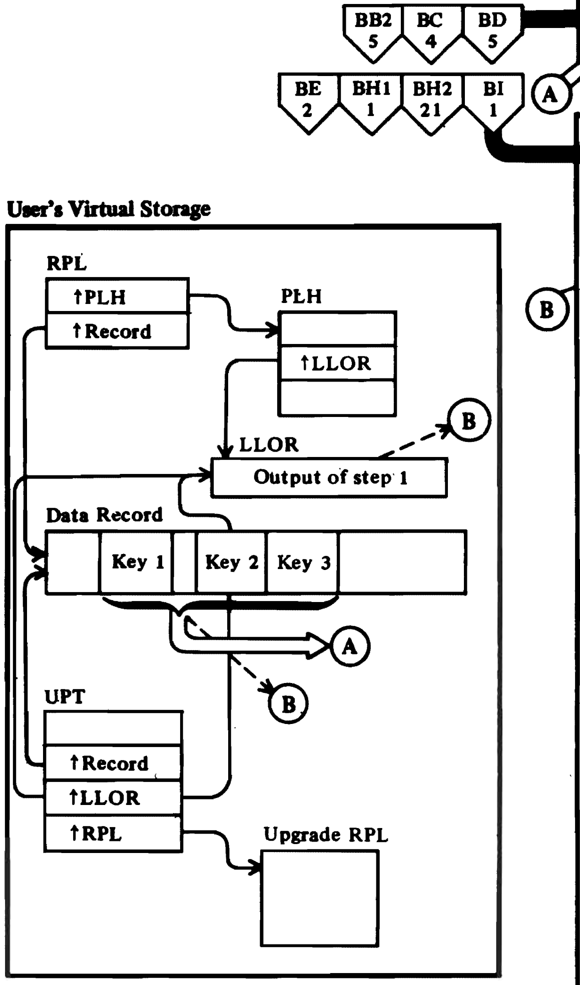
The PLH identifies the alternate-index record positioned at; the WAX indicates the pointer within the alternate-index record positioned at. The alternate-index record contains either prime-key pointers (for a key-sequenced base cluster) or RBA pointers (for an entry-sequenced base cluster).

4 **IDA019RX**

5 **IDA019RX**

The inner RPL is built by VSAM Open. It is used to read the alternate index and to gain access to the base cluster.

Diagram BR. Upgrading Alternate Indexes



For GET-Update:

1. Save the portion of the data record that contains all of its key fields. → 14

For PUT or ERASE:

2. Is upgrading for PUT-update?

No Yes

3. Has any alternate key field changed?

Yes No → 14

4. Get exclusive control of the upgrade table.

Repeat steps 5-12 for each alternate index in the upgrade set.

5. Is there a new alternate key to add to the alternate index?

No Yes

6. Read the alternate-index record that has the new key, if there is a record, or build a new record.
7. Add to the alternate-index record a pointer to the data record.
8. Write the alternate-index record.

9. Is there an old alternate key to delete from the alternate index?

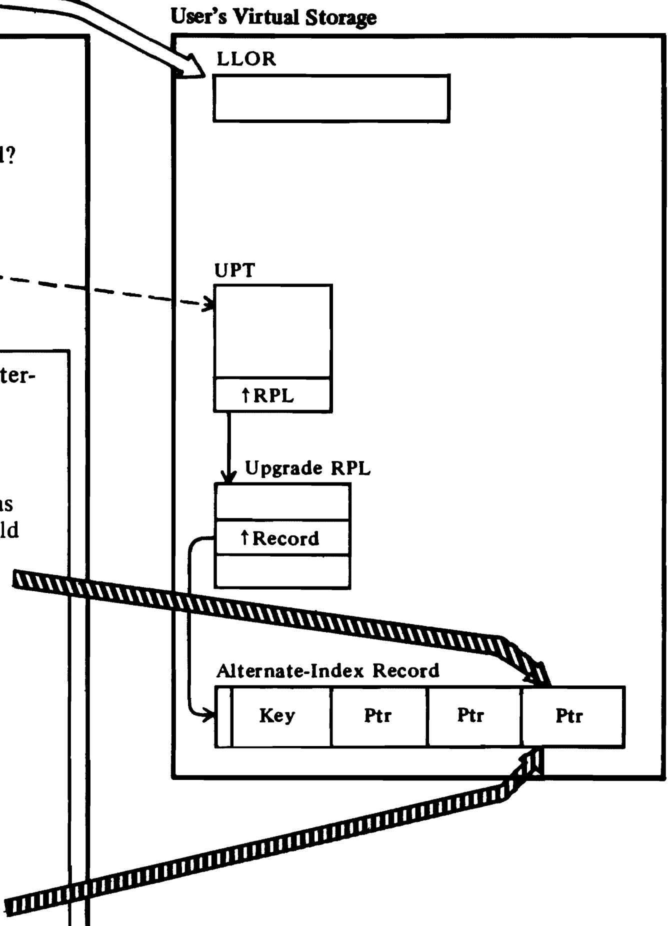
No Yes

10. Read the alternate-index record that has the old key.
11. Remove from the alternate-index record the pointer to the data record.
12. If the alternate-index record contains no other pointers, delete the record; otherwise write it.

13. Release the upgrade table from exclusive control.

1 3

14. Return to the caller.



Notes for Diagram BR

1 IDA019RU

The LLOR is just large enough to contain the "least length of the data record" that contains the record's prime key, if any, and all of its alternate keys.

5 IDA019RU

For ERASE, there can be no new alternate key to add. For PUT-insert, there is a new key. For PUT-update, there is a new key if the alternate key for the alternate index being upgraded has changed.

6 IDA019RU calls **IDA019R4**

7 IDA019RU

8 IDA019RU calls **IDA019R4**

9 IDA019RU

For PUT-insert, there can be no alternate key to delete. For ERASE, there is a key to delete. For PUT-update, there is a key to delete if the alternate key for the alternate index being upgraded has changed.

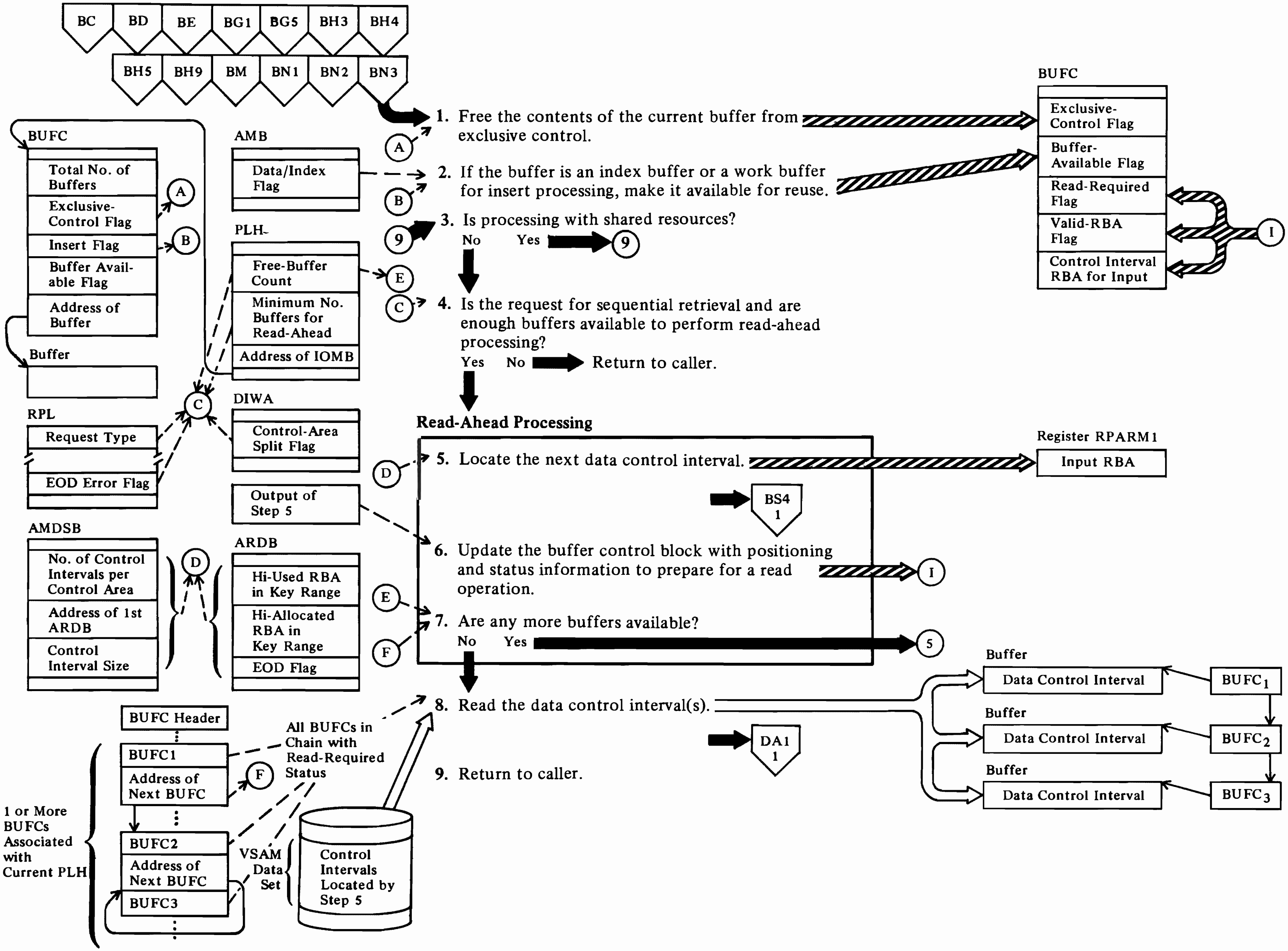
10 IDA019RU calls **IDA019R4**

11 IDA019RU

12 IDA019RU calls **IDA019R4**

13 IDA019RU

Diagram BS1. Buffer Management: Freeing the Buffer and Doing Read-Ahead Processing



1. Free the contents of the current buffer from exclusive control.
2. If the buffer is an index buffer or a work buffer for insert processing, make it available for reuse.
3. Is processing with shared resources?
   No
   Yes ⑨
4. Is the request for sequential retrieval and are enough buffers available to perform read-ahead processing?
   Yes
   No — Return to caller.

Read-Ahead Processing

5. Locate the next data control interval.
6. Update the buffer control block with positioning and status information to prepare for a read operation.
7. Are any more buffers available?
   No
   Yes ⑤
8. Read the data control interval(s).
9. Return to caller.

Notes for Diagram BS1

IDA019RZ is the Buffer Management interface module. For buffer management *without* shared resources, it calls IDA019R2; for buffer management *with* shared resources, it calls IDA019RY.

1 Removal of exclusive control allows other requests for the data control interval in the current buffer to be satisfied.

Processing without Shared Resources

IDA019RZ: IDAFREEB calls **IDA019R2**

If share-option 4 is specified, the buffer contents are forgotten.

If the data insert buffer or an index buffer is being freed, the test-and-set byte is cleared and exclusive control is released.

If the buffer being freed contains a segment of a spanned record, IDA019R2 releases exclusive control, but ensures that exclusive control is kept for the buffer that contains the first segment.

Processing with Shared Resources

IDA019RZ: IDAFREEB calls **IDA019RY**

If the buffer being freed has been modified, its modification mask is set to indicate the transaction ID of the modifier. If the buffer doesn't contain a segment of a spanned record held in exclusive control, exclusive control is released, the use count in the BUFC is decremented, and, if share-option 4 is specified, the buffer is marked empty.

2 **IDA019RZ**

4 **IDA019R2**

If the user is retrieving records sequentially or if a control area is being split, reading ahead speeds processing up by reading data into buffers before it is requested and while previous data is being processed.

5 **IDA019R2: RDAHEAD** calls **IDA019RW (IDAFRBA)**

During the loop represented by steps 5-7, if a buffer is encountered which has I/O outstanding, control is passed to step 8 before doing the locate processing at step 5.

If the return code from IDAFRBA indicates that the end of a control area has been reached, control is passed to step 8.

If IDAFRBA sets an end-of-data flag in the RPL, then an invalid-RBA flag is set in the current buffer, the error flags set by IDAFRBA in the RPL are cleared, and control is returned to step 8.

6 **IDA019R2: GETEXCL**

IDA019R2 initializes the RBA fields and read flags of the BUFC of each empty buffer if:

- The read threshold has been reached (the number of buffers required for read-ahead buffering have been freed),
- The request is for sequential retrieval,
- The request is for a control-area split, or
- The request is for spanned-record retrieval.

After the BUFC is updated, the control interval specified in the BUFC is placed under exclusive control. If the PLH indicates that the current request requires exclusive control (for example, if it is an update request), the exclusive-control flag in the current BUFC is set on.

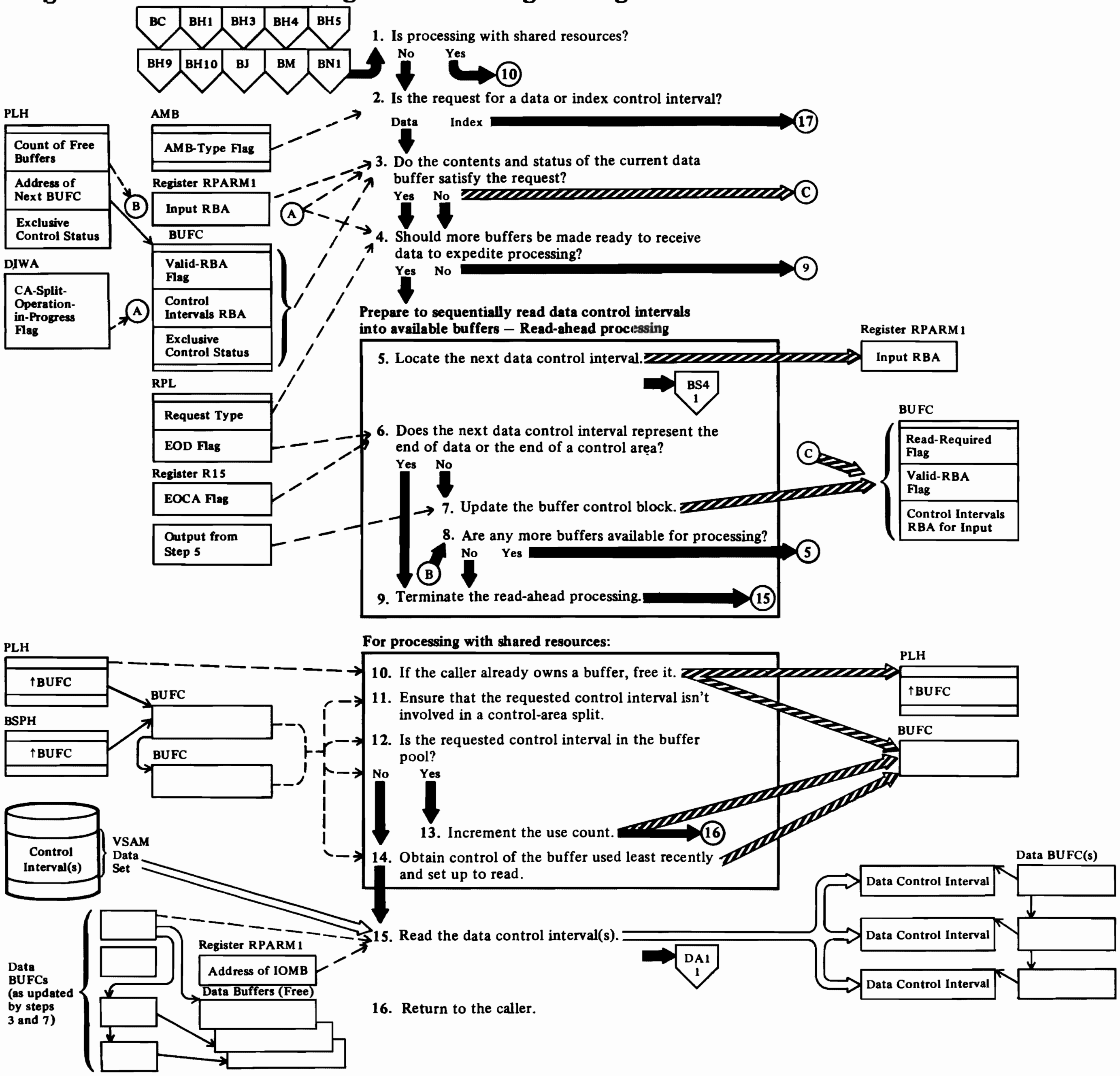
Exclusive control is immediately relinquished and error return codes are set if:
(a) the input RBA specified in the current BUFC falls within a control area which is being split, or if
(b) another BUFC with exclusive control specifies the same input or output RBA as the input RBA specified in the current BUFC.

8 **IDA019R2 calls IDAM19R3**

9 **IDA019R2**

Before returning to the caller, IDA019R2 advances the data-buffer address in the PLH to the next available data buffer.

Diagram BS2. Buffer Management: Reading a Designated Control Interval into a Buffer



1. Is processing with shared resources?

2. Is the request for a data or index control interval?

3. Do the contents and status of the current data buffer satisfy the request?

4. Should more buffers be made ready to receive data to expedite processing?

Prepare to sequentially read data control intervals into available buffers — Read-ahead processing

5. Locate the next data control interval.

6. Does the next data control interval represent the end of data or the end of a control area?

7. Update the buffer control block.

8. Are any more buffers available for processing?

9. Terminate the read-ahead processing.

For processing with shared resources:

10. If the caller already owns a buffer, free it.

11. Ensure that the requested control interval isn't involved in a control-area split.

12. Is the requested control interval in the buffer pool?

13. Increment the use count.

14. Obtain control of the buffer used least recently and set up to read.

15. Read the data control interval(s).

16. Return to the caller.

Notes for Diagram BS2

This diagram describes the Get-RBA function of Buffer Management.

3 **IDA019RZ: IDAGRB** calls **IDA019R2**

The BUFC for the current data buffer is examined to determine whether the requested control interval is already in the buffer and whether there is an exclusive-control conflict. If the requested control interval is already in the current data buffer and there isn't an exclusive-control conflict, IDA019R2 returns to the caller. If the requested control interval isn't in the buffer or if there is an exclusive-control conflict, the requested control interval must be read into the data buffer.

(There is an exclusive-control conflict if the user changes his request from simple retrieval (nonexclusive control) to retrieval-with-update (exclusive control). In this case, the BUFC would indicate nonexclusive control for the simple retrieval, and the placeholder would reflect exclusive control for the retrieval-with-update.)

When a read is required and exclusive control of the control interval is needed (that is, the user is doing a read-for-update), tests are performed to determine whether the control area containing the requested control interval is being split or whether a BUFC associated with another placeholder has the control interval under exclusive control. If either of these conditions exists, IDA019R2 sets an error code and returns to the caller. If neither exists, the BUFC for the current data buffer is given exclusive control of the control interval.

4 **IDA019R2: RDAHEAD**

For sequential retrieval or control-area splits, reading ahead (anticipatory buffering) speeds up processing.

5 **IDA019R2** calls **IDA019RW (IDAFRBA)**

10 **IDA019RZ** calls **IDA019RY**

No string can own more than one index, one data, and one insert buffer at a time. IDA019RY enforces this rule by freeing a buffer if the request would otherwise violate the rule.

13 If data is in the process of being read into the buffer, IDA019RY calls IDA019R5 (IDADRQ) to wait until I/O has completed. If the use count is incremented to more than one and the request is for exclusive control, a read-exclusive error is indicated.

14 **IDA019RY**

If a buffer not in use is found, it is written if its contents have been modified, and the read flag in its BUFC is set on. If no buffer not in use can be found, a logical error is indicated, and processing continues at step 16.

15 **IDA019R2 or IDA019RY** calls **IDAM19R3**

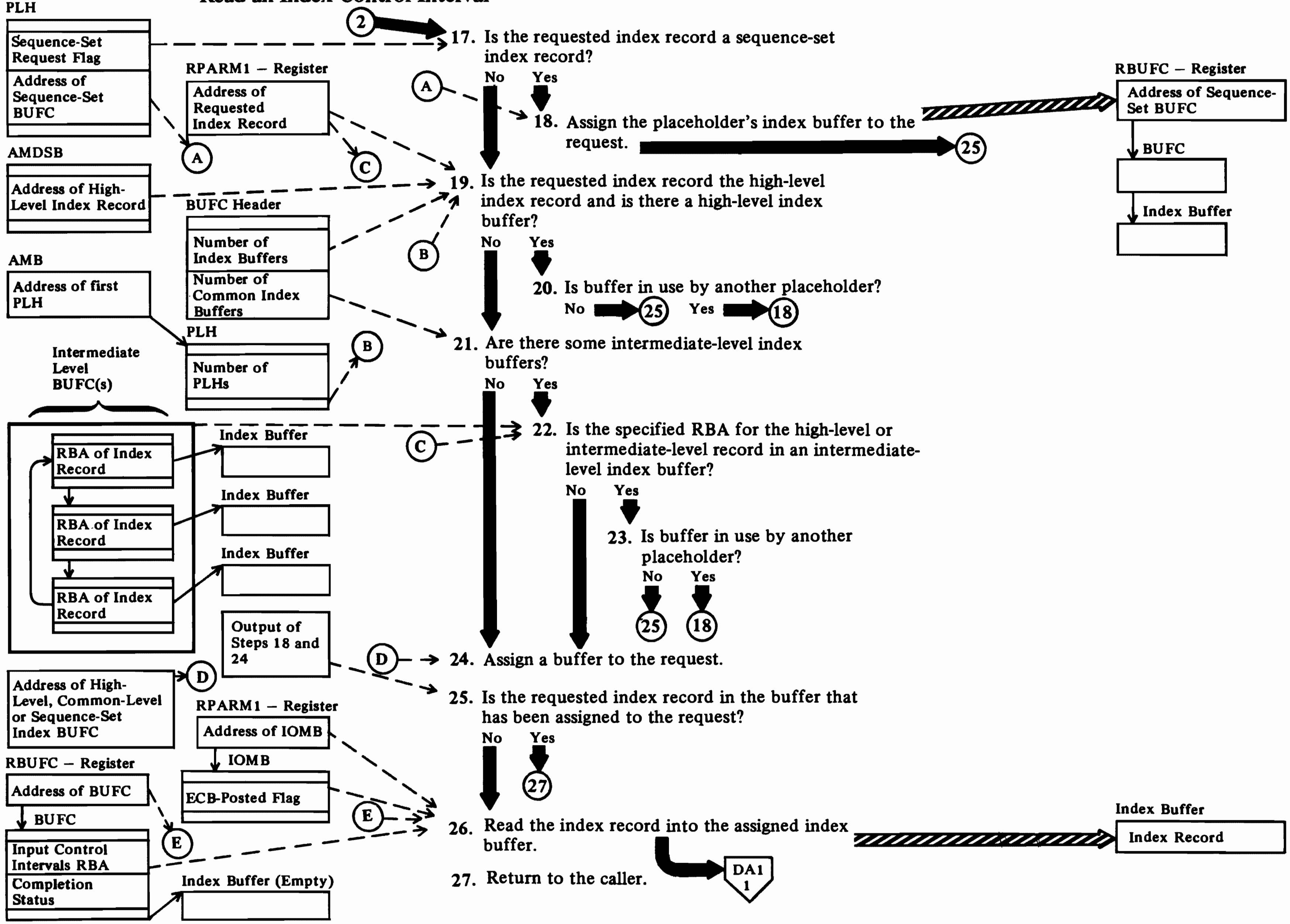
The specified control interval is read into the buffer associated with the current placeholder.

IDA0192 or IDA019RY calls **IDA019RZ (IDAWAIT)**

IDAWAIT waits until I/O is completed and tests whether a read error has occurred. If so, an RPL error code is returned to the caller.

Diagram BS3. Buffer Management: Reading a Designated Control Interval into a Buffer

Read an Index Control Interval



17. Is the requested index record a sequence-set index record?
No Yes

18. Assign the placeholder's index buffer to the request.

19. Is the requested index record the high-level index record and is there a high-level index buffer?
No Yes

20. Is buffer in use by another placeholder?
No 25 Yes 18

21. Are there some intermediate-level index buffers?
No Yes

22. Is the specified RBA for the high-level or intermediate-level record in an intermediate-level index buffer?
No Yes

23. Is buffer in use by another placeholder?
No Yes
25 18

24. Assign a buffer to the request.

25. Is the requested index record in the buffer that has been assigned to the request?
No Yes
27

26. Read the index record into the assigned index buffer.

27. Return to the caller.

Notes for Diagram BS3

18 IDA019RZ: IDAGRB calls **IDA019R2**

The index buffer pool has for each placeholder one index buffer that holds a sequence-set index record. When there are additional index buffers, the first one contains the highest-level index record, and the others contain intermediate-level index records.

19 IDA019R2

When a buffer is available, the highest-level index record is kept in it (for noncreate processing) for as long as the record continues to be the highest-level record. If the number of entries required to index the records in the next-lower level becomes too large for the highest-level record, a higher level is created. (The highest level of an index always has only one record.)

22 IDA019R2

Intermediate-level index records are kept in a buffer for as long as possible—that is, until a buffer is required to read in another index record.

24 IDA019R2

When the specified RBA in step 22 is not in a buffer, a free buffer is assigned to the request. If no buffers are free, one is made free. Buffers containing intermediate-level index records are candidates to be used. In some cases even a buffer containing the highest-level record or a sequence-set record is used.

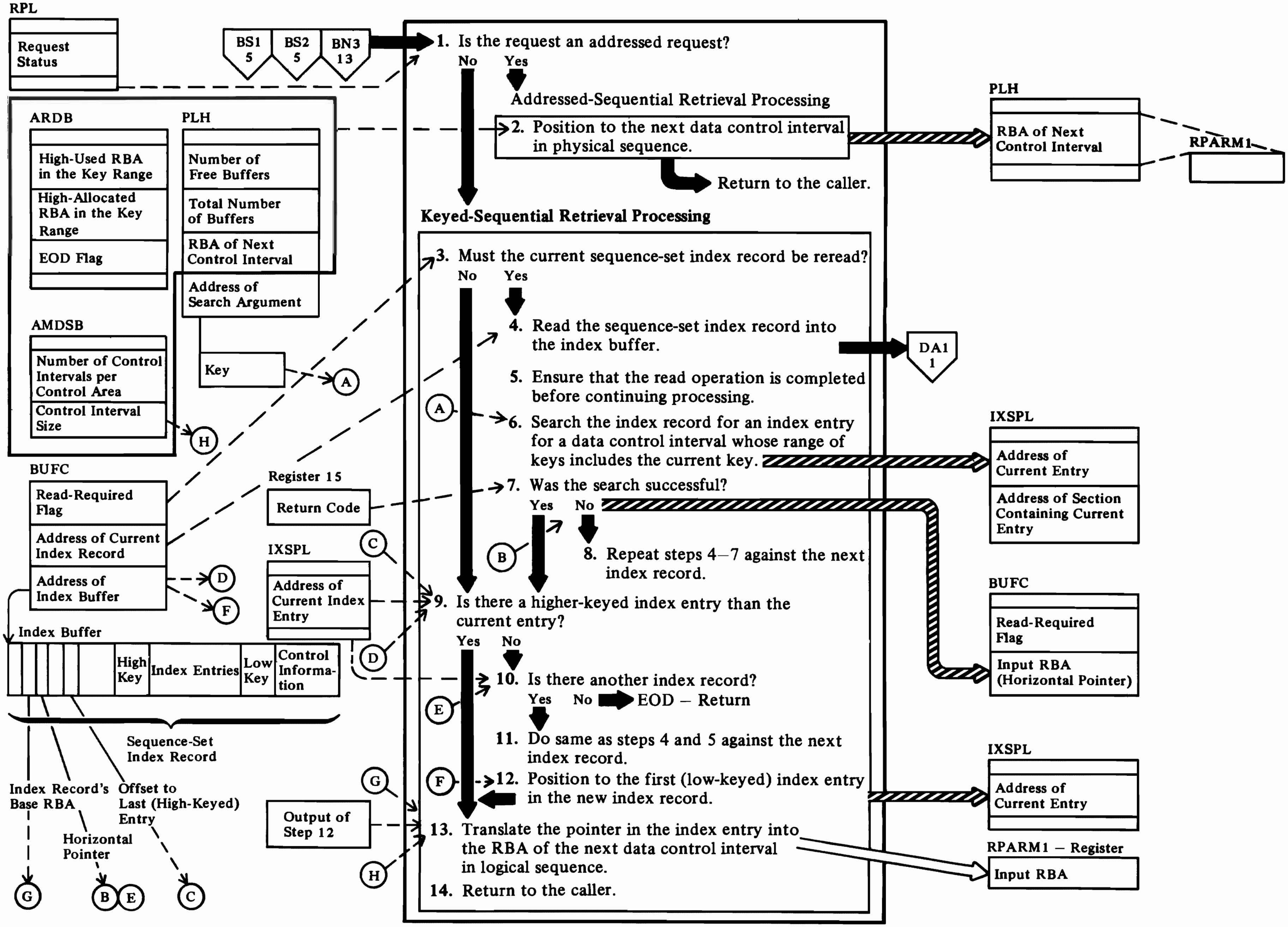
26 IDA019R2: READBFR calls **IDAM19R3**

The specified control interval is read into the buffer associated with the current placeholder.

IDA019R2 calls **IDA019RZ (IDAWAIT)**

IDAWAIT waits until I/O is completed and tests whether a read error has occurred. If so, an RPL error code is returned to the caller.

Diagram BS4. Buffer Management: Locating the Next Data Control Interval



1. Is the request an addressed request?
   No Yes

Addressed-Sequential Retrieval Processing

2. Position to the next data control interval in physical sequence.

Return to the caller.

Keyed-Sequential Retrieval Processing

3. Must the current sequence-set index record be reread?
   No Yes
4. Read the sequence-set index record into the index buffer.
5. Ensure that the read operation is completed before continuing processing.
6. Search the index record for an index entry for a data control interval whose range of keys includes the current key.
7. Was the search successful?
   Yes No
8. Repeat steps 4–7 against the next index record.
9. Is there a higher-keyed index entry than the current entry?
   Yes No
10. Is there another index record?
    Yes No EOD – Return
11. Do same as steps 4 and 5 against the next index record.
12. Position to the first (low-keyed) index entry in the new index record.
13. Translate the pointer in the index entry into the RBA of the next data control interval in logical sequence.
14. Return to the caller.

Notes for Diagram BS4

The Find-RBA function of Buffer Management, which is used in sequential retrieval operations, finds the RBA of the next control interval in collating sequence for keyed sequential requests or in entry sequence for addressed sequential requests.

1 **IDA019RZ** calls **IDA019RW (IDAFRBA)**

A key-sequenced data set may be processed by addressed sequential access. If control-interval or control-area splits have occurred, records retrieved by addressed access may not be in the same order as records retrieved by keyed access. That is, the entry sequence of records may not be the same as their key sequence.

2 **IDA019RW: IDAFRBA**

Before the current control-interval address in the placeholder is set to the RBA of the physically next control interval, IDAFRBA tests whether some buffers await I/O and whether the current control interval is the last one in a control area. If both conditions hold, IDAFRBA returns to the caller rather than set the RBA in the placeholder ahead. The reason for this is to avoid potential problems for sequential update processing and end-of-volume processing.

If the control intervals in a key range are exhausted, the address in the PLH is advanced to that of the first control interval in the next key range. If there are no additional key ranges for the data set, IDAFRBA returns an end-of-data error indicator to the caller.

3 **IDA019RW: IDAFRBA**

If a control-interval split has modified the sequence-set index record in the buffer, the record is reread before processing it any further.

4 **IDA019RW: IDAFRBA** calls **IDAM19R3**

5 **IDA019RW: IDAFRBA**

If the I/O Manager returns an error code, IDAFRBA returns to the caller.

IDA019RZ: IDAWAIT

If the specified I/O request has completed, processing continues at step 6. If a synchronous request hasn't completed, IDAWAIT issues a WAIT macro on the ECB for the index and processing continues at step 6 when the request completes. For an asynchronous request, IDAWAIT returns to the user's problem program, unless an I/O-completion interrupt occurs, in which case processing continues at step 6.

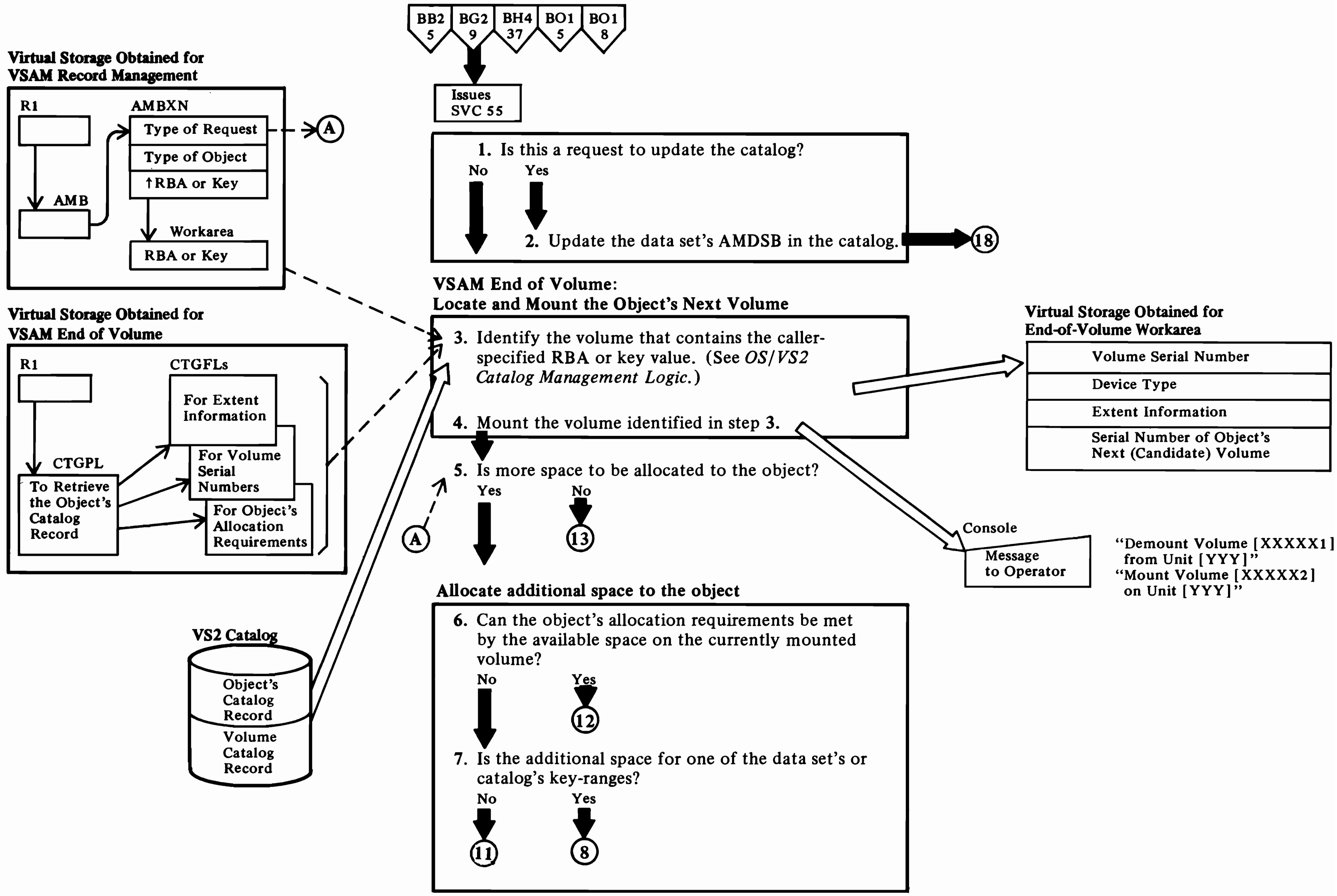
6 **IDA019RC**

The highest-keyed entries in each section are searched from right to left (that is, from lower to higher) until the entry whose key is greater than or equal to the search argument is found. Then the entries in that section are searched from right to left until the entry whose key is equal to or greater than the search argument is found.

10 **IDA019RW: IDAFRBA**

Before advancing to the sequence-set index record, IDAFRBA tests whether all processing related to the current control area is completed (see note for step 2). If some processing remains, IDAFRBA returns to the caller.

Diagram BT1. VSAM End of Volume: Obtaining the VSAM Object's Next Volume



BB2 5
BG2 9
BH4 37
BO1 5
BO1 8
Issues SVC 55
Virtual Storage Obtained for VSAM Record Management
R1
AMB
AMBXN
Type of Request
Type of Object
↑RBA or Key
Workarea
RBA or Key
A
Virtual Storage Obtained for VSAM End of Volume
R1
CTGPL
To Retrieve the Object's Catalog Record
CTGFLs
For Extent Information
For Volume Serial Numbers
For Object's Allocation Requirements
VS2 Catalog
Object's Catalog Record
Volume Catalog Record
A
18
13
12
11
8
Virtual Storage Obtained for End-of-Volume Workarea
Volume Serial Number
Device Type
Extent Information
Serial Number of Object's Next (Candidate) Volume
Console
Message to Operator
"Demount Volume [XXXXX1] from Unit [YYY]"
"Mount Volume [XXXXX2] on Unit [YYY]"

1. Is this a request to update the catalog?
No
Yes
2. Update the data set's AMDSB in the catalog.

VSAM End of Volume:
Locate and Mount the Object's Next Volume

3. Identify the volume that contains the caller-specified RBA or key value. (See *OS/VS2 Catalog Management Logic.*)
4. Mount the volume identified in step 3.
5. Is more space to be allocated to the object?
Yes
No

Allocate additional space to the object

6. Can the object's allocation requirements be met by the available space on the currently mounted volume?
No
Yes
7. Is the additional space for one of the data set's or catalog's key-ranges?
No
Yes

Notes for Diagram BT1

Diagram BT describes VSAM End-of-Volume processing. VSAM End of Volume is called by VS2 End of Volume when SVC 55 is issued by VSAM Record Management. VSAM End of Volume provides these services:

- When the GET routine detects that the requested record is not on any of the currently mounted volumes for the data set, a volume is demounted, if necessary, and the volume that contains the requested record is mounted.
- When a PUT request cannot be completed because there is no more space in the object, additional space is allocated to the object. The amount is based on the object's space allocation requirement. If enough space is available to satisfy the object's space allocation requirement, the space is allocated from the free space in:
  - First, the VSAM data space containing the object.
  - Next, the volume containing the object. If an object's key range is assigned more space, space is allocated from the volume containing the key range if the object has not been assigned an overflow volume. Otherwise, (for key range only) space is allocated from another volume that has been assigned to the key range's object as an overflow volume.
  - Finally, another VSAM volume that has been assigned to the object as a candidate volume.

1 **IDA0557A**

The request is either to handle an end-of-volume condition or to update information in the catalog.

2 **IDA0557A: CATUPD** (which calls **IDA0192C**)

The AMDSB contains statistics for the data set.

3 **IDA0557A: VOLLOC** (calls **ARDBSCH**)

The volume information sets of fields (in the object's catalog record) contain the volume serial number of each volume (used or candidate) assigned to the object. The volume information sets of fields also contain the low and high key values of each key range, and the low and high RBA values of each extent in the object.

If the end-of-volume request is for more space on the currently mounted volume, the volume's serial number is in the end-of-data ARDB.

4 **IDA0557A: VOLLOC** (calls **VOLMNT**)

The VSAM Volume Mount and Verify routine (IDA0192V) confirms that the specified volume is mounted. If no device is available for the volume, the VSAM Volume Mount and Verify routine requests that the operator demount a volume not in use. If all devices contain volumes currently in use, the VSAM Volume Mount and Verify routine sets the volume-not-mounted return code and returns to the caller.

5 **IDA0557A: ALLOCSPC**

If the AMBXN's allocate-space request option indicator is on, End of Volume gets more space for the object.

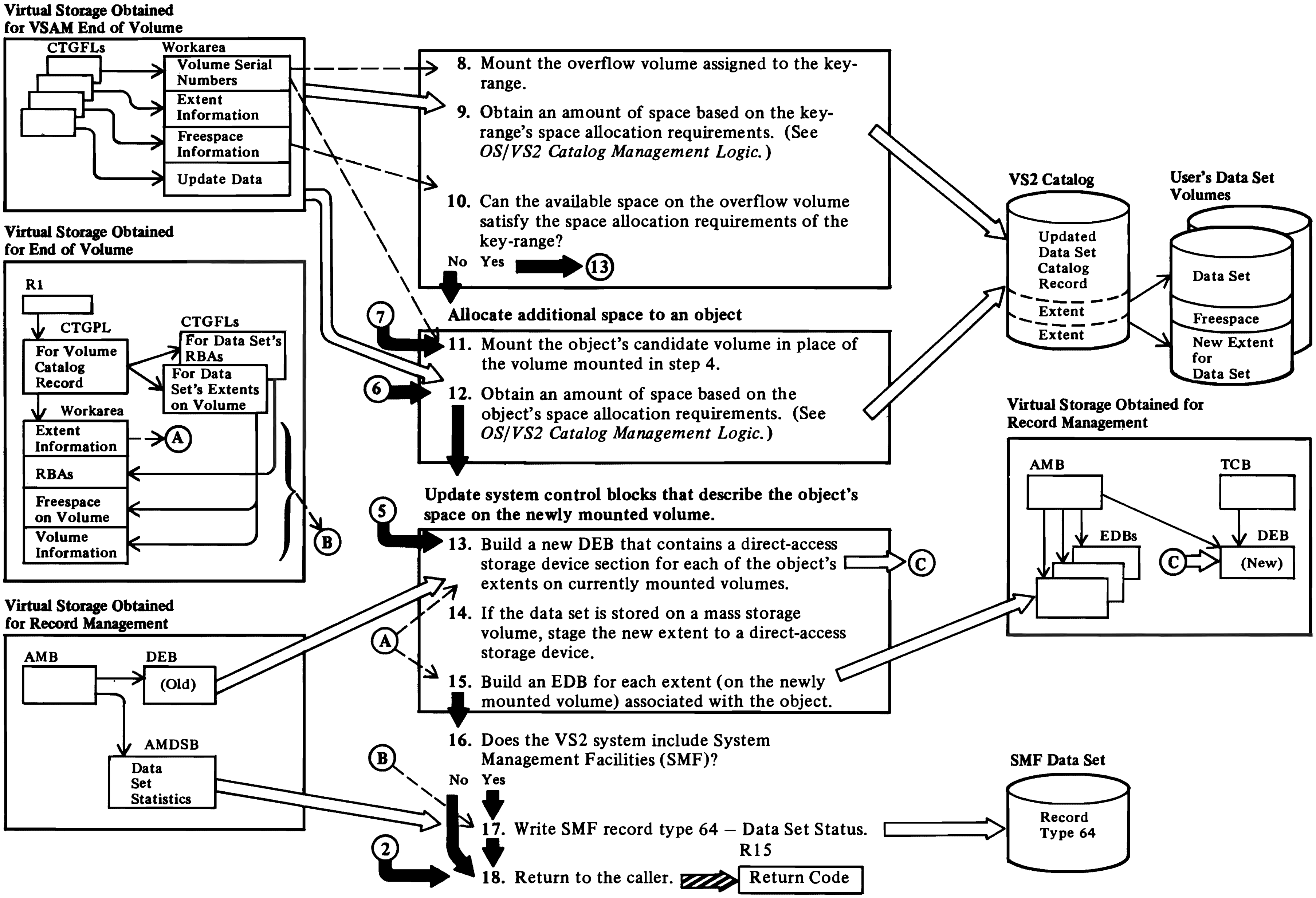
See "Data Areas" for details about the AMB and the AMBXN.

6 **IDA0557A: ALLOCSPC** (calls **CATALC**)

The volume catalog record defines a VSAM direct access volume in terms of the objects it contains, the VSAM data spaces it contains, and the available (free) space in each of it's data spaces.

See *OS/VS2 Catalog Management Logic* for details about the volume catalog record.

Diagram BT2. VSAM End of Volume: Allocating Additional Space to a VSAM Object



8. Mount the overflow volume assigned to the key-range.

9. Obtain an amount of space based on the key-range's space allocation requirements. (See *OS/VS2 Catalog Management Logic.*)

10. Can the available space on the overflow volume satisfy the space allocation requirements of the key-range?

No Yes → 13

Allocate additional space to an object

11. Mount the object's candidate volume in place of the volume mounted in step 4.

12. Obtain an amount of space based on the object's space allocation requirements. (See *OS/VS2 Catalog Management Logic.*)

Update system control blocks that describe the object's space on the newly mounted volume.

13. Build a new DEB that contains a direct-access storage device section for each of the object's extents on currently mounted volumes.

14. If the data set is stored on a mass storage volume, stage the new extent to a direct-access storage device.

15. Build an EDB for each extent (on the newly mounted volume) associated with the object.

16. Does the VS2 system include System Management Facilities (SMF)?

No Yes

17. Write SMF record type 64 – Data Set Status.

18. Return to the caller.

Notes for Diagram BT2

8 IDA0557A: VOLSW (calls **CATLOCNC** and **VOLMNT)**

If the key range's object has an overflow volume assigned to it, additional space for the key range is allocated from the overflow volume. If no overflow volume is assigned to the object, steps 8 through 10 are bypassed and the space is allocated from the object's candidate volume.

9 IDA0557A: VOLSW (calls **CATALC** and **CATUPDVO)**

The object's catalog record describes its space allocation requirements.

10 IDA0557A: VOLSW (calls **CATLOCNC)**

If there is not enough available space on the overflow volume to satisfy the allocation requirements of the key range, space is allocated from the object's candidate volume.

11 IDA0557A: ALLOCSPC (calls **VOLSW)**

If the volumes are full, and no other volume (candidate) is assigned to the object, End of Volume sets the space-not-allocated return code and returns to the caller.

See *OS/VS2 Access Method Services* for a description of how candidate volumes are assigned to VSAM objects.

12 IDA0557A: CATALC

The object's catalog record describes its space allocation requirements.

See *OS/VS2 Catalog Management Logic* for details about the catalog record details, and the volume information set-of-fields.

13 IDA0557A: CTLBLK (calls **DSCTLBLK)**

See "Data Areas" for details about the ACB and the EDB. See *OS/VS2 Data Areas* for details about the DEB.

End of Volume builds a new DEB and EDB that replace the existing DEB and EDB. The new DEB and EDB contain extent information that describe:

- Each of the object's extents (on currently mounted volumes) that was not affected by the End-of-Volume process.
- Each extent that defines the object's newly obtained space (if any).
- None of the object's extents on volumes that were demounted.

14 IDA0192D

This module issues an ACQUIRE to the Mass Storage System.

15 IDA0557A: DSCTLCLK (calls **CATLOCXT** and **CATLOCRB)**

See *OS/VS2 Catalog Management Logic* for details about the data set catalog record, and the volume information set of-fields.

See "Data Areas" for details about the EDB.

17 IDA0557A: SMFUPD (calls **CATLOCDS)**

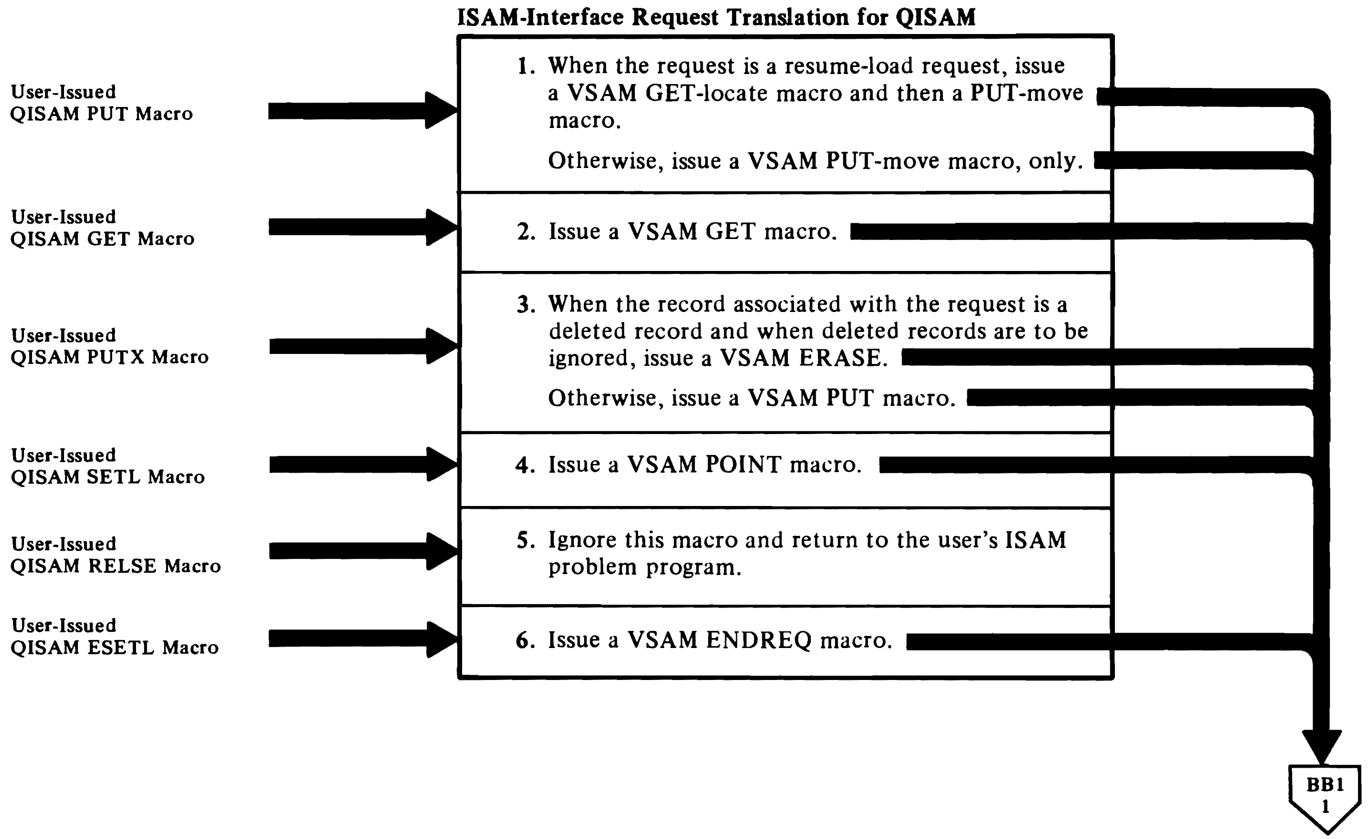
See *OS/VS2 System Programming Library: System Management Facilities (SMF)* for a description of SMF record type 64.

18 IDA0557A: TERM, PROBDET

See "Diagnostic Aids" for details about the VSAM End-of-Volume return codes.

If an error is detected, End of Volume attempts to determine the type of error and builds a message describing the error.

Diagram BU1. ISAM-Interface: Processing a VSAM Data Set with an ISAM-User's Program



ISAM-Interface Request Translation for QISAM

User-Issued QISAM PUT Macro

1. When the request is a resume-load request, issue a VSAM GET-locate macro and then a PUT-move macro.

   Otherwise, issue a VSAM PUT-move macro, only.

User-Issued QISAM GET Macro

2. Issue a VSAM GET macro.

User-Issued QISAM PUTX Macro

3. When the record associated with the request is a deleted record and when deleted records are to be ignored, issue a VSAM ERASE.

   Otherwise, issue a VSAM PUT macro.

User-Issued QISAM SETL Macro

4. Issue a VSAM POINT macro.

User-Issued QISAM RELSE Macro

5. Ignore this macro and return to the user's ISAM problem program.

User-Issued QISAM ESETL Macro

6. Issue a VSAM ENDREQ macro.

Notes for Diagram BU1

1 IDAIIPM1: QISAM PUT Processing

To handle an ISAM PUT-Locate request, VSAM uses the ISAM-Interface buffer to contain records to be written. For ISAM PUT-move requests, the user supplies the buffer. (Note: In both cases, VSAM treats the buffer as the user's work area, and transfers records to its own output buffers before writing them.)

For ISAM resume-load requests, a GET-locate is issued to VSAM to search the previously created data set for a key greater than or equal to the key of the first record to be written by resume-load. If the VSAM search is unsuccessful, it is assumed that the previous last key and the new key are in correct sequence, and load processing continues.

A successful search indicates that the new key is less than a key already in the data set (a logical error); and control is passed to the user's ISAM SYNAD routine if it exists. Otherwise, an ABEND is issued.

2 IDAIIPM2: QISAM GET Processing

If the ISAM GET request is preceded by a SETL request (used to determine whether the located record was a deleted record), the retrieved record is moved from the ISAM-Interface buffer to the user's buffer and a VSAM GET macro is not issued.

When the ISAM GET request is in locate mode or specifies data-only, the ISAM-Interface buffer is used for the record; otherwise, the user's buffer is used. (Note: Data-only implies that the key resides at the beginning of the data record; the relative key position of the record is 0.) A VSAM GET macro is issued. If the request specifies move-mode and data-only options, the data (minus the key) is moved into the user's buffer. When a deleted record is retrieved, and such records are to be ignored, successive GET macros are issued until a normal record is retrieved.

3 IDAIIPM2: QISAM PUTX Processing

If the record to be written had only the data portion of the record retrieved (see note 2), the data is moved from the user's buffer to the ISAM-Interface buffer to rejoin its key before it is written; otherwise, the complete record already resides in the appropriate buffer.

The record is then examined to determine whether it is marked as a deleted record. Deleted records are ignored, if requested, by issuing a VSAM ERASE macro to eliminate the original record from the data set. A VSAM PUT macro is issued for those records that are to be written.

4 IDAIIPM2: QISAM SETL Processing

The validity of the request is tested, and if two SETL requests have been issued without an intervening GET, PUTX, or ESETL macro, an invalid SETL macro has been issued or an invalid generic key has been used. An invalid request error code is set and control is passed to the ISAM-Interface SYNAD routine (see note 11).

If the request is valid, the address of the key to be located is placed in the RPL, and a VSAM POINT macro is issued.

If the data set contains deleted records and if the request is directed at a specific record's key, a VSAM GET macro is issued to retrieve the record. If the record is a deleted record, a no-record-found indicator is set in the DCB and control is passed to the ISAM-Interface SYNAD routine (see note 11).

5 IDAIIPM2: QISAM RELSE Processing

This request is ignored by the ISAM-Interface routine, and control is immediately returned to the user. The release function is not required by ISAM-Interface or VSAM because each QISAM request handled by ISAM-Interface uses only a single data record for request processing.

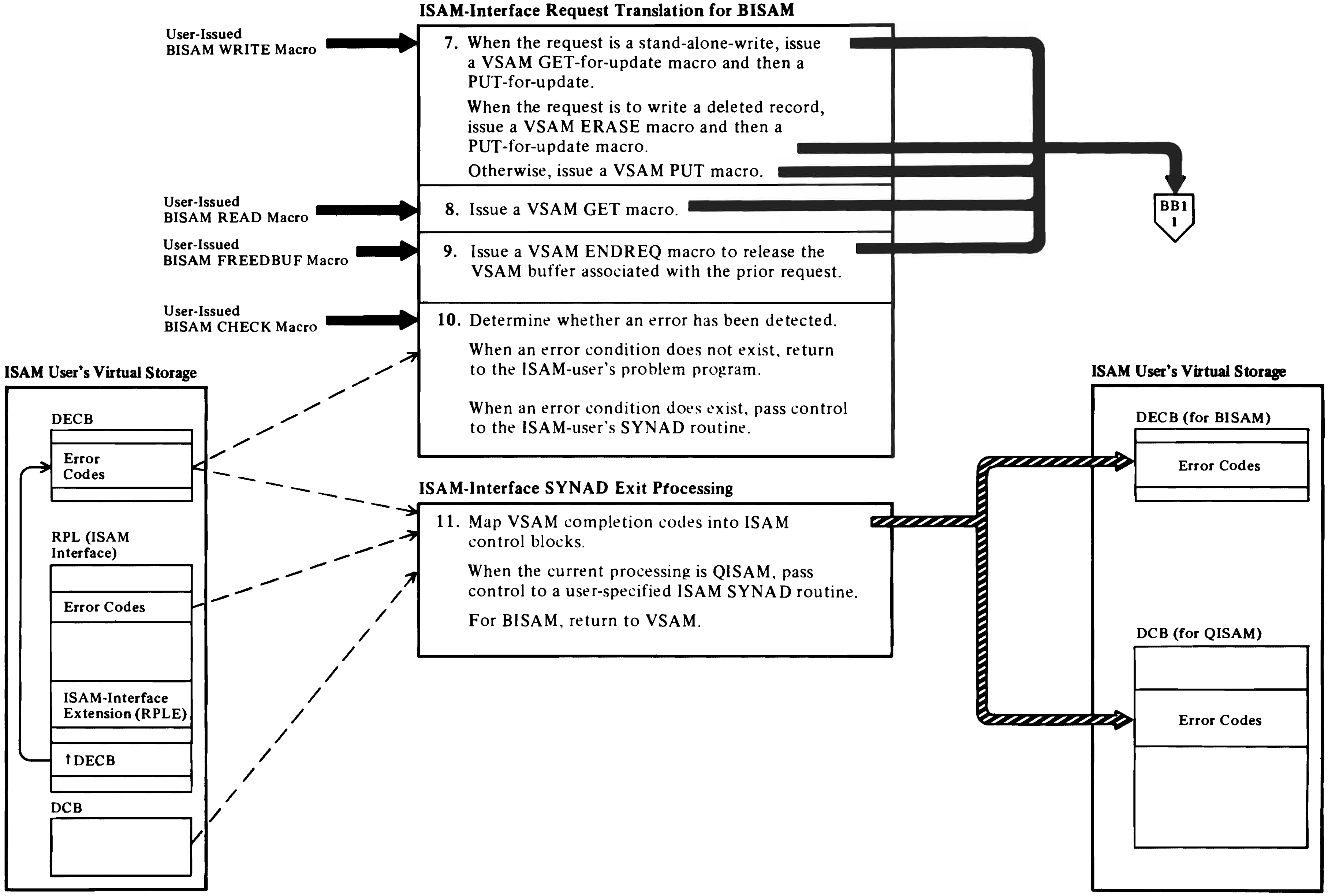
6 IDAIIPM2: QISAM ESETL Processing

A VSAM ENDREQ macro is issued to release any VSAM resources. ISAM Interface resets the scan-mode indicator in the IICB, which enables another SETL request to be issued, and returns control to the user.

IDAIIPM2: QISAM EODAD Processing

This routine receives control when VSAM reaches an end-of-data condition. The ISAM EODAD routine is given control if one has been specified; otherwise, an ABEND is issued.

Diagram BU2. ISAM-Interface: Processing a VSAM Data Set with an ISAM User's Program



ISAM-Interface Request Translation for BISAM

7. When the request is a stand-alone-write, issue a VSAM GET-for-update macro and then a PUT-for-update.

   When the request is to write a deleted record, issue a VSAM ERASE macro and then a PUT-for-update macro.

   Otherwise, issue a VSAM PUT macro.

8. Issue a VSAM GET macro.

9. Issue a VSAM ENDREQ macro to release the VSAM buffer associated with the prior request.

10. Determine whether an error has been detected.

    When an error condition does not exist, return to the ISAM-user's problem program.

    When an error condition does exist, pass control to the ISAM-user's SYNAD routine.

ISAM-Interface SYNAD Exit Processing

11. Map VSAM completion codes into ISAM control blocks.

    When the current processing is QISAM, pass control to a user-specified ISAM SYNAD routine.

    For BISAM, return to VSAM.

Notes for Diagram BU2

7 IDAIIPM3: BISAM WRITE Processing

The ISAM-Interface RPLs are searched for one which is associated with the current request's DECB. If an RPL is not found, an available RPL is assigned to the request and initialized. If an RPL is not available, an invalid request is indicated in the DECB and a return is made to the user's problem program.

If the write request is an ISAM stand-alone-write for update, VSAM GET-for-update and PUT-for-update macros are issued to satisfy the request.

For a write request to overlay an existing data record with a deleted record, the VSAM PUT macro is issued to satisfy the request unless the option to ignore the deleted record is specified. In this case, the ERASE macro is issued. (Note: Deleted records have a X'FF' in their first byte.)

For a write-key-new request, a VSAM PUT is issued. If VSAM returns an error code indicating that the record to be written is a duplicate of an existing data record, ISAM-Interface issues a VSAM GET to retrieve the existing data record to determine whether it is a deleted record. If the record is a deleted record, a VSAM PUT-for-update request is issued to replace it with the new record.

When VSAM returns control, the ISAM-Interface RPL is released (disconnected from the DECB), a VSAM ENDREQ macro is issued to free the VSAM resources, and the request is posted complete.

8 IDAIIPM3: BISAM READ Processing

The RPLs are searched for one which is associated with the current request's DECB. If an RPL is not found, an available RPL is assigned to the request and initialized. If an RPL is not available, a return is made to the user's problem program.

After establishing the buffer to be used (that is, an ISAM buffer or an ISAM-Interface buffer) and adjusting the record pointer to include a record descriptor word (RDW) for variable-length records, a VSAM GET macro is issued.

When VSAM returns control, the ISAM-Interface RPL is released (disconnected from the DECB) and a VSAM ENDREQ macro is issued to free the VSAM resources, unless the ISAM request was a successful read-for-update.

9 IDAIIFBF: BISAM FREEDBUF Processing

This routine issues a SYNCH SVC to get into problem program state and then searches the ISAM-Interface request-string for an RPL associated with the current ISAM DECB. When found, a VSAM ENDREQ macro is issued to free the resources held by the RPL. The RPL is then disconnected from the DECB. If an associated RPL is not found, a return is made to the user's problem program.

If the RPL is found and processing of it is complete, a VSAM ENDREQ macro is issued to free the VSAM resources, and then the ISAM-Interface RPL is released (disconnected from the DECB) for reuse by another request.

10 IDAIIPM3: BISAM CHECK Processing

The ISAM-Interface Check routine tests for an error code in the DECB (see note 3). If an error is not detected, a return is made to the user's problem program. If an error is detected, the Check routine passes control to the user's ISAM SYNAD routine if it exists; otherwise, an ABEND is issued.

11 IDAIISM1: ISAM-Interface SYNAD Processing

The ISAM-Interface SYNAD routine is entered by a VSAM processing routine when an error condition is detected.

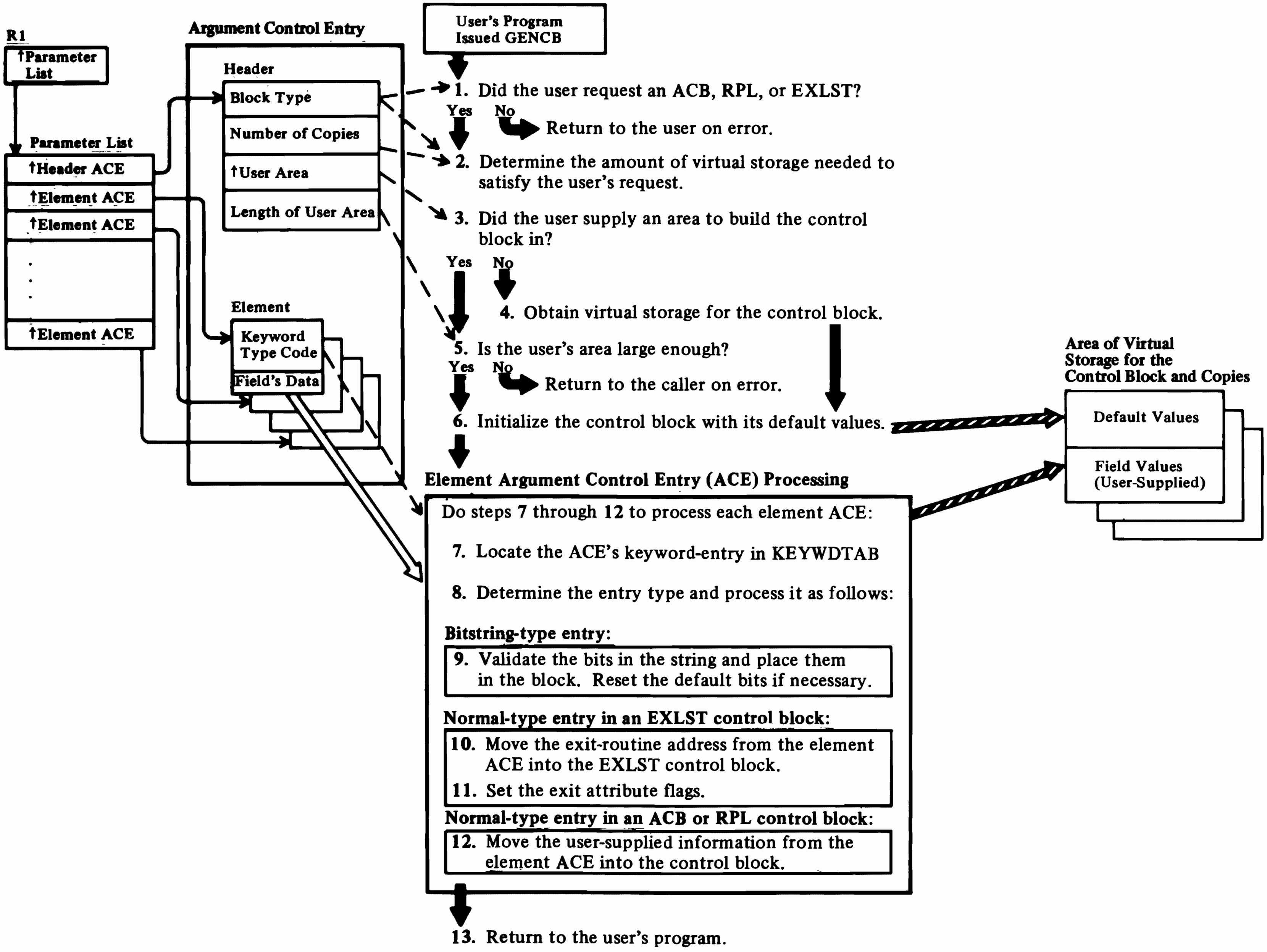
For QISAM processing, the VSAM error codes in the RPL are copied into the DCB, and for BISAM processing, the error codes are copied into the DECB.

For QISAM processing, control is passed to the user's ISAM SYNAD routine if it exists. If it does not exist, an ABEND is issued.

For BISAM processing, a return is made to VSAM, which returns to the ISAM-Interface BISAM processing routine and then to the user's problem program. An ensuing ISAM CHECK macro causes the user's ISAM SYNAD routine to receive control if it exists (see note 10).

The ISAM-Interface SYNAD routine also builds the SYNADAF message.

Diagram CA. GENCB: Build a New Control Block



1. Did the user request an ACB, RPL, or EXLST?
   No: Return to the user on error.
2. Determine the amount of virtual storage needed to satisfy the user's request.
3. Did the user supply an area to build the control block in?
4. Obtain virtual storage for the control block.
5. Is the user's area large enough?
   No: Return to the caller on error.
6. Initialize the control block with its default values.

Element Argument Control Entry (ACE) Processing

Do steps 7 through 12 to process each element ACE:

7. Locate the ACE's keyword-entry in KEYWDTAB
8. Determine the entry type and process it as follows:

Bitstring-type entry:

9. Validate the bits in the string and place them in the block. Reset the default bits if necessary.

Normal-type entry in an EXLST control block:

10. Move the exit-routine address from the element ACE into the EXLST control block.
11. Set the exit attribute flags.

Normal-type entry in an ACB or RPL control block:

12. Move the user-supplied information from the element ACE into the control block.

13. Return to the user's program.

Notes for Diagram CA

1 IDA019C1

The GENCB macro is issued to create an ACB, RPL, or EXLST dynamically.

2-5

The ACB and RPL are fixed-length control blocks, but the EXLST is variable-length. The Control Block Manipulation routine calculates the amount of space needed for the control block and any copies the user requested. The Control Block Manipulation routine issues a GETMAIN macro to obtain the required virtual storage for any block for which a user area is not provided.

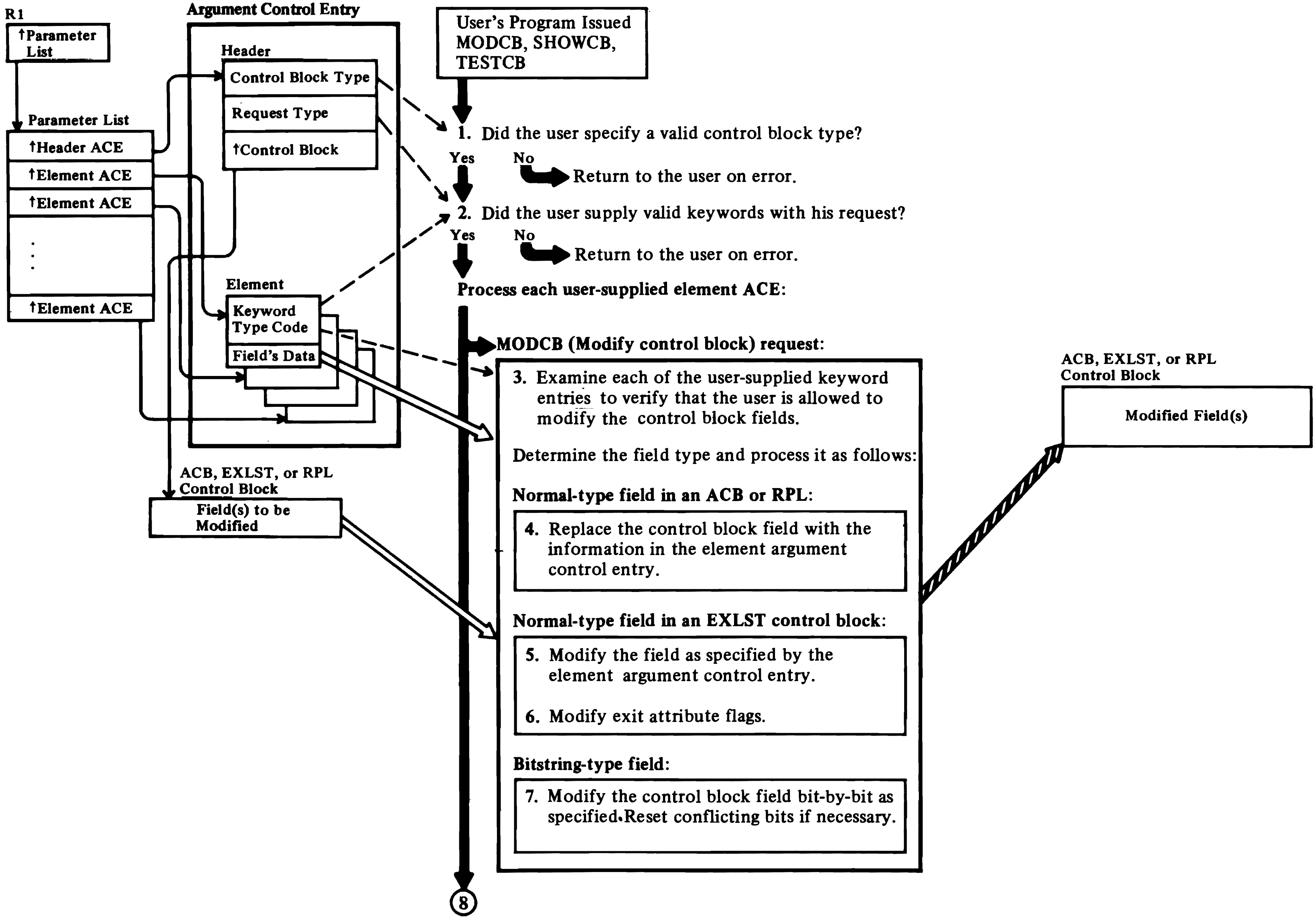
6

The block is initialized to its default values. Information is subsequently added to the block as specified by the element argument control entries (ACEs)

11

The exit attribute flags indicate that an exit address is present, active, inactive, or set during link-edit.

Diagram CB1. MODCB, SHOWCB, TESTCB: Modify, Display, or Test a Control Block



1. Did the user specify a valid control block type?

   No: Return to the user on error.

2. Did the user supply valid keywords with his request?

   No: Return to the user on error.

Process each user-supplied element ACE:

MODCB (Modify control block) request:

3. Examine each of the user-supplied keyword entries to verify that the user is allowed to modify the control block fields.

Determine the field type and process it as follows:

Normal-type field in an ACB or RPL:

4. Replace the control block field with the information in the element argument control entry.

Normal-type field in an EXLST control block:

5. Modify the field as specified by the element argument control entry.

6. Modify exit attribute flags.

Bitstring-type field:

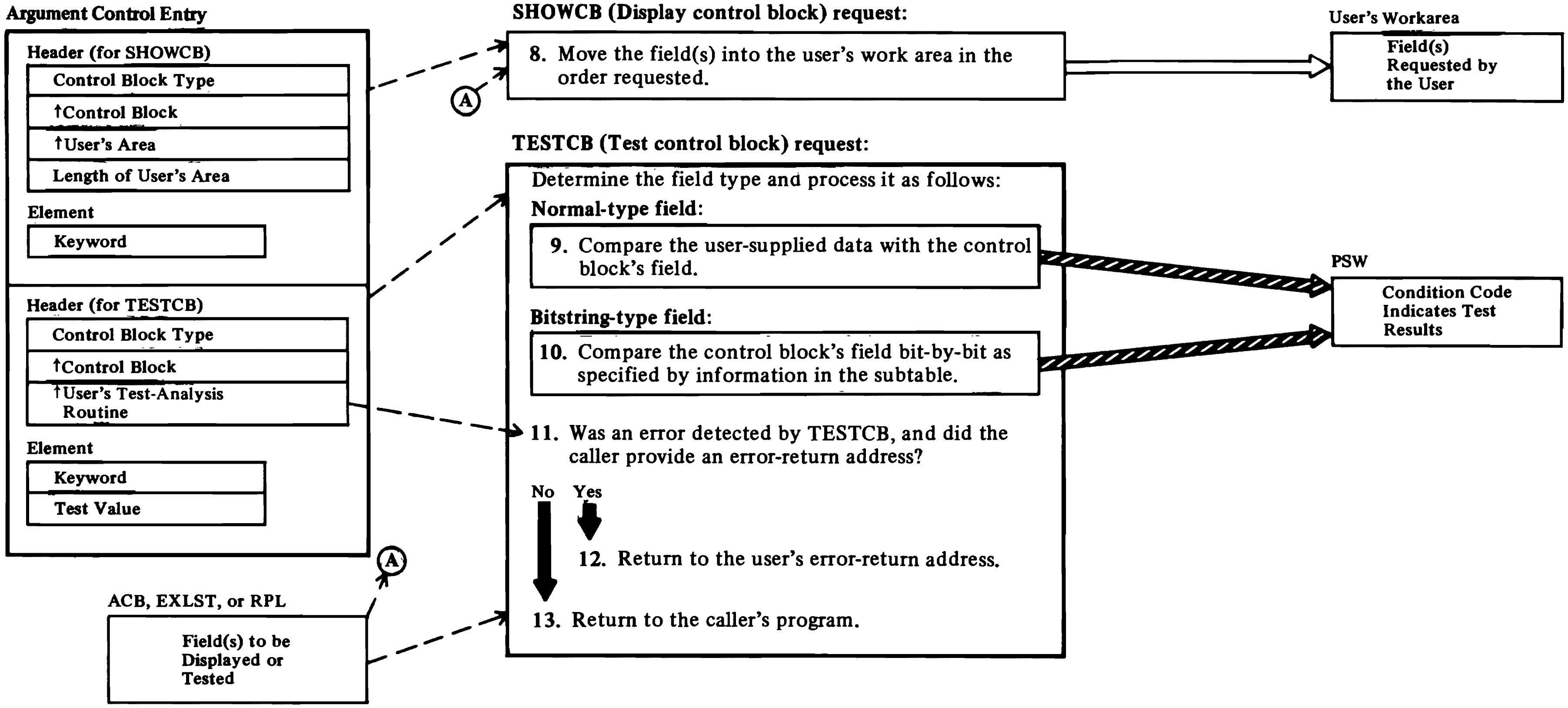
7. Modify the control block field bit-by-bit as specified.Reset conflicting bits if necessary.

Notes for Diagram CB1

1 **IDA019C1**

The MODCB, SHOWCB, and TESTCB macros are issued to modify, display, and test, respectively, the ACB, RPL, and EXLST control blocks in the user's address space.

Diagram CB2. MODCB, SHOWCB, TESTCB: Modify, Display, or Test a Control Block



SHOWCB (Display control block) request:

8. Move the field(s) into the user's work area in the order requested.

TESTCB (Test control block) request:

Determine the field type and process it as follows:

Normal-type field:

9. Compare the user-supplied data with the control block's field.

Bitstring-type field:

10. Compare the control block's field bit-by-bit as specified by information in the subtable.

11. Was an error detected by TESTCB, and did the caller provide an error-return address?

12. Return to the user's error-return address.

13. Return to the caller's program.

Notes for Diagram CB2

4-13

The field attribute table entry contains the length, offset from the beginning of the block, and characteristics of the field in the control block.

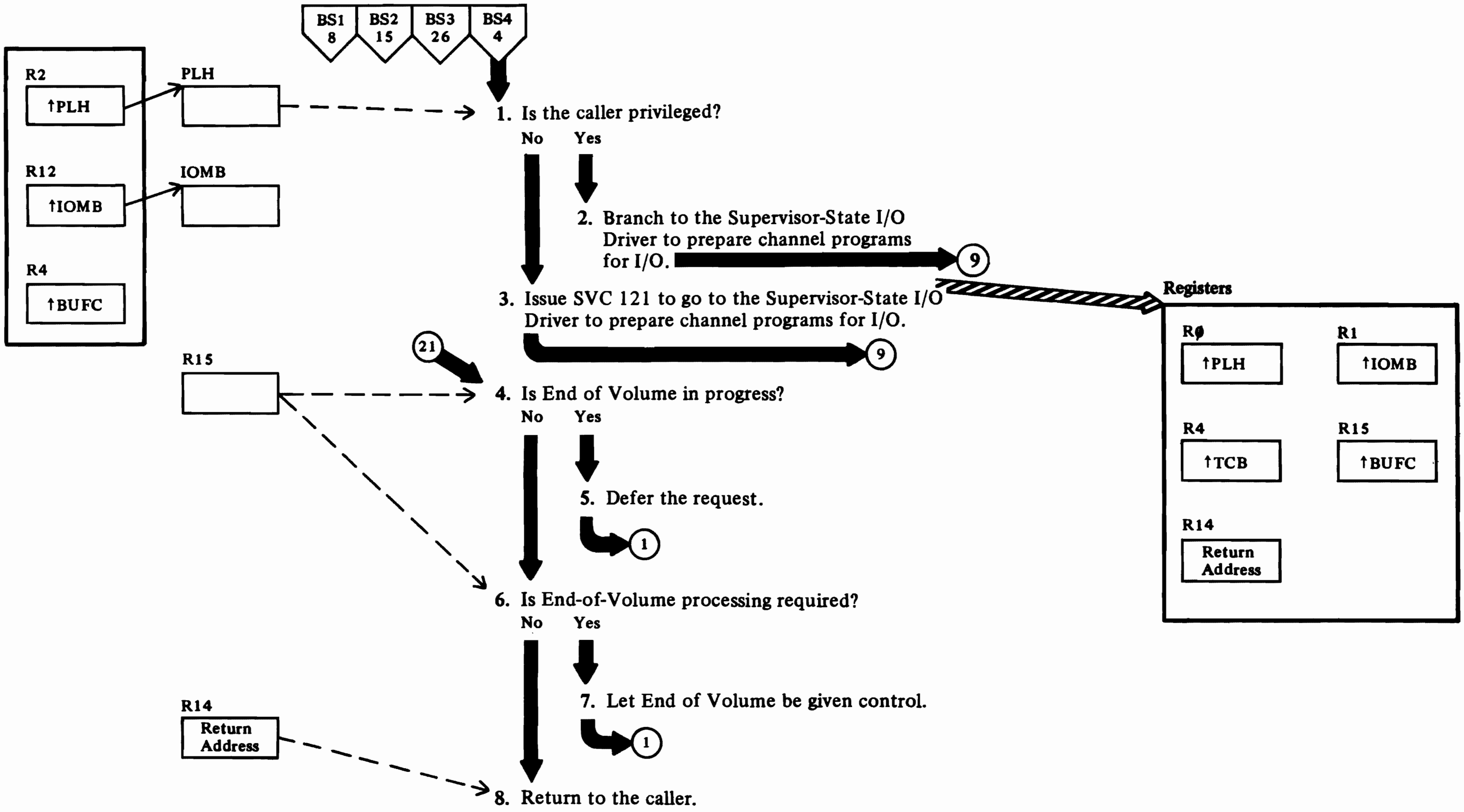
Three types of entries are identified in the field attribute table: bitstring, normal, and entries that require a special subroutine to process them.

If the entry is a bitstring type, the field attribute table points to a series of bit entries in the bitstring table that are used to modify the control block (MODCB), or are compared to a value supplied by the user (TESTCB).

If the entry is a normal type, the element argument control entry is moved into the block (MODCB), a character string or field is moved into the user's area (SHOWCB), or the user's argument field is compared with the appropriate fields in the block (TESTCB).

Diagram DA1. I/O Management

Ensure That I/O Can Be Scheduled


BS1 8
BS2 15
BS3 26
BS4 4
R2
↑PLH
R12
↑IOMB
R4
↑BUFC
PLH
IOMB
R15
R14
Return Address
1. Is the caller privileged?
No
Yes
2. Branch to the Supervisor-State I/O Driver to prepare channel programs for I/O.
9
3. Issue SVC 121 to go to the Supervisor-State I/O Driver to prepare channel programs for I/O.
9
21
4. Is End of Volume in progress?
No
Yes
5. Defer the request.
1
6. Is End-of-Volume processing required?
No
Yes
7. Let End of Volume be given control.
1
8. Return to the caller.
Registers
R∅
↑PLH
R1
↑IOMB
R4
↑TCB
R15
↑BUFC
R14
Return Address

Notes for Diagram DA1

VSAM Buffer Management (IDA019R2 and IDA019RY) calls the I/O Manager. It enters module IDAM19R3, the Problem-State I/O Driver (PIOD), at IDA019R3.

When Buffer Management calls the I/O Manager in privileged state, IDAM19R3 obtains a local lock for storage protection. In order to free the lock in case IDAM19R3 fails, IDAM19R3 has a functional recovery routine (with entry point PIODFRR) that gets control from VS2 Recovery Termination Management when an error occurs while IDAM19R3 is processing. This routine also issues an SDUMP macro to record information in SYS1.DUMP.

2 IDAM19R3: SCHDSIOD

If Buffer Management has called the I/O Manager in privileged state (storage protect key less than 8), IDAM19R3 sets the key to 0, obtains a local lock, and branches directly to IGC121. If an error condition is indicated upon return from attempting to obtain a local lock, IDAM19R3 issues ABEND 377(179) with reason code 4(04). (See Diagram DA2.)

3 IDAM19R3: SCHDSIOD

If Buffer Management has called the I/O Manager in nonprivileged state, IDAM19R3 sets up registers as required and issues SVC 121 to give control to IGC121. (See Diagram DA2.)

4 IDAM19R3: EOVTEST

When End of Volume is in progress, it requires (for control-block integrity) that all IOMBs for the data and index AMBs be inactive. Thus I/O cannot be scheduled for this IOMB. Besides, if the current request from Buffer Management requires End-of-Volume processing, the I/O Manager must wait until End of Volume is finished anyway.

5 IDAM19R3: EOVTEST calls **IDA019R5 (IDADRQ)**

IDADRQ waits upon an indication that End of Volume has completed.

7 IDAM19R3: EOVTEST calls **IDA019R5 (IDAEOVIF)**

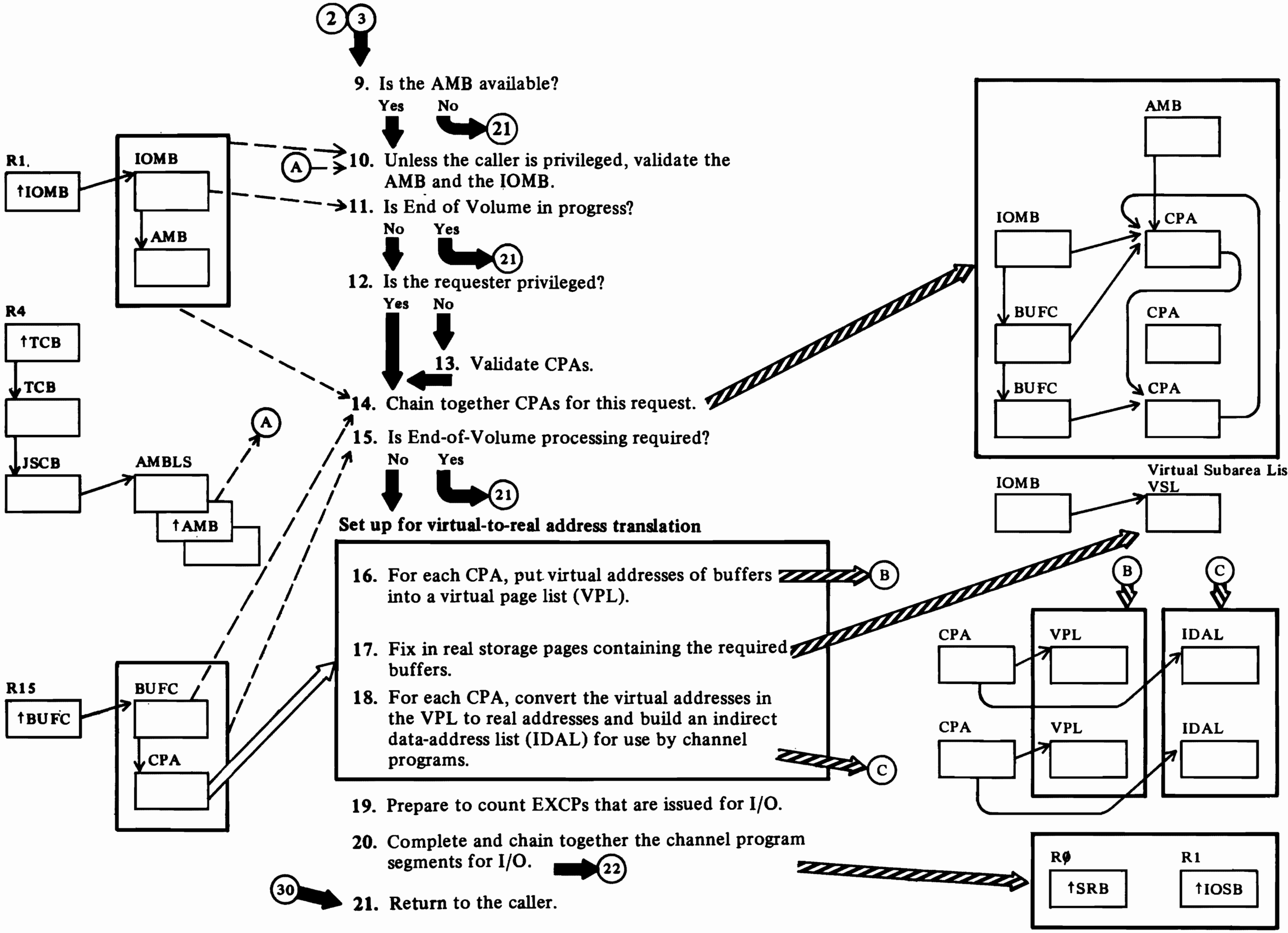
As far as IGC121 is concerned, End of Volume is in progress as soon as IDAM19R3 calls IDAEOVIF. However, IDAEOVIF may not be able to start End of Volume immediately, since all IOMBs of the data and index AMBs must become inactive. (See note for step 4 and Diagram DA2.)

8 IDAM19R3: EXIT

Buffer Management reacquires control after the I/O has been scheduled. See Diagram DA4 for a description of how the I/O Manager gives control back to the caller after the I/O is completed.

Diagram DA2. I/O Management

Prepare for Channel Programs to Be Completed for I/O



9. Is the AMB available?
   Yes / No → 21

10. Unless the caller is privileged, validate the AMB and the IOMB.

11. Is End of Volume in progress?
    No / Yes → 21

12. Is the requester privileged?
    Yes / No

13. Validate CPAs.

14. Chain together CPAs for this request.

15. Is End-of-Volume processing required?
    No / Yes → 21

Set up for virtual-to-real address translation

16. For each CPA, put virtual addresses of buffers into a virtual page list (VPL).

17. Fix in real storage pages containing the required buffers.

18. For each CPA, convert the virtual addresses in the VPL to real addresses and build an indirect data-address list (IDAL) for use by channel programs.

19. Prepare to count EXCPs that are issued for I/O.

20. Complete and chain together the channel program segments for I/O. → 22

21. Return to the caller.

Notes for Diagram DA2

IGC121, also known as the Supervisor-State I/O Driver (SIOD), makes up this phase of the I/O Manager.

ICG121 has a functional recovery routine (with entry point IDA121F1) that gets control from VS2 Recovery Termination Management when an error occurs while IGC121 is processing. This routine frees pages fixed in real storage by IGC121, releases the local lock that IGC121 obtains for storage protection, and issues an SDUMP to record information in SYS1.DUMP.

9 IGC121

10 IGC121: RBKEY, VALIDCBS, CHKIOMB

IGC121 verifies that the AMB address in the IOMB matches a data or index AMB address in the AMBL chain identified by the JSCB.

It verifies that the IOMB address given as input matches an IOMB address in the chain identified by the AMB.

For an invalid AMB or IOMB, IGC121 issues ABEND 377(179) with reason code 8(08).

11 IGC121: TESTEOV

IGC121 may already (in a previous pass) have requested that IDAM19R3 call IDA019R5 (IDAEOVIF). If End of Volume has begun and isn't finished, IDAM19R3 defers the request.

14 IGC121: CHKCPA, CHAINCPA

The IOMB points to the first of a chain of BUFCs to be used for this request. Each BUFC points to a CPA on a chain of CPAs identified by the AMB.

Each CPA has two chain fields. One chains together each CPA associated with the AMB. The other is used by IGC121 to chain together just those CPAs associated with the request from Buffer Management.

For a nonprivileged requester, IGC121 first checks whether each CPA identified by the BUFCs is on the chain identified by the AMB. If not, IGC121 issues ABEND 377(179) with reason code 12(0C).

IGC121 then points the IOMB to the first CPA for the request and chains together the CPAs for the request. That is, CPAs on the AMB chain that aren't referenced by a BUFC for the request aren't chained with those for the request.

15 IGC121: WRITE, READ (which call **CONVERT)**

For writes and reads in a request, IGC121 locates EDBs (extent definition blocks) and verifies that the data-set extents described by the existing EDBs cover the RBAs contained in the request. If an RBA for a BUFC is not covered by any EDB, VSAM End of Volume must create one. If IGC121 has previously asked IDAM19R3 to call IDA019R5 (IDAEOVIF), IGC121 indicates that the BUFC has been processed and that an RBA is invalid. The channel program segment for this BUFC isn't chained with the others (by IDA121A2) for the VS2 I/O Supervisor. (See Diagram DA3.)

16 IGC121: BLDVPL

The VPL (virtual page list) for a request may already have been built before IGC121 was entered. A VPL is a list of virtual addresses. It has one entry for each physical record of 2048 bytes or less to be contained in the buffer (a buffer contains the contents of a control interval); it has two entries for each 4096-byte physical record.

17 IGC121: BLDVPL, PAGEIN1 (which branch to the VS2 PGFIX Routine)

The VSL (virtual subarea list) is a parameter list to communicate to the VS2 PGFIX Routine the virtual addresses of buffers to be fixed in real storage. (It is also called a PFL—page fix list.) It contains a beginning and ending virtual address for each buffer.

To bring each page into real storage for fixing, IGC121 executes a TM (test under mask) instruction to reference each part of storage addressed by the entries in the VPLs.

IGC121 passes to the PGFIX Routine the address of the VSL. Upon return from PGFIX, if all pages haven't been brought into real storage for fixing, IGC121 executes the TM instruction again and continues (without branching to PGFIX again, as PGFIX saves requests for page fixing).

If the PGFIX Routine indicates that an error occurred, IGC121 issues ABEND 377(179) with reason code 16(10).

After pages are fixed, IGC121 indicates in the IOMB that they were fixed by IGC121 for VSAM Record Management. The end appendages use this indication to free the pages after the request is completed. (See Diagram DA4.)

Buffers may already be fixed and the PGFIX Routine not needed.

18 IGC121: BLDIDAL calls **PAGEOUT (which branches to the VS2 PGFREE Routine)**

Each virtual address in the VPLs is translated to a real address with the LRA instruction. If the real address cannot be found, IGC121 branches to the VS2 PGFREE Routine to free pages it has caused to be fixed in real storage and issues ABEND 377(179) with reason code 20(14).

If no BUFC has associated with it a valid write or read channel program segment, IGC121 returns to IDAM19R3.

19 IGC121: STOREUCB calls **VS2 IEASMFEX**

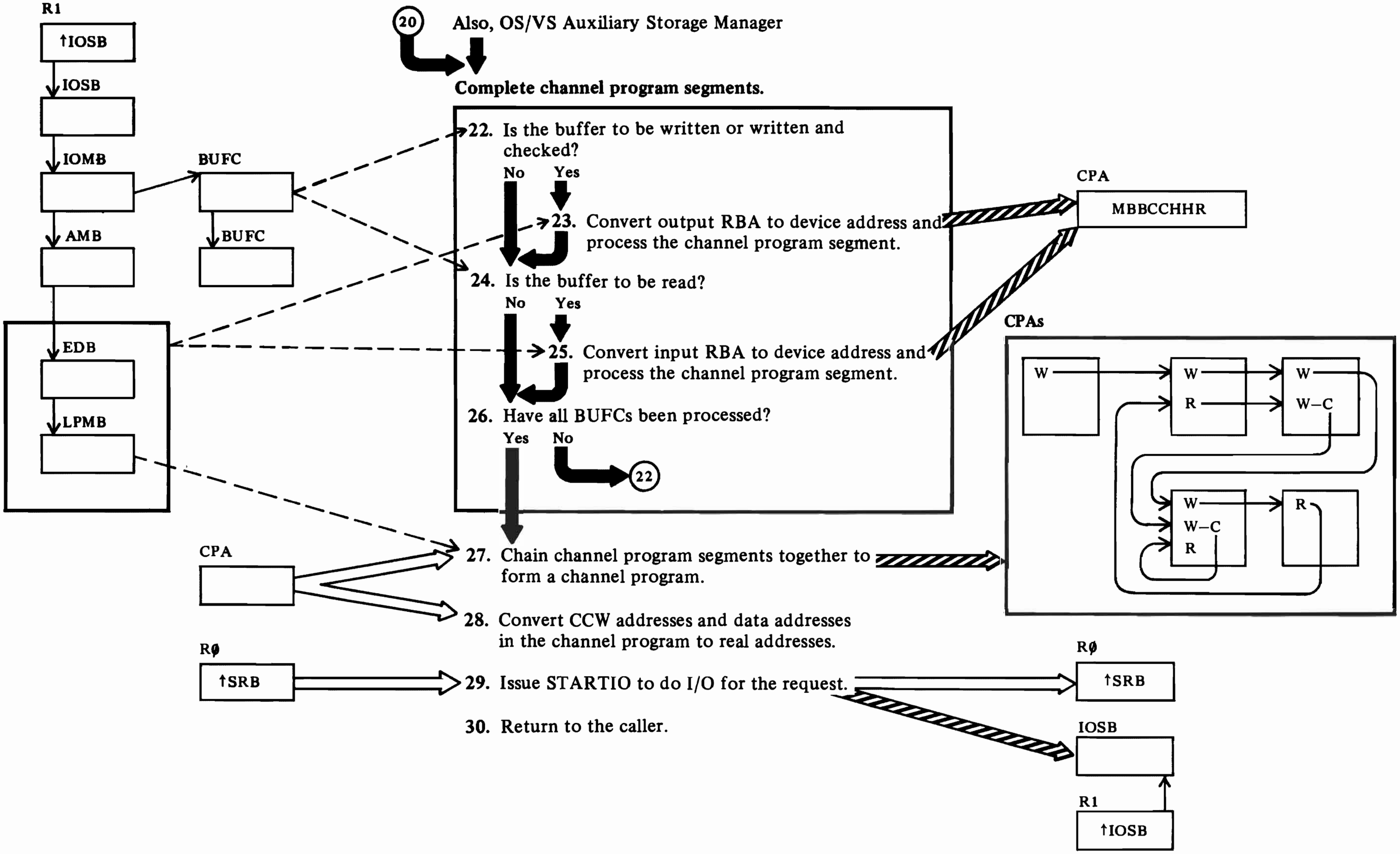
ICG121 calls System Management Facilities to count EXCPs when the VS2 I/O Supervisor does I/O for the request.

20 IGC121: CALLABP calls **IDA121A2**

See Diagram DA3.

Diagram DA3. I/O Management

Complete a Channel Program for I/O



20 Also, OS/VS Auxiliary Storage Manager

Complete channel program segments.

22. Is the buffer to be written or written and checked?
 No Yes

23. Convert output RBA to device address and process the channel program segment.

24. Is the buffer to be read?
 No Yes

25. Convert input RBA to device address and process the channel program segment.

26. Have all BUFCs been processed?
 Yes No → 22

27. Chain channel program segments together to form a channel program.

28. Convert CCW addresses and data addresses in the channel program to real addresses.

29. Issue STARTIO to do I/O for the request.

30. Return to the caller.

Notes for Diagram DA3

IDA121A2, the Actual Block Processor (ABP), makes up this phase of the I/O Manager. It converts input and output RBAs to device addresses, completes the channel program segments, and chains them together to form a channel program that the VS2 I/O Supervisor schedules to do the I/O operations that IDA121A2 is passing on from the requester.

A request to gain access to the contents of a control interval has been translated to a channel program that gains access to physical records (that is, to "actual blocks").

The request originated in either VSAM Record Management or VS2 Auxiliary Storage Management. IDA121A2 gets control from VSAM by way of IDAM19R3 and IGC121. From the Auxiliary Storage Manager it gets control directly.

IDA121A2 has a functional recovery routine (with entry point IDA121F2) that gets control from VS2 Recovery Termination Management when an error occurs while IDA121A2 is processing. This routine frees pages fixed in real storage by IGC121, releases the local lock that IDA121A2 obtains for storage protection, and issues an SDUMP to record information in SYS1.DUMP.

22 IDA121A2: CPGEN

Three I/O operations may be associated with a BUFC: write, write-check, read. They may be combined in five ways: write, write with write-check, read, write and read, write with write-check and read.

23 IDA121A2: WRITE, DOWRITE, FMTWRITE, CCWGEN (which call CONVERT)

In order to connect a processing program with the actual data recorded on direct-access storage devices, extent definitions blocks (EDBs) must be created to describe storage areas. As areas are filled, VSAM End of Volume must create a new EDB to describe the next area to receive data.

If the appropriate EDB for RBA conversion is not available, IDA121A2 indicates the fact in the BUFC and bypasses processing the channel program segment.

IDA121A2 completes a write-format or a write-update channel program segment and a write-check segment, as indicated in the BUFC.

24 IDA121A2: READT

See note to step 22.

25 IDA121A2: READT, DOREAD (which call **CONVERT**)

See note to step 23. VSAM End of Volume must create an EDB to describe an area on a new volume when requests reference data on volumes whose extents were not previously described.

If the appropriate EDB for RBA conversion isn't available, IDA121A2 indicates the fact in the BUFC and bypasses processing the channel program segment.

IDA121A2 completes a read channel program segment.

26 IDA121A2

If the VS2 Auxiliary Storage Manager is the requester, IDA121A2 builds an indirect data-address list (IDAL) for each CPA. Each IDAL has only a beginning and an ending buffer address, since, for Auxiliary Storage Management, buffer size is equal to 2048. (Compare with notes for steps 16-18.)

In the last BUFC, the pointer to the next BUFC points to the first BUFC processed. If no channel program segments have been built (no appropriate EDBs for conversion), IDA121A2 returns to the caller.

27 IDA121A2: CHAIN, PASS1, PASS2, PASS3 (which call **SETSECTR) (which branches to VS2 IECSCR1)**

IDA121A2 first locates the first CPA that has a valid channel program segment and prepares it for rotational position sensing if the CPA indicates the request is for a device with RPS.

If no CPA with a valid channel program segment is found, IDA121A2 returns to the caller.

Each CPA may have 1, 2, or 3 channel program segments. IDA121A2 chains segments together by passing through the CPAs first to connect the first segments' in each CPA, then to connect the second segments (if any), and finally the third segments (if any).

Segments whose CPAs have been invalidated (such as by no appropriate EDB's having been found) are omitted from the chain.

28 IDA121A2: STARTIOX calls **PAGEOUT**

All addresses in the channel program are converted as required: data addresses, chain addresses, TIC addresses. Addresses converted may be virtual addresses or incorrect real addresses. Conversion is done by adding positive or negative offsets between virtual and real addresses or between incorrect and correct real addresses.

The channel program segments in the channel program may be in different pages of real storage. To chain the CCW of one segment to the CCW of the next segment, IDA121A2 converts addresses using the LRA instruction (load real address).

IDA121A2 puts information in the IOSB so the VS2 I/O Supervisor can control the channel program execution.

If any conversion of virtual to real addresses fails (LRA instruction), IDA121A2 issues ABEND 377(179) with reason code 24(18).

29 IDA121A2 calls VS2 Basic I/O Supervisor (via STARTIO)

Before issuing the STARTIO for a request from a caller not in supervisor state, IDA12A2 changes the caller's key to 0 (zero) (IOSB IOSCKY field), to be able to store information in the CPA.

The VS2 I/O Supervisor schedules the I/O and returns immediately to IDA121A2. After the I/O is finished, an end appendage gets control. (See Diagram DA4.)

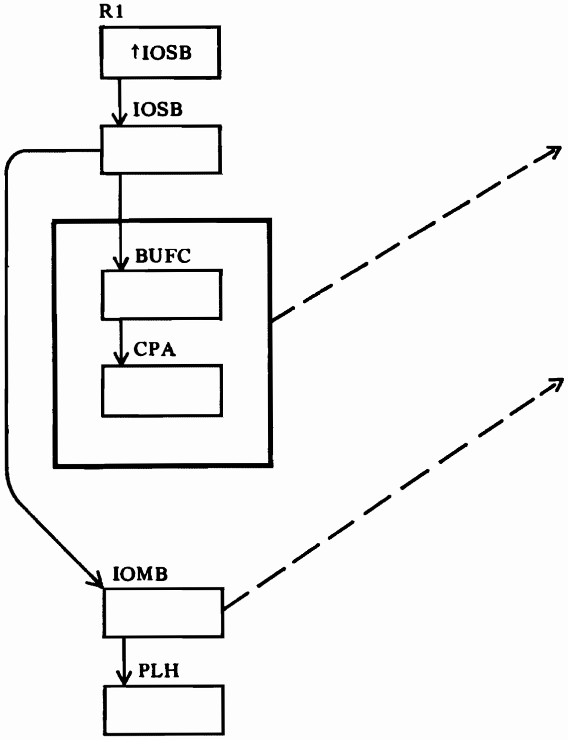
IDA121A2 sets up the address of its routine ABPTERM to get control from the I/O Supervisor in case an error occurs before an end appendage has got control. ABPTERM turns on the completion bit and error bits in the first BUFC and returns to the caller.

30 IDA121A2

At this return I/O is not finished (it may not even be started yet). The caller gets control back to manage the use of the time until the I/O is completed.

Diagram DA4. I/O Management

Processing after I/O Is Completed



OS/VS I/O Supervisor

31. Did I/O complete successfully?
    Yes No → 43

32. Indicate request completion in BUFCs.

33. Is a purge in progress and is the request synchronous?
    No Yes

34. Decrement the count of outstanding I/O operations.

35. Did OS/VS Auxiliary Storage Management make the request?
    No Yes

36. Indicate that it is to receive control. → 42

37. Is the request asynchronous?
    No Yes → 40

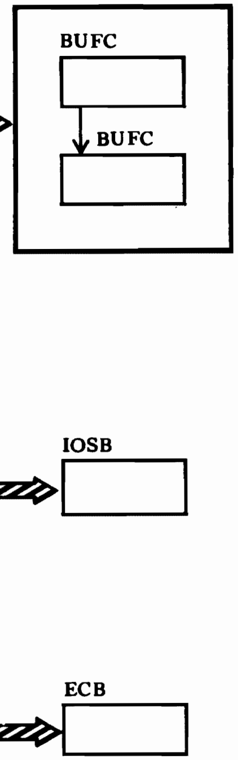
38. Indicate I/O completion in an ECB.

39. Is End of Volume waiting to process?
    Yes No → 41

40. Schedule IDA121A5 to run. (See steps 47–51.)

41. Free pages fixed in real storage by IGC121.

42. Return to OS/VS I/O Supervisor.



Notes for Diagram DA4

The VSAM Normal End Appendage is IDA121A3. It has a functional recovery routine (with entry point IDA121F3) that gets control from VS2 Recovery Termination Management when an error occurs while IDA121A3 is processing. This routine essentially duplicates IDA121A3's processing. It determines what IDA121A3 wasn't able to do and carries out the task, in order to complete the Normal End function if possible.

This functional recovery routine also has its own functional recovery routine (F3FRR) that gets control from VS2 Recovery Termination Management when an error occurs while IDA121F3 is processing. It frees pages fixed in real storage by ICG121 and issues an SDUMP to record in SYS1.DUMP information in the system queue area, the private service area, the GTF trace tables, the common service area, and pages in the private area of the address space, including the local system queue area and the scheduler work area.

31 VS2 IECVPST

The end appendages get control from the VS2 I/O Supervisor Post Status Routine.

32 IDA121A3

BUFCs in error (or BUFCs whose CPAs were in error) have already had bits set by IGC121 or IDA121A2.

34 IDA121A3 branches to VS2 IECVQCNT

VS2 IECVQCNT

IECVQCNT decrements the count of requests yet to complete. An end appendage can have control during a purge only for a quiesce operation, in which case requests are allowed to complete in usual fashion, but without new requests' being allowed. When the count has been decremented to zero, the purge can be completed.

35 IDA121A3

If there is an "address to which to return" in the IOMB, then the VS2 Auxiliary Storage Manager called the I/O Manager (entering at IDA121A2). This address is put into the IOSB.

38 IDA121A3 branches to the VS2 POST Routine

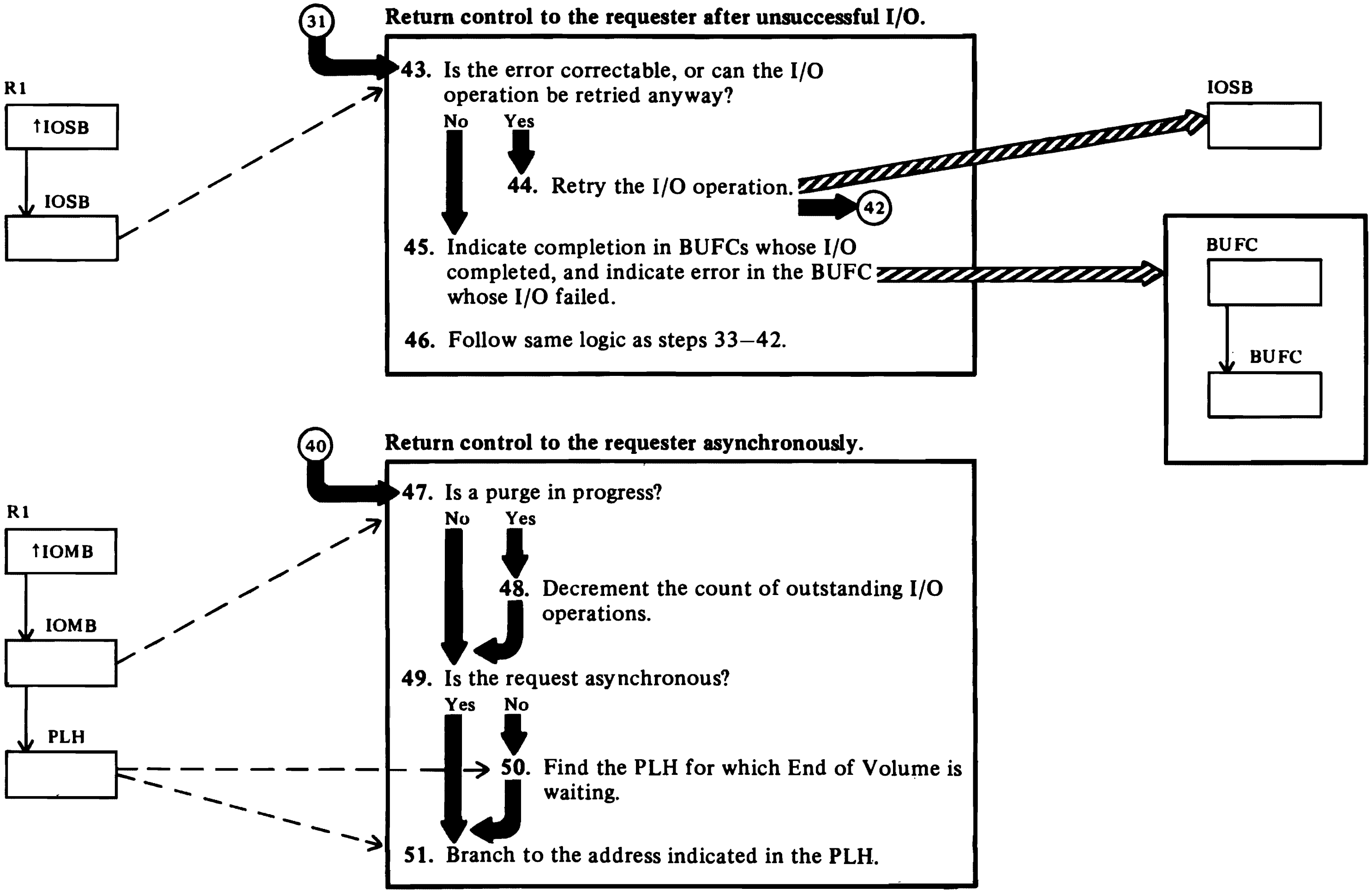
39 IDA121A3

If IGC121 determined that End of Volume should be called, IDA019R5 (IDAEOVIF) may have had to wait to give control to End of Volume, because, for control-block integrity, all the IOMBs associated with the data AMB and the index AMB must be allowed to become inactive before End of Volume executes. (See note for step 4.) IDAEOVIF is given control by way of IDA121A5 to test whether End of Volume can be executed yet. (See step 50.)

40 IDA121A3: SCHDASYN branches to the VS2 Stage II Exit Effector

Diagram DA5. I/O Management

Processing after I/O is Completed



Return control to the requester after unsuccessful I/O.

43. Is the error correctable, or can the I/O operation be retried anyway?

44. Retry the I/O operation.

45. Indicate completion in BUFCs whose I/O completed, and indicate error in the BUFC whose I/O failed.

46. Follow same logic as steps 33–42.

Return control to the requester asynchronously.

47. Is a purge in progress?

48. Decrement the count of outstanding I/O operations.

49. Is the request asynchronous?

50. Find the PLH for which End of Volume is waiting.

51. Branch to the address indicated in the PLH.

Notes for Diagram DA5

The VSAM Abnormal End Appendage is IDA121A4. It has a functional recovery routine (with entry point IDA121F4) that gets control from VS2 Recovery Termination Management when an error occurs while IDA121A4 is processing. This routine frees pages that were fixed in real storage by IGC121 and issues an SDUMP to record information in SYS1.DUMP. It issues ABEND 633(279) with reason code 4(04) if there was an invalid BUFC—one whose originally assigned virtual storage no longer belongs to the user.

43 IDA121A4: NONPERM calls **LOCATECP, CHGEPTRS**

For a correctable error, IDA121A4 finds the channel program segment where the error occurred and sets the IOSB to begin the retry operation at that channel program segment.

IDA121A4: PERMERR, ERRPROC (which call **LOCATECP, CHGEPTRS)**

An asynchronous request with error code X'41' or X'44' can be (if it hasn't already been) retried on a device whose volume can be demounted and mounted at another location. (This is called "dynamic device reconfiguration.")

45 IDA121A4: SETBITS

Request bits in a BUFC are turned off to indicate specific I/O requests were carried out. The complete bit is turned on if *all* requests were carried out.

If a protection check occurred during I/O, IDA121A4 issues an ABEND 633(279) with reason code 20(14).

The VSAM Asynchronous Routine is IDA121A5.

48 IDA121A5 branches to VS2 IECVQCNT

For asynchronous requests, the end appendages don't decrement the count of requests not yet completed. They leave it to IDA121A5. (See note for step 34.)

50 IDA121A5

If none of the PLHs indicates that End of Volume is waiting to be able to process, return to the caller. If one of them does, branching to the address indicated in this PLH (step 51) gets to End of Volume.

PROGRAM ORGANIZATION

VSAM program listings contain the details of VSAM's documentation. You get into the listings from the method of operation diagrams.

Once you have located the module or routine name that interests you in the diagrams, you are ready to turn to the listing to find the additional information you require.

Module Prologues

Each VSAM module listing begins with a description of the module, called the module prologue.

The information contained in VSAM module prologues is described in the topics that follow.

Module name: The external procedure name of the module (for example, IFG0192A).

Descriptive name: The English name of the module (for example, VSAM Open).

Status: The version and release level of the module.

Function: A brief step-by-step explanation of the functions performed by this module. Function is divided into steps so that you may more easily locate the routine responsible for each step.

Notes: A generalized heading that includes (1) any dependencies, for example, CPU model or features, that will affect the operation of this module, (2) any restrictions that apply to this module, (3) symbols used to represent registers and register usage, (4) symbolic name of the maintenance area for this module and whether the maintenance area is used or reserved, and (5) any special terms and acronyms that are used within this module that are not necessarily used elsewhere in the documentation.

Module type: A description of the type of this module (for example, procedure or macro), the name of the compiler used/required to create this module, the amount of storage required by this module for executable code and associated data, and the attributes of the module (for example, reentrant or read-only).

Entry point: The name of the point at which control can enter this module, the conditions of entry, the calling sequence by which control was given, including any parameters passed and the names of modules that may enter at this entry point.

Input: A description of anything this module gets or references, for example, registers, control blocks, and data. The means by which this module gains access to the input is included.

Output: A description of registers, control blocks, and data areas at output; any messages issued as a result of this module's processing are included.

Exit-normal: A description of conditions at and reasons for normal exit from this module, including the names of modules called by this module.

Exit-error: A description of conditions at and reasons for any error exit from this module.

External references: A list of modules, data areas, etc., defined outside of or accessible outside of this module.

Tables: A list of all local tables and work areas, that is, data areas built and used only within this module.

Macros: A description of system macros used by this module.

Change activity: A list of any change activity to this module.

Module Flow Compendiums

Reading Program Organization Compendiums

Program organization compendiums are descriptions of VSAM functions, in terms of module (procedure) calls and usage. The compendium and descriptive notes, keyed to the compendium, are on the same page whenever possible, or on facing pages.

The compendium shows the flow of control between VSAM modules in order to perform a VSAM function. Figure 5 shows a compendium figure. A single-headed arrow (between IGC0001I and IFG0193A) indicates that control is passed from one module to another and does not return. A double-headed arrow indicates that control is returned when the "called" module completes its processing.

Blocks that are indented (otherwise contained within another block) are called to perform a specified function and return, when finished, to the caller.

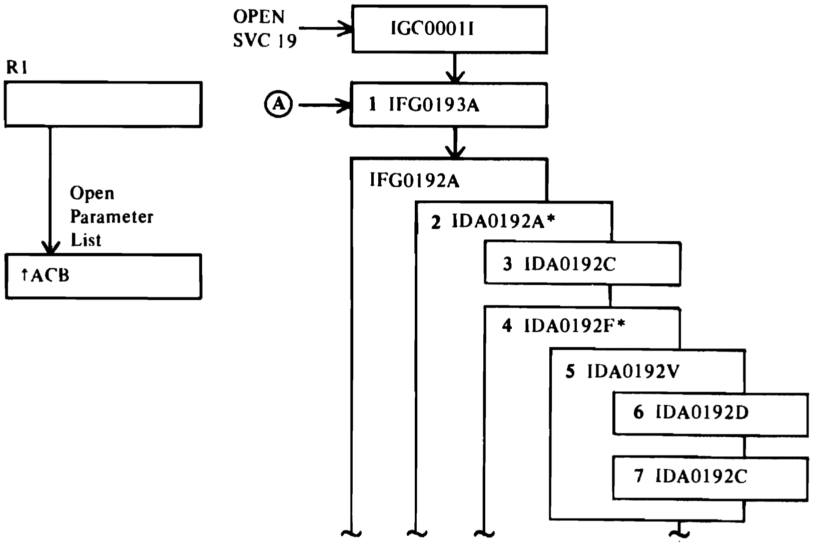


Figure 5. Program Organization Compendium Figure

For example, IDA0192A calls IDA0192C to retrieve information from the catalog.

Numbers and letters in bold-face type refer to descriptive notes. The notes tell what the caller expects the called module (procedure) to do.

Data-Set Management Compendiums

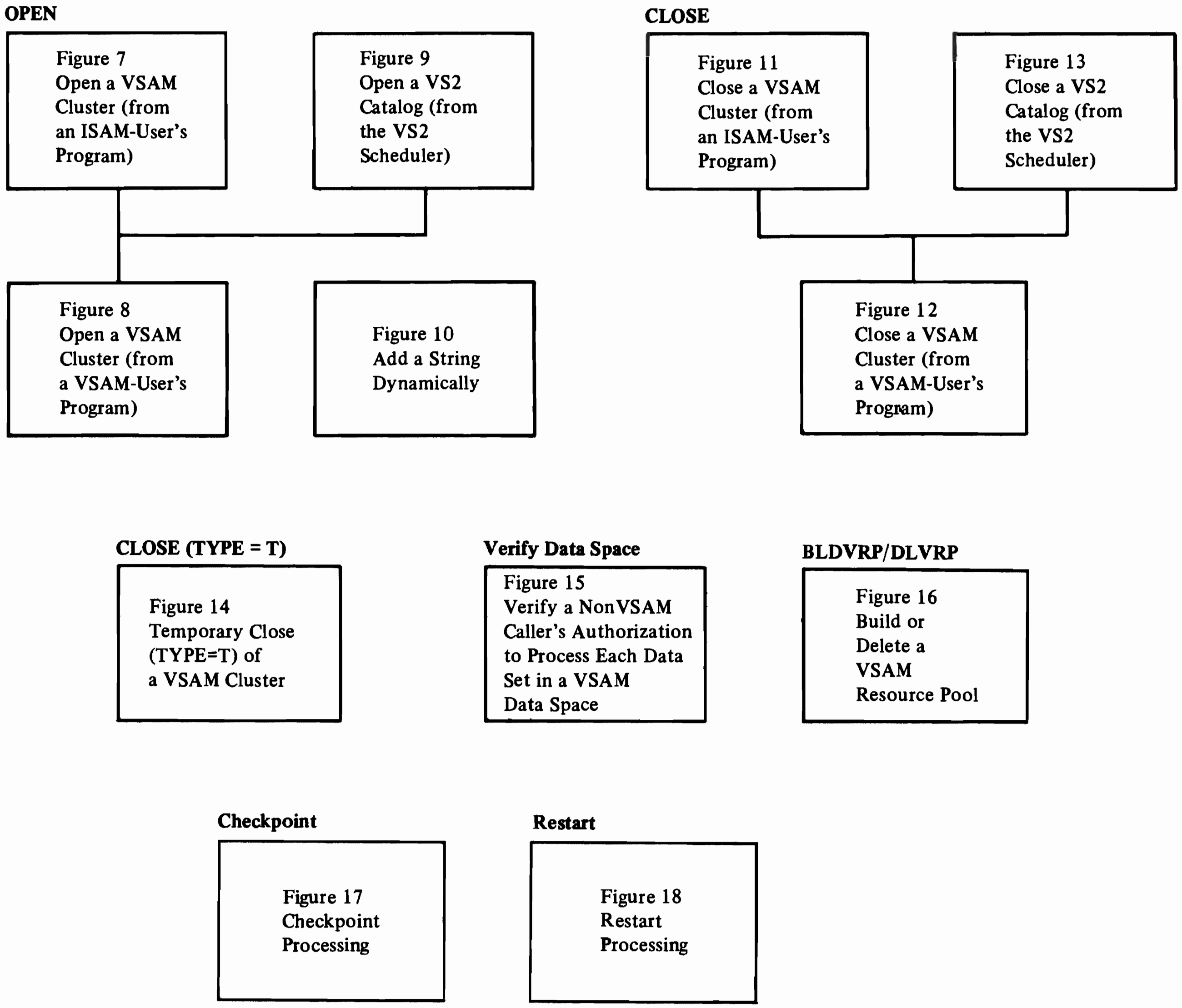

OPEN
Figure 7 Open a VSAM Cluster (from an ISAM-User's Program)
Figure 9 Open a VS2 Catalog (from the VS2 Scheduler)
Figure 8 Open a VSAM Cluster (from a VSAM-User's Program)
Figure 10 Add a String Dynamically
CLOSE
Figure 11 Close a VSAM Cluster (from an ISAM-User's Program)
Figure 13 Close a VS2 Catalog (from the VS2 Scheduler)
Figure 12 Close a VSAM Cluster (from a VSAM-User's Program)
CLOSE (TYPE = T)
Figure 14 Temporary Close (TYPE=T) of a VSAM Cluster
Verify Data Space
Figure 15 Verify a NonVSAM Caller's Authorization to Process Each Data Set in a VSAM Data Space
BLDVRP/DLVRP
Figure 16 Build or Delete a VSAM Resource Pool
Checkpoint
Figure 17 Checkpoint Processing
Restart
Figure 18 Restart Processing

Figure 6. Data Set Management Program Organization Contents

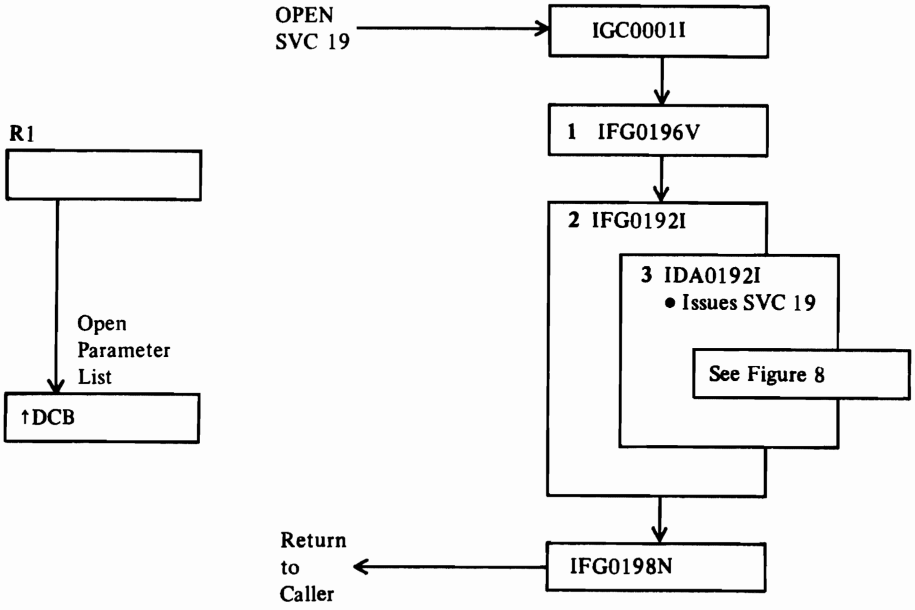


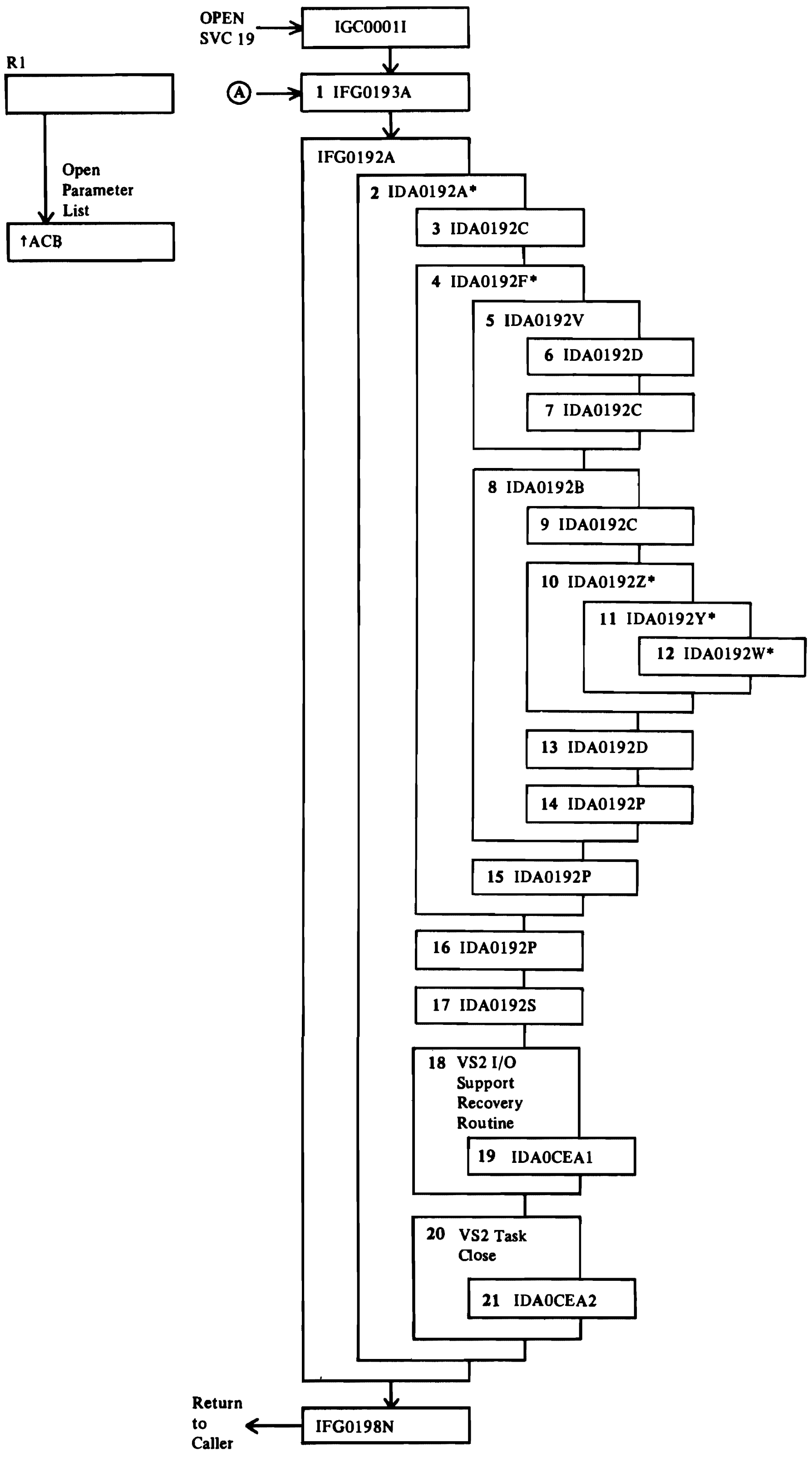
Figure 7. Open a VSAM Cluster (from an ISAM-User's Program)

Notes for Figure 7

1 IGC0001I, IFG0196V, and IFG0198N are VS2 Open modules (see *OS/VS2 Open/Close/EOV Logic* for details).

2 IFG0192I is an alias-name for IFG0192A.

3 IDA0192I is the ISAM-Interface: Open module. It is an alias for IDA0192A.



Note: * indicates that the module calls IDA0192M for virtual storage. IDA0192M is the VSAM Virtual-Storage Manager.

Figure 8. Open a VSAM Cluster (from a VSAM-User's Program)

Notes for Figure 8

1 IGC0001I, IFG0193A, and IFG0198N are VS2 Open modules (see *OS/VS2 Open/Close/EOV Logic* for details).

A IFG0191Y (in Figure 9) XCTLs to IFG0193A to open a VS2 catalog. Open-processing and return-to-the-caller continues as shown in this figure.

2 IDA0192A is the VSAM Open module. It builds the BIB, WSHD, and dummy DEB.

3 IDA0192C calls VS2 Catalog Management (LOCATE) to retrieve information about the VSAM object being opened from its catalog record.

4 IDA0192F opens base, path, and upgrade clusters. It builds the ACB, AMBL, CMB, UPT, VAT, and VMT.

5 IDA0192V ensures that the required minimum number of the object's direct-access volumes are mounted.

6 If the data set is stored on a mass storage volume, IDA0192D stages (via a Mass Storage System ACQUIRE) the data set to a direct-access storage device.

7 IDA0192C checks the time stamp.

8 IDA0192B opens VSAM clusters.

9 IDA0192C calls VS2 Catalog Management (LOCATE) to retrieve volume serial numbers from the object's catalog record.

10 IDA0192Z builds the following control blocks:

AMB
AMBXN
AMDSB
ARDB
DEB
EDB
IQE
IRB
IWA
LPMB

11 IDA0192Y builds the:

BUFC
Buffers
IOMB
IOSB
PLH
RPL
SRB
WAX

12 IDA0192W builds the CPA control block.

13 If the data set is stored on a mass storage volume, IDA0192D stages (via a Mass Storage System ACQUIRE) the data set to a direct-access storage device.

14 Whenever a VSAM Open module detects an error, IDA0192P issues a diagnostic message and traces VSAM control blocks if the Generalized Trace Facility (GTF) is active.

15 Same as step 14.

16 Same as step 14.

17 IDA0192S writes SMF record type 62.

18-19
IDAOCEA1 runs as an ESTAE exit routine when an error occurs in Open. It logs system information and returns to the VS2 I/O Support Recovery Routine to continue with termination.

20-21
IDAOCEA2 locates and frees storage used for VSAM data sets in the system queue area and the common service area.

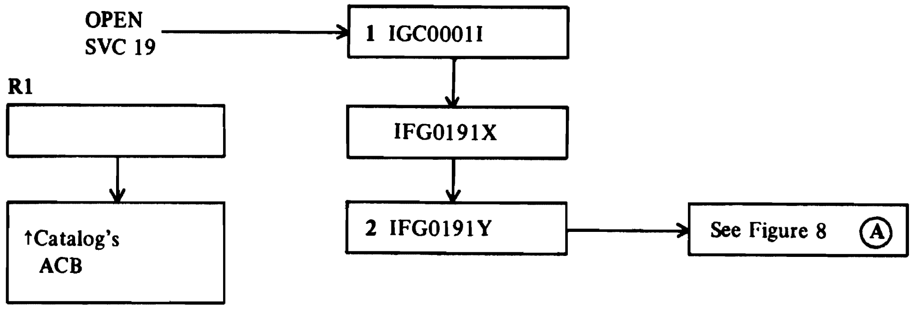


Figure 9. Open a VS2 Catalog (from the VS2 Scheduler)

Notes for Figure 9

1 IGC0001I is a VS2 Open module.

2 IFG0191X and IFG0191Y are VS2 Catalog Open: ACB Processing modules. These modules perform special processing for the catalog's ACB, then transfer control (using the XCTL macro) to IFG0193A (in Figure 8).

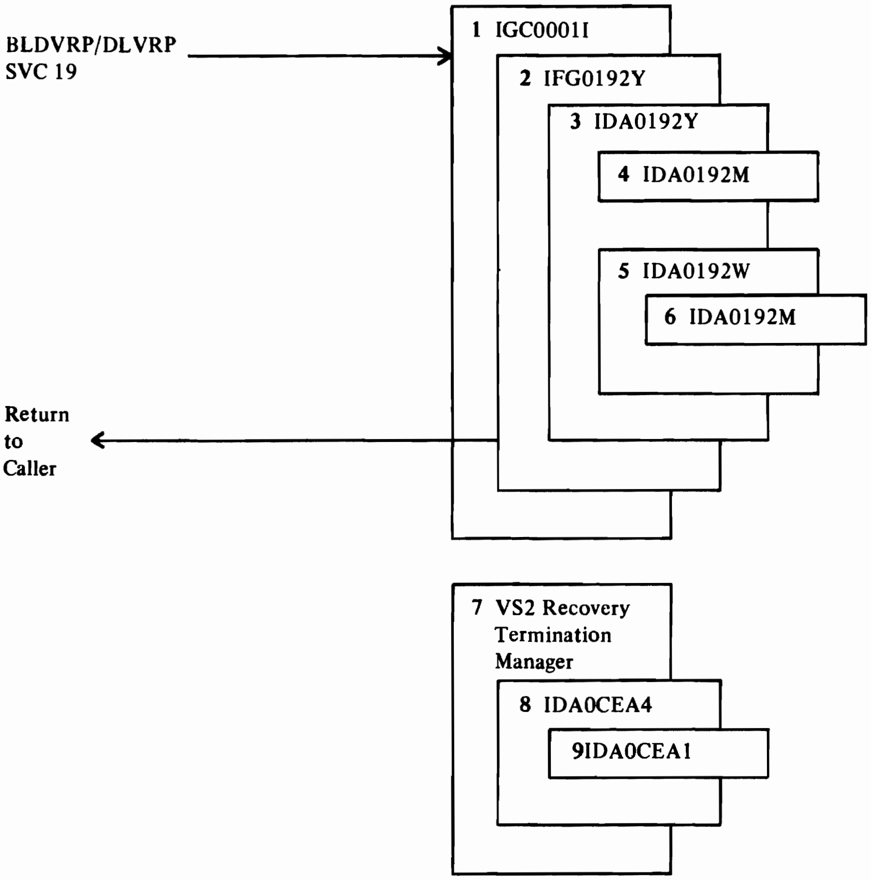


Figure 10. Add a String Dynamically

Notes for Figure 10

1 IGC0001I determines whether SVC 19 was issued for a VSAM ACB (subtype X'11'). It obtains a pseudo FORCORE: base prefix, extended prefix, WTG, RPL, and work area. The visual ID is 'VRP.'

2 IFG0192Y is the second entry point in csect IFG0192A in load module IFG0192A. It:

- Sets audit-trail information in FORCORE
- Ensures that the key of the parameter list is the caller's key, which IFG0192Y uses for processing
- Establishes DXVKEY, DXUDCBAD, and DXPDCBAB
- Moves the BLDVRP parameter list (which appears to be an ACB with subtype X'11') to protected storage
- Establishes ESTAE (IDAOCEA4) in case an error occurs in subsequent processing

When IFG0192Y receives control back from IDA0192Y, it:

- Cancels the ESTAE
- Frees the pseudo FORCORE (with IECRES macro)

3 IDA0192Y builds conrol blocks to add a string for Record Management processing. It:

- Enqueues busy on the data set to prevent concurrent Data-Set Management requests (OPEN, CLOSE, CLOSE(TYPE=T))
- Builds string blocks for the data AMB: BUFC, PLH/IOMBXN, IOMB/IQE, SRB/IOSB/PFL, CPA (and, for a path, RPL/WAX)
- For a key-sequenced data set, builds string blocks for the index AMB: BUFC, CPA
- Adjusts the string count in the CMB for the data set
- Chains string blocks (with swap/save)

4 IDA0192M allocates virtual storage for IDA0192Y to use to build control blocks.

5 IDA0192W builds required channel programs and CPAs.

6 Same as step 4.

7 The VS2 Recovery Termination Manager gets control when an error occurs in VS2.

8 IDAOCEA4 is the BLDVRP/DLVRP ESTAE routine.

9 IDAOCEA1 is the Data-Set Management Recovery Routine for error recording.

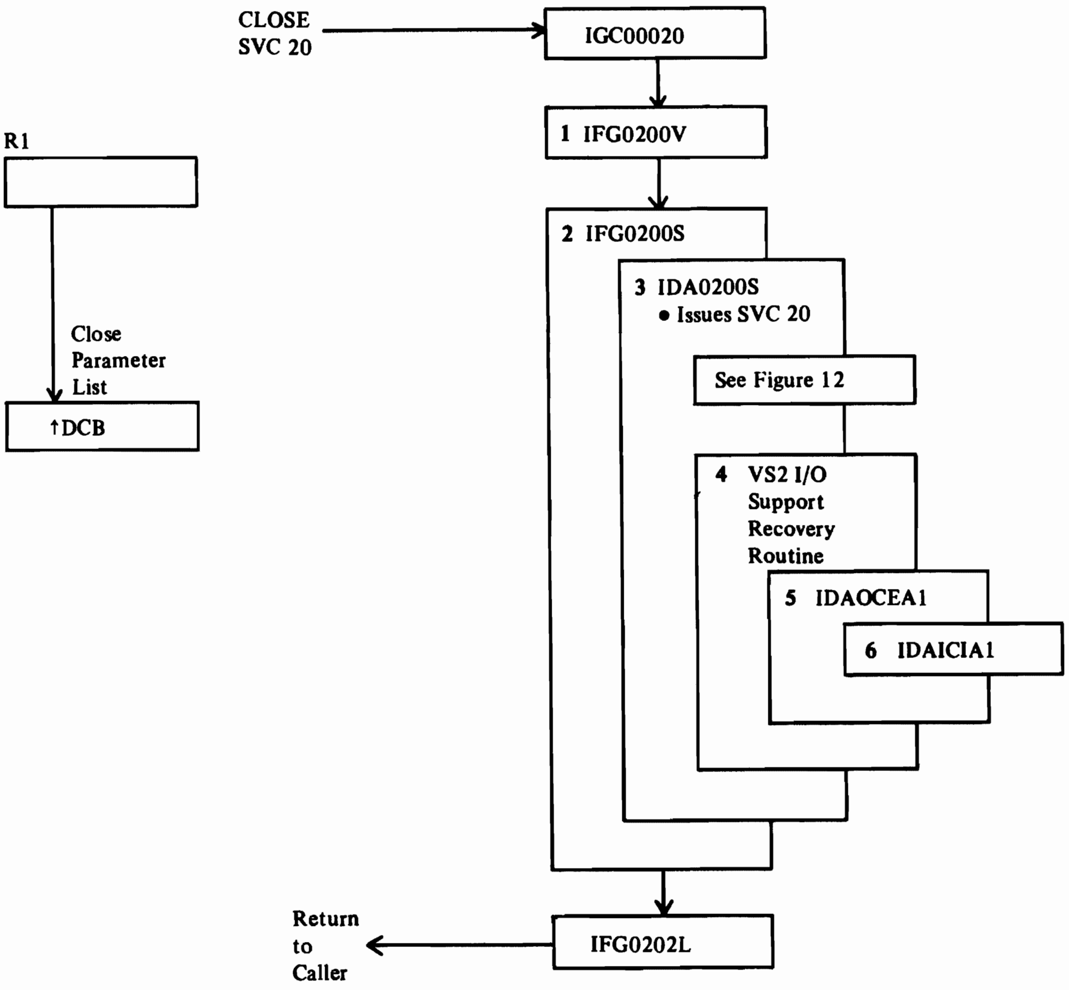


Figure 11. Close a VSAM Cluster (from an ISAM-User's Program)

Notes for Figure 11

1 IGC00020, IFG0200V, and IFG0202L are VS2 Close modules (see *OS/VS2 Open/Close/EOV Logic* for details).

2 IFG0200S is an alias-name for IFG0192A.

3 IDA0200S is the ISAM-Interface Close module. It is an alias for IDA0192A.

4-6 The VS2 I/O Support Recovery Routine is an ESTAE routine. IDAOCEA1 is the VSAM ESTAE routine, and IDAICIA1 is the ISAM-Interface ESTAE routine. IDAICIA1 frees ISAM-Interface work areas and records information in SYS1.DUMP or the SYSABEND data set.

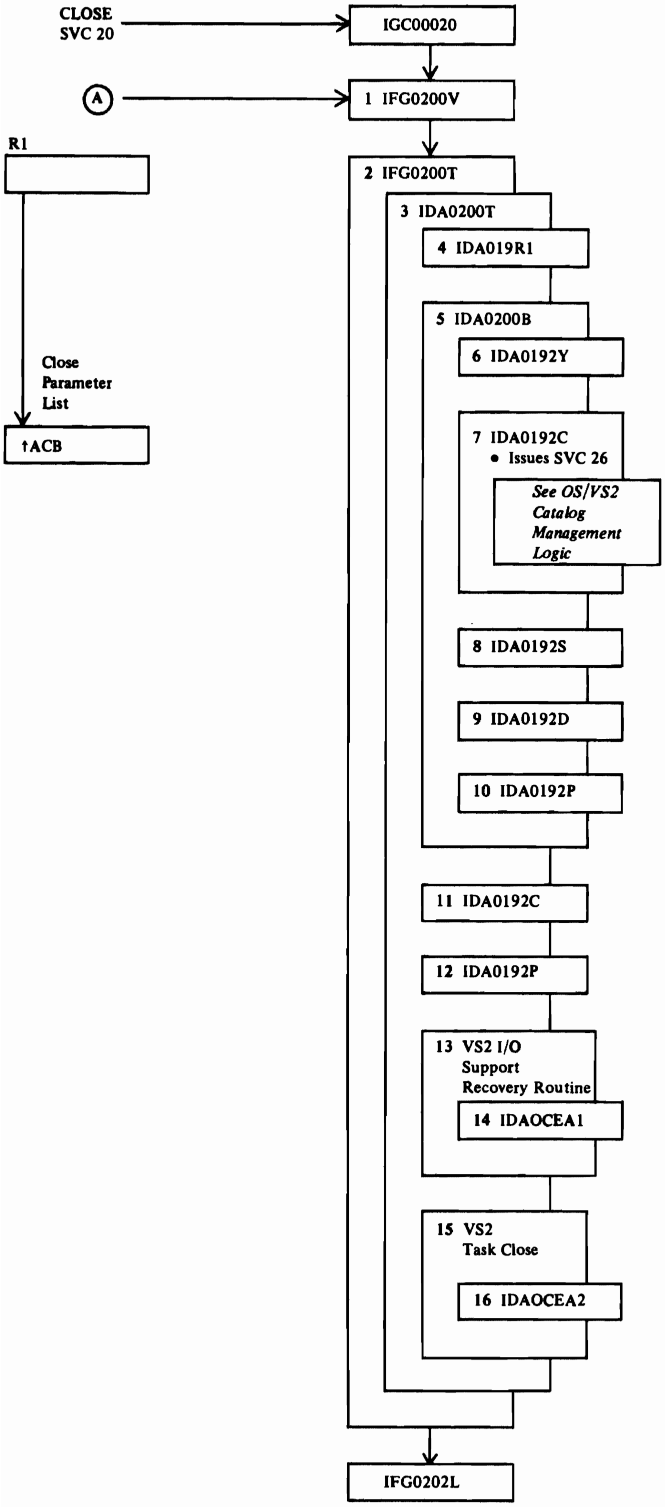


Figure 12. Close a VSAM Cluster (from a VSAM-User's Program)

Notes for Figure 12

1 IGC00020, IFG0200V, and IFG0202L are VS2 Close modules (see *OS/VS2 Open/Close/EOV Logic* for details).

A IFG0200N (in Figure 13) XCTLs to IFG0200V to close a VS2 catalog or catalog recovery area. Close-processing and return-to-the-caller continues as shown in this figure.

2 IFG0200T is an alias-name for IFG0192A.

3 IDA0200T is the VSAM Close module. It frees control blocks.

4 For an output data set, IDA0200T issues an ENDREQ macro for VSAM to write out buffers and finish I/O for the data set. (See Figure 31.)

5 IDA0200B closes VSAM clusters.

6 IDA0192Y builds control blocks for the WRTBFR macro, when Record Management indicates they are needed. The storage obtained for the control blocks is freed by Close.

7 IDA0192C calls VS2 Catalog Management (UPDATE) to modify statistical information in the object's catalog record.

8 IDA0192S writes SMF record(s) type 64.

9 If the data set is stored on a mass storage volume, IDA0192D destages (via a Mass Storage System RELINQUISH) the data set from direct-access storage to mass storage.

10 Whenever IDA0200B detects an error, IDA0192P issues a diagnostic message.

11 When a catalog is being closed, IDA0192C calls VS2 Catalog Management (LOCATE) to indicate that Close has finished.

12 IDA0192P issues a diagnostic message whenever IDA0200T detects an error.

13-14
IDAOCEA1 runs as an ESTAE exit when an error occurs in Open. It logs system information and returns to the VS2 I/O Support Revovery Routine to continue with termination.

15-16
IDAOCEA2 locates and frees storage used for VSAM data sets in the system queue area and the common service area.

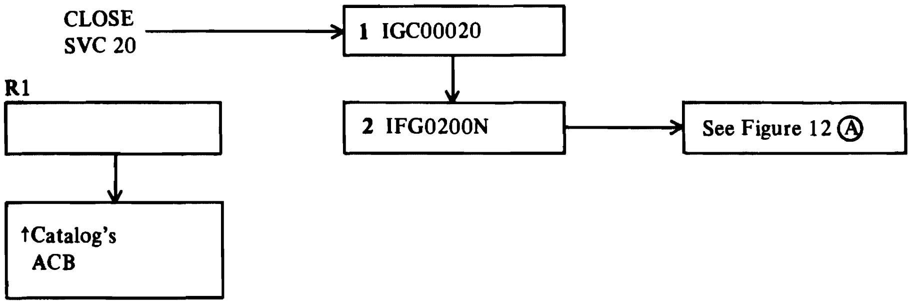


Figure 13. Close a VS2 Catalog (from the VS2 Scheduler)

Notes for Figure 13

1 IGC00020 is a VS2 Close module (see *OS/VS2 Open/Close/EOV Logic* for details).

2 IFG0200N is the VS2 Catalog Close: ACB Processing module. It performs special processing for the catalog's ACB, then XCTLs to IFG0200V (in Figure 12).

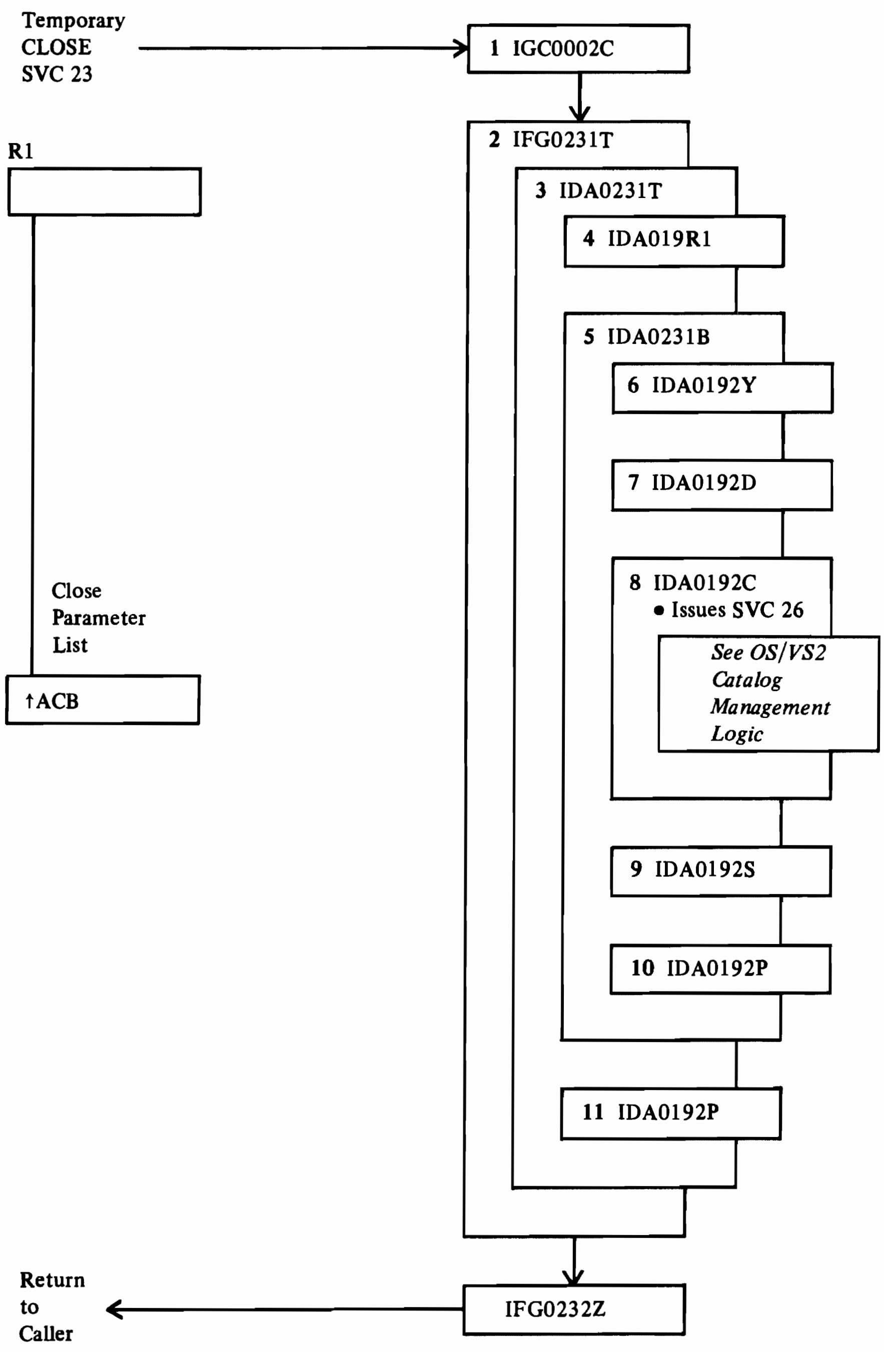


Figure 14. Temporary Close (TYPE=T) of a VSAM Cluster

Notes for Figure 14

1 IGC0002C and IFG0232Z are VS2 temporary-Close modules (see *OS/VS2 Open/Close/EOV Logic* for details).

2 IFG0231T is an alias-name for IFG0192A.

3 IDA0231T is the VSAM temporary-Close module.

4 For an output data set, IDA0231T issues an ENDREQ macro for VSAM to write out buffers and finish I/O for the data set. (See Figure 31.)

5 IDA0231B is the VSAM Close (TYPE=T) module for closing clusters.

6 IDA0192Y builds control blocks for the WRTBFR macro, when Record Management indicates they are needed. The storage obtained for the control blocks is freed by Close.

7 If the data set is stored on a mass storage volume and is defined with the DESTAGEWAIT attribute, IDA0192D destages (via a Mass Storage System RELINQUISH) the data set from direct-access storage to mass storage and waits until destaging is completed. If the data set was not bound in direct-access storage, IDA0192D restages (via a Mass Storage System ACQUIRE) the data set from mass storage to direct-access storage.

8 IDA0192C calls VS2 Catalog Management (UPDATE) to modify statistical information in the object's catalog record.

9 IDA0192S writes SMF record(s) type 64.

10 Whenever IDA0231B detects an error, IDA0192P issues a diagnostic message.

11 IDA0192P issues a diagnostic message whenever IDA0231T detects an error.

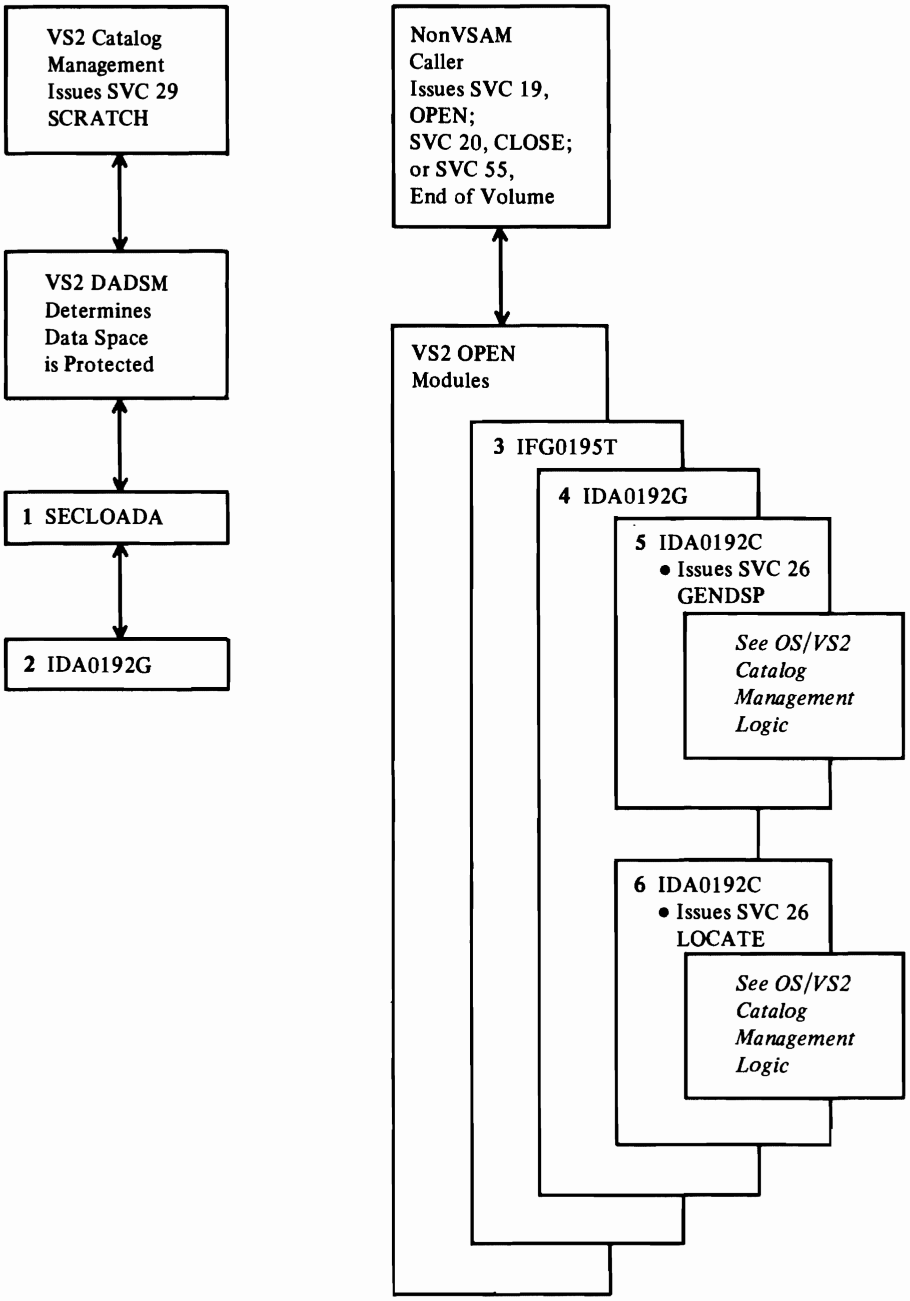

VS2 Catalog Management Issues SVC 29 SCRATCH
VS2 DADSM Determines Data Space is Protected
1 SECLOADA
2 IDA0192G
NonVSAM Caller Issues SVC 19, OPEN; SVC 20, CLOSE; or SVC 55, End of Volume
VS2 OPEN Modules
3 IFG0195T
4 IDA0192G
5 IDA0192C
• Issues SVC 26 GENDSP
See OS/VS2 Catalog Management Logic
6 IDA0192C
• Issues SVC 26 LOCATE
See OS/VS2 Catalog Management Logic

Figure 15. Verify a NonVSAM Caller's Authorization to Process Each Data Set in a VSAM Data Space

Notes for Figure 15

Note: The VTOC contains a format-1 (identifier) DSCB to describe each VSAM data space. The DSCB indicates that the space it describes is protected and that the caller must provide the correct password before access is granted.

When a VSAM data space is shared (nonunique), the caller must provide the correct master password for each data set in the data space before he is allowed to process the data space.

1 When the caller is the VS2 DADSM Scratch Routine and the format-1 DSCB identifies a VSAM data space, SECLOADA passes control to IDA0192G. (See *OS/VS2 Open/Close/EOV Logic* for SECLOADA details.)

2 When the caller is authorized (is in key 0 and supervisor state), IDA0192G does no further checking.

3 When a utility program issues OPEN, CLOSE, or the SVC for End of Volume, IFG0195T determines that the caller is other than VSAM or ISAM Interface and that the format-1 DSCB is protected. (See *OS/VS2 Open/Close/EOV Logic* for details.)

4 IDA0192G verifies the caller's authorization to process the data space.

5 IDA0192C issues SVC 26 (GENDSP) to VS2 Catalog Management to obtain the DSNAME of each VSAM data set in the data space.

6 IDA0192C issues SVC 26 (LOCATE) to cause VS2 Catalog Management to verify that the caller can supply each protected data set's master password.

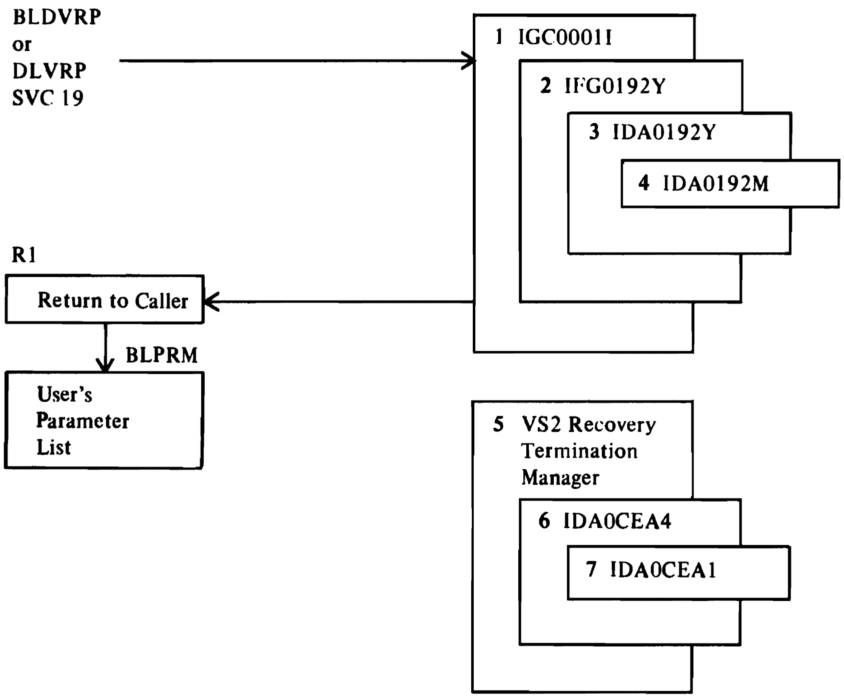


Figure 16. Build or Delete a VSAM Resource Pool

Notes for Figure 16

1 IGC0001I determines whether SVC 19 was issued for a VSAM ACB (subtype X'11') and determines whether SVC 19 is for BLDVRP or DLVRP. It obtains a pseudo FORCORE: base prefix, extended prefix, WTG, RPL, and work area. The visual ID is 'VRP'

2 IFG0192Y is the second entry point in csect IFG0192A in load module IFG0192A. It:

- Sets audit-trail information in FORCORE
- Ensures that the key of the parameter list is the caller's key, which IFG0192Y uses for processing
- Establishes DXVKEY, DXUDCBAD, and DXPDCBAB
- Moves the BLDVRP parameter list (which appears to be an ACB with subtype X'11') to protected storage
- Establishes ESTAE (IDAOCEA4) in case an error occurs in subsequent processing

When IFG0192Y receives control back from IDA0192Y, it:

- Cancels the ESTAE
- Frees the pseudo FORCORE (with IECRES macro)

3 IDA0192Y does validity checking and builds or deletes control blocks for a VSAM global (GSR) or local (LSR) resource pool: VSRT, WSHD, CPA header, PLHs, BSPH, BUFCs, buffers. For BLDVRP, it chains the VSRT to the VAT; for DLVRP, it unchains it. It uses a pseudo FORCORE, an ACB work area (IDAOPWRK), and a copy of the BLDVRP parameter list in protected storage.

4 IDA0192M allocates virtual storage for IDA0192Y to use to build control blocks.

5 The VS2 Recovery Termination Manager gets control when an error occurs in VS2.

6 IDAOCEA4 is the BLDVRP/DLVRP ESTAE routine.

7 IDAOCEA1 is the Data-Set Management Recovery Routine for error recording.

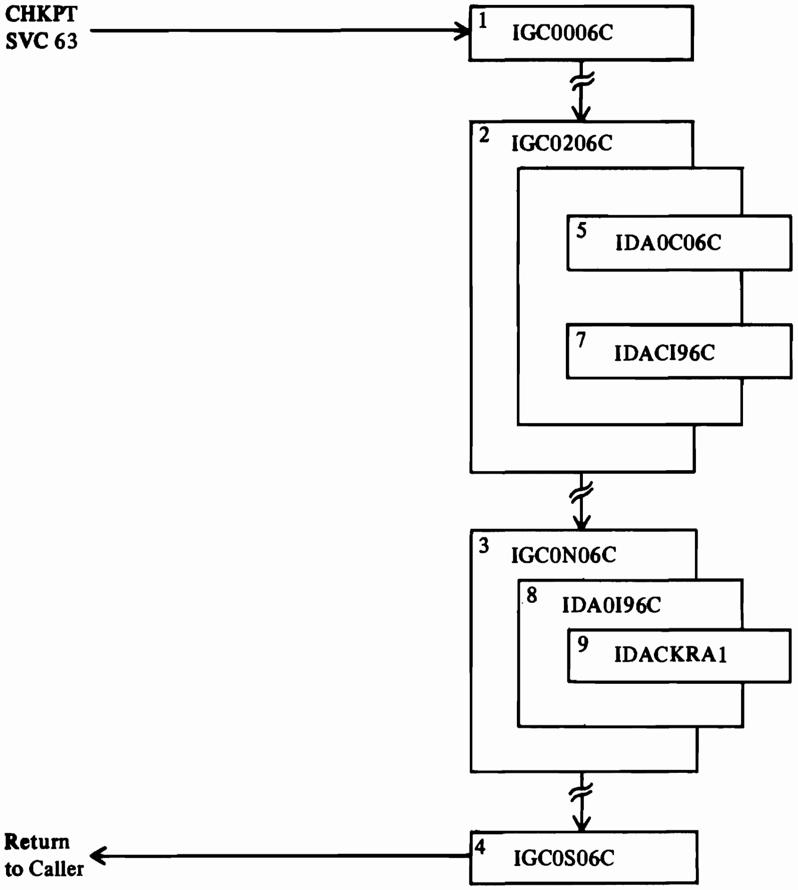


Figure 17. Checkpoint Processing

Notes for Figure 17

1-4 IGC0006C, IGC0206C, IGC0N06C, and IGC0S06C are VS2 checkpoint modules described in *OS/VS2 Checkpoint/Restart Logic*.

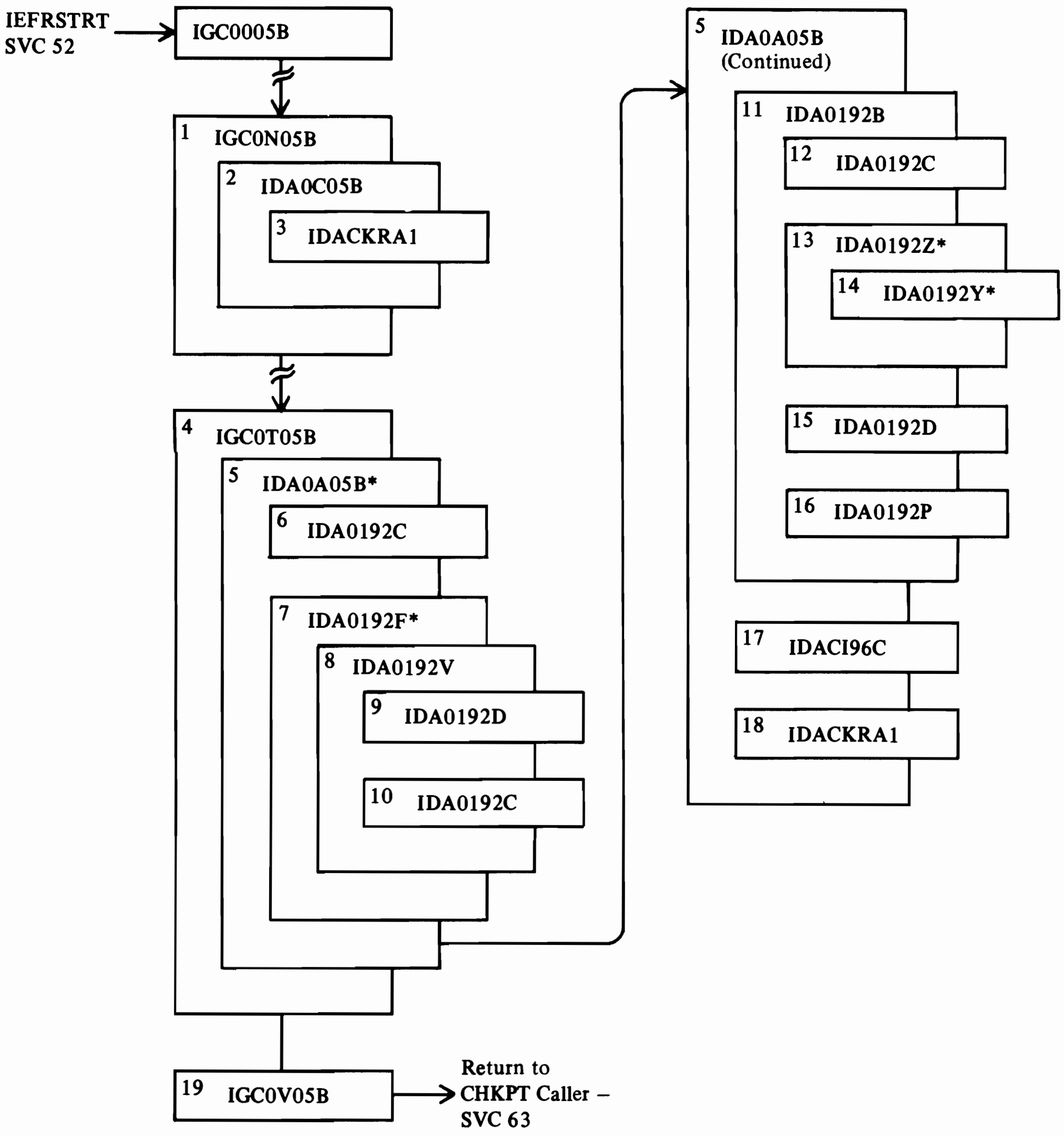
5 IDA0C06C is the VSAM checkpoint module. It saves information required by restart in VCRCORE.

6 IDACKRA1 is the VSAM checkpoint/restart ESTAE routine. It provides problem determination information and attempts to pass control to a retry routine.

7 If any errors occur during checkpoint processing, IDACI96C frees all VCRCORE.

8 If a valid JSCBSHR field exists, IDA0196C builds an SSCR. If any VCRCORE exists, it is freed.

9 Same as step 6.



*This module calls IDA0192M, the VSAM Virtual Storage Manager.

Figure 18. Restart Processing

Notes for Figure 18

1 IGC0005B and IGC0N05B are VS2 restart modules described in *OS/VS2 Checkpoint/Restart Logic.*

2 IDA0C05B is the VSAM restart SSCR and DEB module. It restores the JSCBSHR, frees AMB DEBs, and sets DEBXCDCB to mark VSAM ACBs noncloseable.

3 IDACKRA1 formats a message in the SDWA, executes the SDUMP macro, and either returns to a retry routine or continues with abnormal termination.

4 IGC0T05B is a VS2 restart module described in *OS/VS2 Checkpoint/Restart Logic.*

5 IDA0A05B is the VSAM restart module. It rebuilds VGTTs and HEBs and does repositioning and verify processing.

6 IDA0192C obtains the relatonships to the cluster being restarted.

7 IDA0192F builds the VMTs for the sphere.

8 For each VCRT processed, IDA0192V ensures that the required number of the cluster's direct-access volumes are mounted.

9 If the data set is stored on a mass storage volume, IDA0192D stages (via a Mass Storage System ACQUIRE) the data set to a direct-access storage device.

10 IDA0192C checks the time stamp.

11 IDA0192B performs volume processing for each AMB in the cluster and does share processing for the cluster.

12 IDA0192C calls VS2 Catalog Management (LOCATE) to retrieve volume serial numbers from the cluster's catalog records.

13 IDA0192Z refreshes AMDSBs and ARDBs and rebuilds DEBs and EDBs.

14 IDA0192Y rebuilds SRBs, IOSBs, and PFLs.

15 If the data set is stored on a mass storage volume, IDA0192D stages (via a Mass Storage System ACQUIRE) the data set to a direct-access storage device.

16 Whenever a VSAM Open module detects an error, IDA0192P issues a diagnostic message and traces VSAM control blocks if the Generalized Trace Facility (GTF) is active.

17 IDACI96C is an external entry for IDA0196C and performs cleanup functions for VSAM Checkpoint/Restart control blocks.

18 Same as step 3.

19 IGC0V05B is a VS2 restart module described in *OS/VS2 Checkpoint/Restart Logic.*

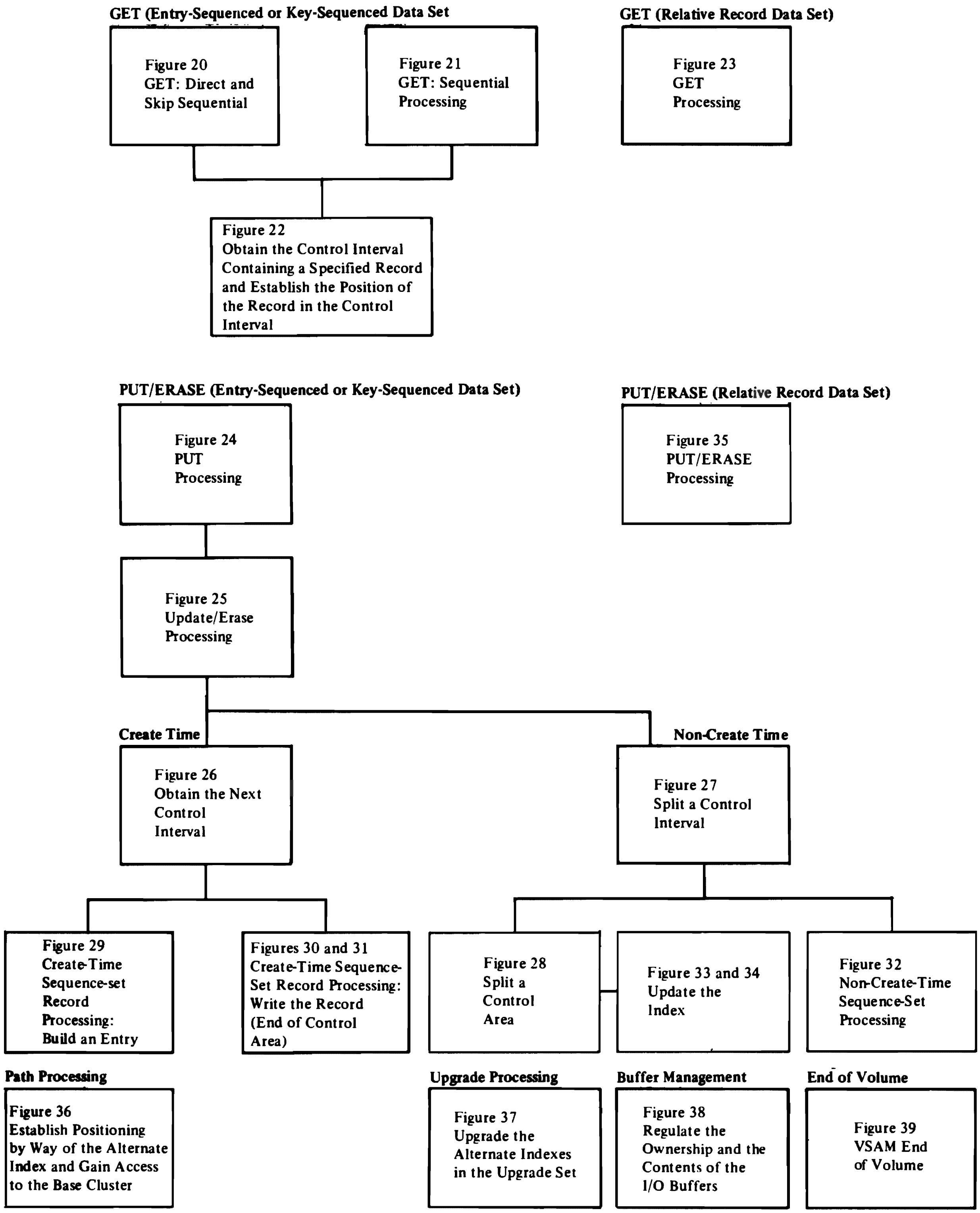


Figure 19. Record-Management Program Organization Contents

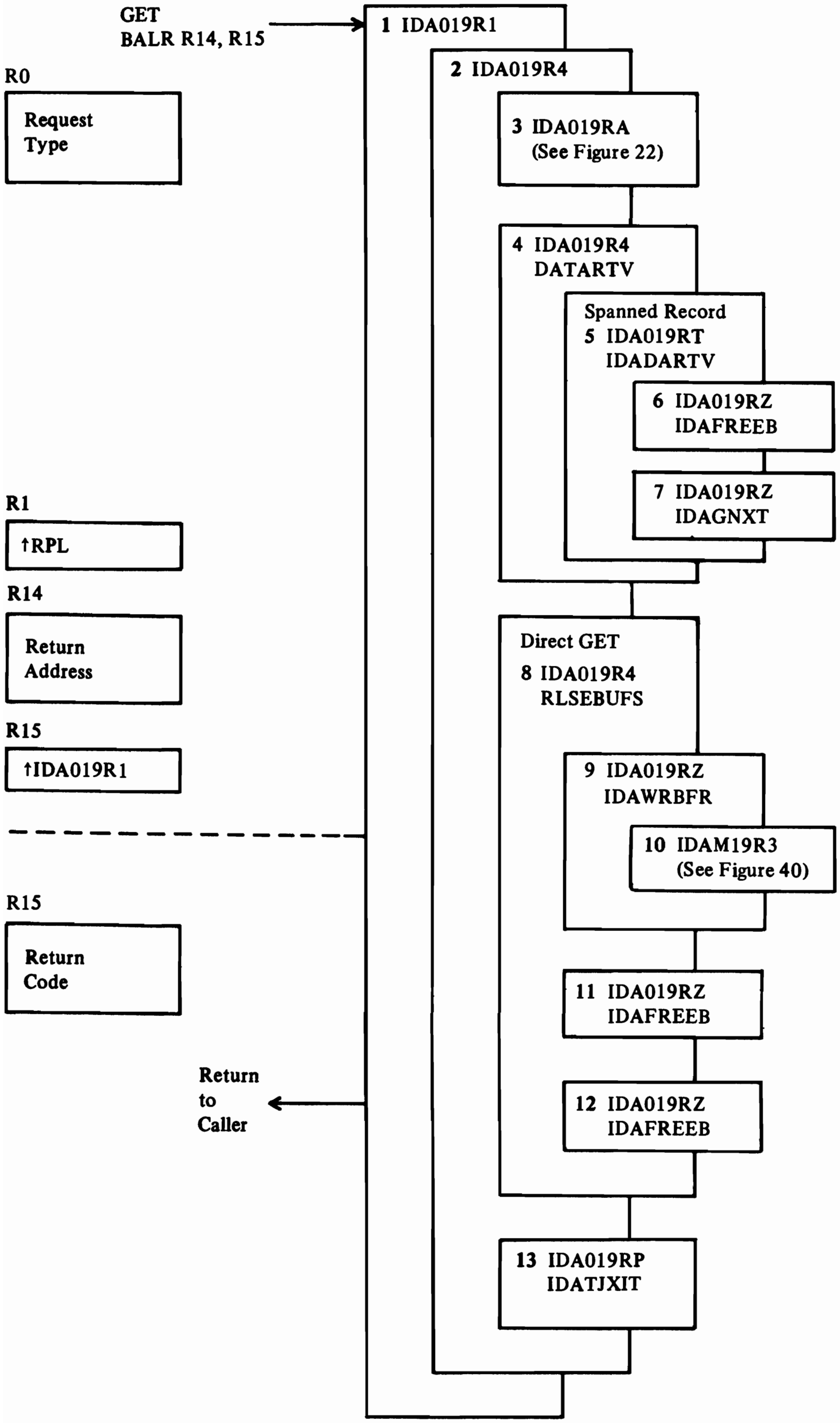


Figure 20. GET: Direct and Skip Sequential Processing (ESDS, KSDS)

Notes for Figure 20

1 IDA019R1 is the common Record Management request module. It verifies that the request is a valid Record-Management macro, then tests the RPL for keyed or addressed processing.

2 When the request requires either keyed or addressed processing (not a control-interval-processing request), IDA019R4 selects the correct processing path for either GET, PUT, or POINT, and for sequential, skip sequential, or direct processing.

3 When the request is either direct GET or skip sequential GET, IDA019RA locates the position of the desired data record in its control interval.

4 DATARTV makes an unspanned data record available to the caller. It sets the RBA of the data record into the RPL. If the caller's request is in locate mode, DATARTV returns a pointer to the record to the caller. If the request is in move mode, DATARTV moves the data record into the caller's record area.

5 IDADARTV moves all of the segments of a spanned record into the user's area.

6 IDAFREEB frees the buffer.

7 IDAGNXT moves the next segment into a buffer.

8 If the request is direct GET and the caller doesn't want to retain the record's position for subsequent record processing requests, RLSEBUFS releases the data record's buffer.

9 If the buffer was changed by a previous update request, IDAWRBFR rewrites the buffer's control interval into the data set.

10 See Figure 40.

11 IDAFREEB frees the data buffer.

12 If the request is keyed, IDAFREEB frees the buffer containing the sequence set control interval associated with the data buffer.

13 If the user's EXLST contains an active journal exit address, IDATJXIT provides the necessary journaling information for the user's journal exit routine.

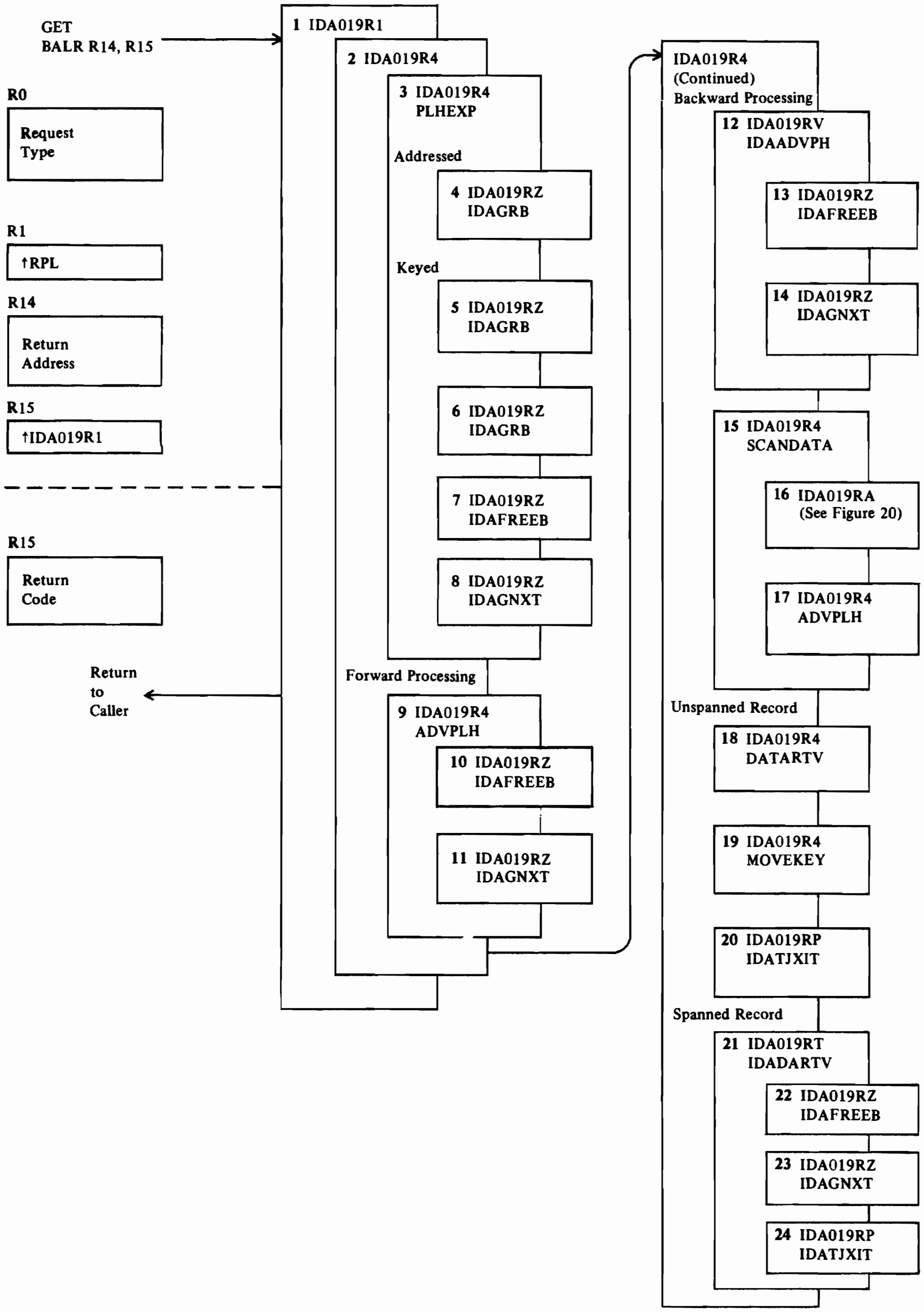


Figure 21. GET: Sequential Processing (ESDS, KSDS)

Notes for Figure 21

1 IDA019R1 is the common Record-Management request module. It verifies that the request is a valid Record-Management macro, then tests the RPL for keyed or addressed processing.

2 When the request requires either keyed or addressed processing (not a control-interval-processing request), IDA019R4 selects the correct processing path for either GET, PUT, or POINT, and for sequential, skip sequential, or direct processing.

3 When the request is sequential GET, PLHEXP tests the status indicators in the placeholder (PLH) to determine if an exceptional condition occurred:

- If the request is the first request after the data set is opened, and isn't preceded by a POINT to position to a starting record:

  4 If the request is addressed, IDAGRB reads in the first data control interval of the data set.

  5 If the request is keyed, IDAGRB reads in the first sequence set control interval.

  6 The sequence set control interval is used to determine the RBA of the first data control interval. IDAGRB retrieves the first data control interval of the key-sequenced data set.

  7 If the first control interval of the key-sequenced data set is empty, IDAFREEB frees its buffer.

  8 IDAGNXT obtains the next control interval of the key-sequenced data set. Steps **7** and **8** are repeated as often as necessary to obtain a nonempty control interval of the key-sequenced data set.

- If the end of data condition occurs, PLHEXP sets a return code and returns to the caller.
- If a read error occurs, ADVPLH skips over the bad data, resets the PLH so that it points to the next good data control interval's RBA, and returns to the caller with a return code set.
- If the previous request encountered a read-exclusive error (not allowed to read the record because another user has exclusive control over it), SCANDATA searches the index to locate the requested record.

9 If no exceptional conditions have been detected, the PLH now points to the record most recently processed by the user. ADVPLH adjusts the PLH so that it points to the next record (desired by this request) in the buffer.

10 If there are no more records in the buffer (that is, the record most recently processed by the user is the control interval's last record), IDAFREEB frees the buffer.

11 IDAGNXT retrieves the next sequential control interval, unless another buffer already contains the control interval. The PLH is set to point to the first data record in the control interval.

12 If no exceptional conditions have been detected, the PLH now points to the record most recently processed by the user. IDAADVPH adjusts the PLH to point to the previous record (desired by this request) in the buffer.

13 If there are no more records in the buffer (that is, the record last processed by the user is the control interval's first record), IDAFREEB frees the buffer.

14 IDAGNXT retrieves the next control interval in descending sequence, unless another buffer already contains the control interval. The PLH is set to point to the last record in the control interval.

15 If the current request is GET-for-update, but the record's buffer is not under the caller's exclusive control, SCANDATA locates the record again to ensure that the PLH now points to it, even though updates might have occurred against it. The buffer is now under the caller's exclusive control.

16 IDA019RA searches the index, if the data set is key-sequenced, or uses the caller-supplied RBA, if the data set is entry-sequenced, to determine the record's location in the buffer.

17 If the placeholder needs to be updated, ADVPLH updates it after the record has been located.

18 DATARTV makes an unspanned data record available to the caller. It sets the RBA of the data record into the RPL. If the caller's request is in locate mode, DATARTV returns a pointer to the record in the caller's RPL. If the request is in move mode, DATARTV moves the data record into the caller's record area.

19 MOVEKEY saves the record's key in the placeholder.

20 If the user's EXLST contains an active journal exit address, IDATJXIT provides the necessary journaling information for the user's journal exit routine.

21 IDADARTV moves all the segments of a spanned record into the user's area.

22 IDAFREEB frees the buffer.

23 IDAGNXT moves the next segment into a buffer.

24 See the note for step 20.

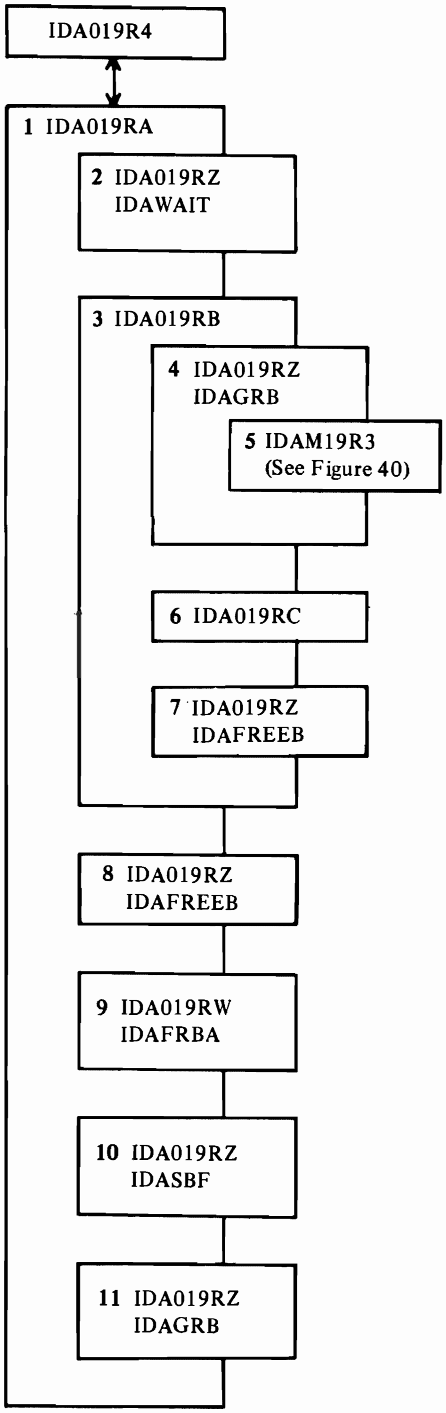


Figure 22. Obtain the Control Interval Containing a Specified Record and Find the Position of the Record in the Control Interval (ESDS, KSDS)

Notes for Figure 22

1 IDA019RA locates the position of a desired data record in a control interval that is in a VSAM Record-Management buffer.

2 If the control interval is being updated by another user, IDAWAIT waits until the updating is complete.

2 If the data set is key-sequenced, IDA019RB searches the index to find the RBA of the desired data record's control interval.

4 IDAGRB obtains an index record to search.

5 See Figure 40.

6 IDA019RC searches the index control interval to locate an index entry containing a key value equal to or greater than the search argument passed by IDA019RB. IDA019RC sets a return code to indicate the status of the search, and a pointer to the requested entry, if found.

7 If IDA019RC hasn't found the termination point for the search (determined by IDA019RB), IDAFREEB releases the buffer containing the just-searched index control interval. **4** through **7** repeat until the termination point for the search is reached.

8 If the placeholder doesn't point to the buffer containing the desired data record, IDAFREEB frees the buffer currently pointed to by the PLH.

9 IDAFRBA determine the RBA of the next sequential (or, if the request is keyed, the next higher keyed) control interval.

10 IDASBF releases all buffers (except one) pointed to by the placeholder—buffers that have been assigned to the placeholder and available for its use, but are not currently in use.

11 IDAGRB retrieves the data record's control interval, located by the previous index search if the data set is key-sequenced or by the caller-specified RBA value if the data set is entry-sequenced.

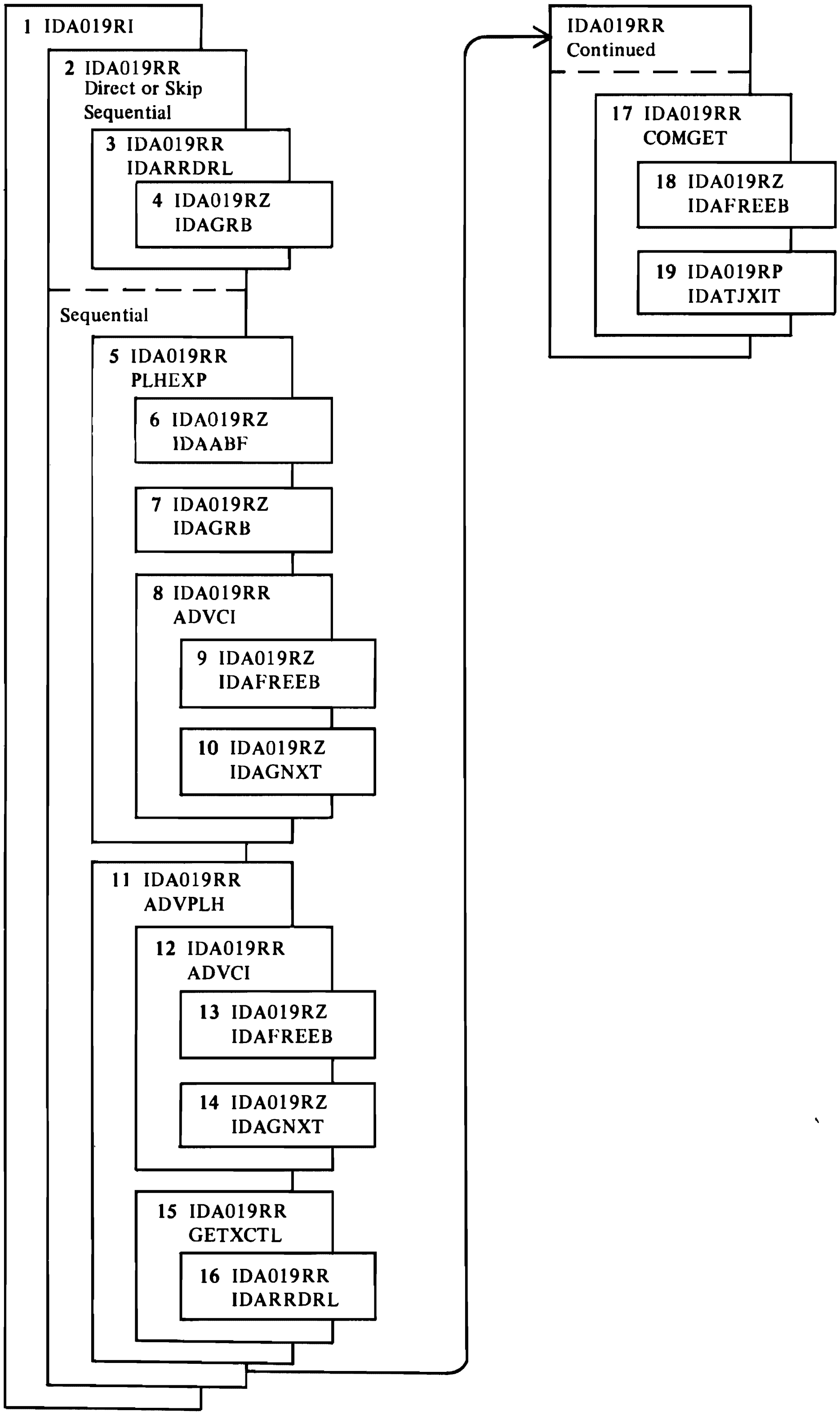


Figure 23. GET Processing (RRDS)

Notes for Figure 23

1 IDA019R1 is the common Record Management request module. It verifies that the request is valid and checks for keyed processing of a relative record data set.

2 IDA019RR selects the processing path for GET, PUT, POINT, or ERASE and for direct, sequential, or skip sequential access.

Direct or Skip Sequential

The search argument (relative record number) is converted to the RBA of the control interval that contains it and the offset of the record in the control interval.

3 For skip sequential access, IDARRDRL verifies that the search argument is greater than the previous one, indicated by positioning.

4 IDAGRB retrieves the control interval by RBA, and IDARRDRL sets the PLH pointer to the record.

Sequential

5 For sequential retrieval, positioning must have been established. Status indicators in the PLH indicate any exceptional condition, which is handled by PLHEXP:

- For the first request after OPEN, positioning is implicitly established at the beginning or the end of the data set (depending on whether processing is to be forward or backward). Steps 6 through 10 handle this exceptional condition.
- If the end of the data set (or the beginning, for backward processing) has already been reached, PLHEXP sets an error code and returns to the caller.
- If there has been a read error, PLHEXP calls ADVPLH, which skips over the unreadable control interval, searches for the next slot that contains a record, and sets the PLH pointer to the record.
- If the control interval couldn't be retrieved before because another request had exclusive control of it, PLHEXP calls GETXCTL to retrieve the control interval.

6 IDAABF adds buffers to the buffer chain for read-ahead buffering.

7 IDAGRB retrieves the first control interval and scans it for the first slot that contains a record.

8 If the control interval doesn't contain a record, ADVCI advances to the next control interval, and the next, until it finds a slot that contains a record.

9 IDAFREEB frees the current data buffer.

10 IDAGNXT retrieves the next sequential control interval.

11 For processing when there is no exceptional condition, ADVPLH advances to the next slot that contains a record and sets the PLH pointer to the record.

12 ADVCI advances to the next slot that contains a record.

13 IDAFREEB frees the current data buffer.

14 IDAGNXT retrieves the next sequential control interval.

15 For GET-update, when the buffer isn't already under exclusive control, GETXCTL retrieves the control interval with exclusive control of the buffer that contains it.

16 IDARRDRL retrieves the control interval by RBA and sets the PLH pointer to the first slot that contains a record.

Common Termination

17 COMGET sets RPL fields for the user, updates statistics, and releases positioning, if necessary.

18 For a direct request that is not for update, not to have string position noted, and not in locate mode, IDAFREEB frees the current data buffer.

19 If the user's EXLST contains an active journal exit address, IDATJXIT provides the necessary journaling information for the user's journal exit routine.

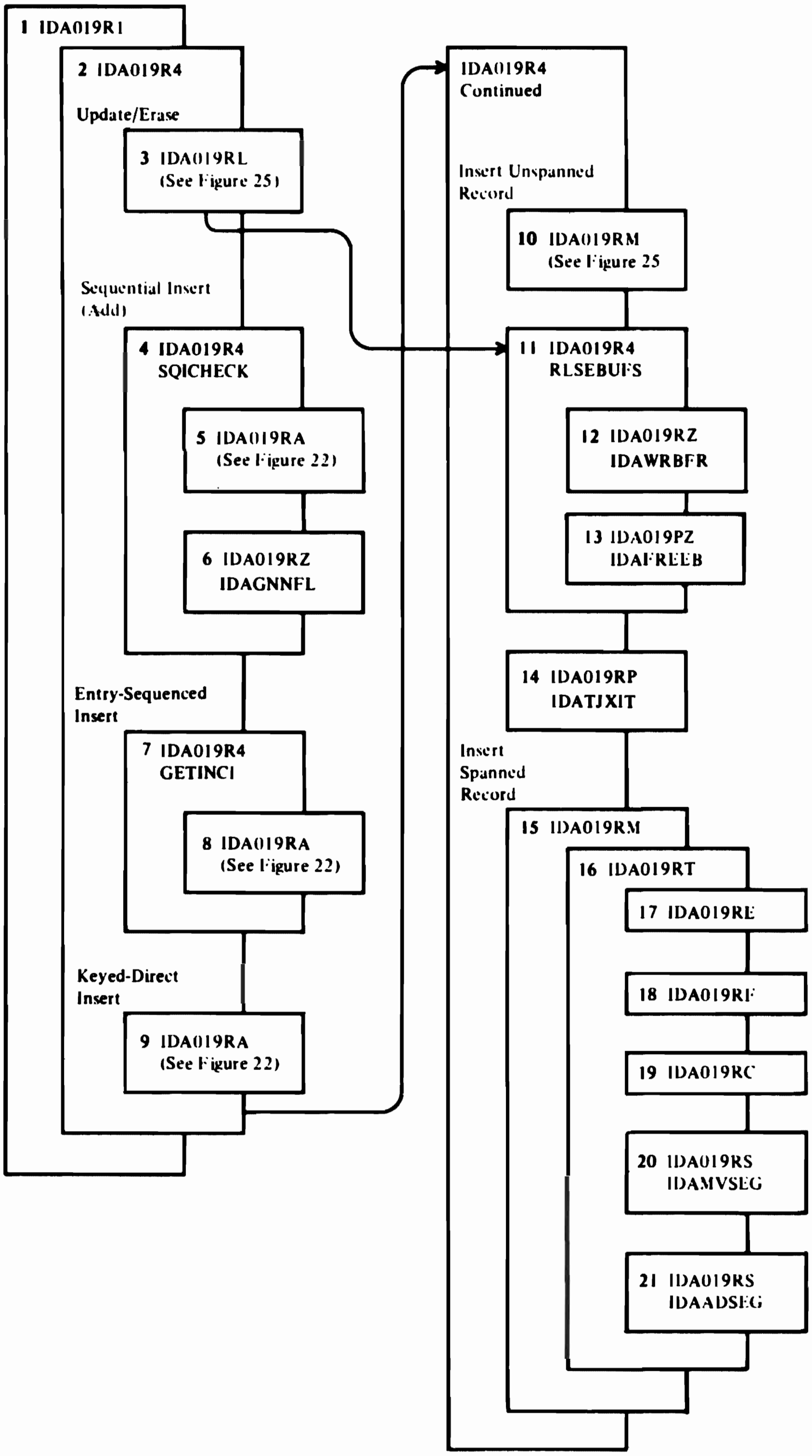


Figure 24. PUT Processing (ESDS, KSDS)

Notes for Figure 24

1 IDA019R1 is the common Record-Management request module. It verifies that the request is a valid Record-Management macro, then tests the RPL for keyed or addressed processing.

2 When the request requires either keyed or addressed processing (not a control-interval-processing request), IDA019R4 selects the correct processing path for either GET, PUT, or POINT, and for sequential, skip sequential, or direct processing.

3 When the request is PUT-Update, IDA019R4 verifies that the previous request was a GET-for-update.

4 When the record is added sequentially to a key-sequenced data set, SQICHECK ensures that the new record's key is in the correct sequence.

5 If the caller's previous request didn't establish a position in the data set, or if the key of the record to be inserted is greater than the key for the current position, IDA019RA searches the index to find the correct position for the new record to be inserted. IDA019RA returns a pointer to the insertion point for the record in the buffer. This process occurs only after the data set has been created.

6 When the first record of a data set is being written, IDAGNNFL obtains an empty buffer to build the control interval's records in. This process occurs only when the data set is being created.

7 When the request is a direct or skip-sequential insert into an entry-sequenced data set, GETINCI ensures that the last control interval that contains data records is available to receive the new data record.

8 IDA019RA locates the correct control interval and reads it into a buffer (if the request is direct).

9 When the request is either PUT-Update or ERASE, IDA019RL either replaces the old record's contents with updated information (PUT-Update) or removes the old record from the data control interval.

When the request is a direct or skip-sequential insert into a key-sequenced data set, IDA019RA searches the index to locate the correct sequence set and data control interval, and reads both control intervals into buffers.

10 IDA019RM inserts the record into the buffer at a previously determined insertion point. IDA019RM builds the record's RDF and inserts the record into the control interval, adjusting other records as necessary.

11 If the request is direct PUT and the caller doesn't want to retain the record's position for subsequent record processing requests, RLSEBUFS releases the data record's buffer.

12 If the buffer was changed by a previous update request, IDAWRBFR rewrites the buffer's control interval into the data set.

13 IDAFREEB frees the buffer currently pointed to by the PLH.

14 If the user's EXLST contains an active journal exit address, IDATJXIT provides the necessary journaling information for the user's journal exit routine.

15 IDA019RM calls IDA019RT for spanned-record insertion.

16 See note for step 15.

17 If the current buffer isn't empty, IDA019RE is called to split the control interval.

18 If the control area hasn't enough free space for the spanned record, IDA019RF is called to split the control area.

19 IDA019RC finds the position of the current entry in the sequence set.

20 IDAMVSEG moves one segment from the user's area to a buffer.

21 IDAADSEG builds a sequence-set entry for the segment.

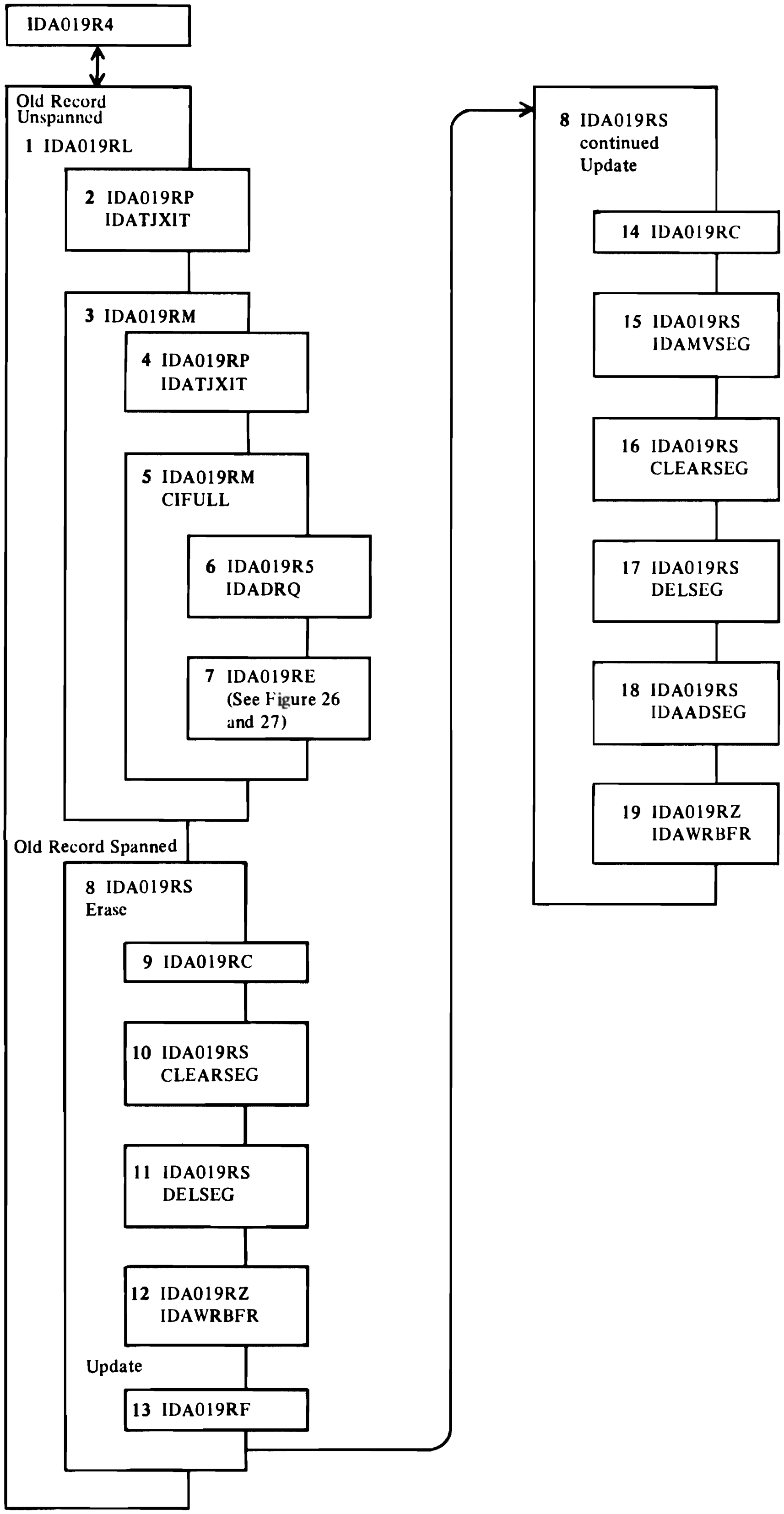


Figure 25. Update/Erase Processing (ESDS, KSDS)

Notes for Figure 25

1 IDA019RL removes an unspanned record from a control interval (ERASE), updates a previously read unspanned record if its length doesn't change (PUT-update), or, if an updated record's length is different, erases the record's contents in the control interval and calls IDA019RM to insert the record into the control interval (PUT-update).

2 If a record was erased, IDATJXIT provides the necessary journaling information for the user's journal exit routine.

3 If the record is a different size, IDA019RM inserts it.

4 If the control interval must be split (the new information is greater than the amount of free space in the control interval), the control interval's original records (before the split) are journalled. IDATJXIT provides the necessary journaling information for the user's journal exit routine. data buffer.

5 CIFUL processes the control interval when it is full and its contents is split (put into two control intervals).

6 The control-interval-split process requires the exclusive use of the DIWA control block. If another request is using the DIWA, IDADRQ waits until the DIWA is available.

7 IDA019RE splits the control interval.

See Figure 26 when the control interval is split during data set creation or during entry-sequenced data set processing.

See Figure 27 when the control interval is split during key-sequenced data set processing after the data set is created.

8 IDA019RS erases or updates a spanned record.

9 IDA019RC locates the record's entry in the sequence set.

10 CLEARSEG gets a buffer, clears it to free space, and writes it to auxiliary storage.

11 DELSEG removes a segment's entry from the sequence set.

12 IDAWRBFR writes the updated sequence-set record.

13 IDA019RF splits the control area if the updated record has additional segments for which free control intervals aren't available in the control area.

14 IDA019RC locates the record's entry in the sequence set.

15 IDAMVSEG moves a segment from the user's area to a buffer.

16 CLEARSEG clears to free space the control intervals occupied by segments removed from an updated record.

17 DELSEG removes a segment's entry from the sequence set when the updated record has fewer segments than the original record.

18 IDAADSEG builds entries in the sequence set for additional segments in the updated record.

19 IDAWRBFR writes the updated sequence-set record.

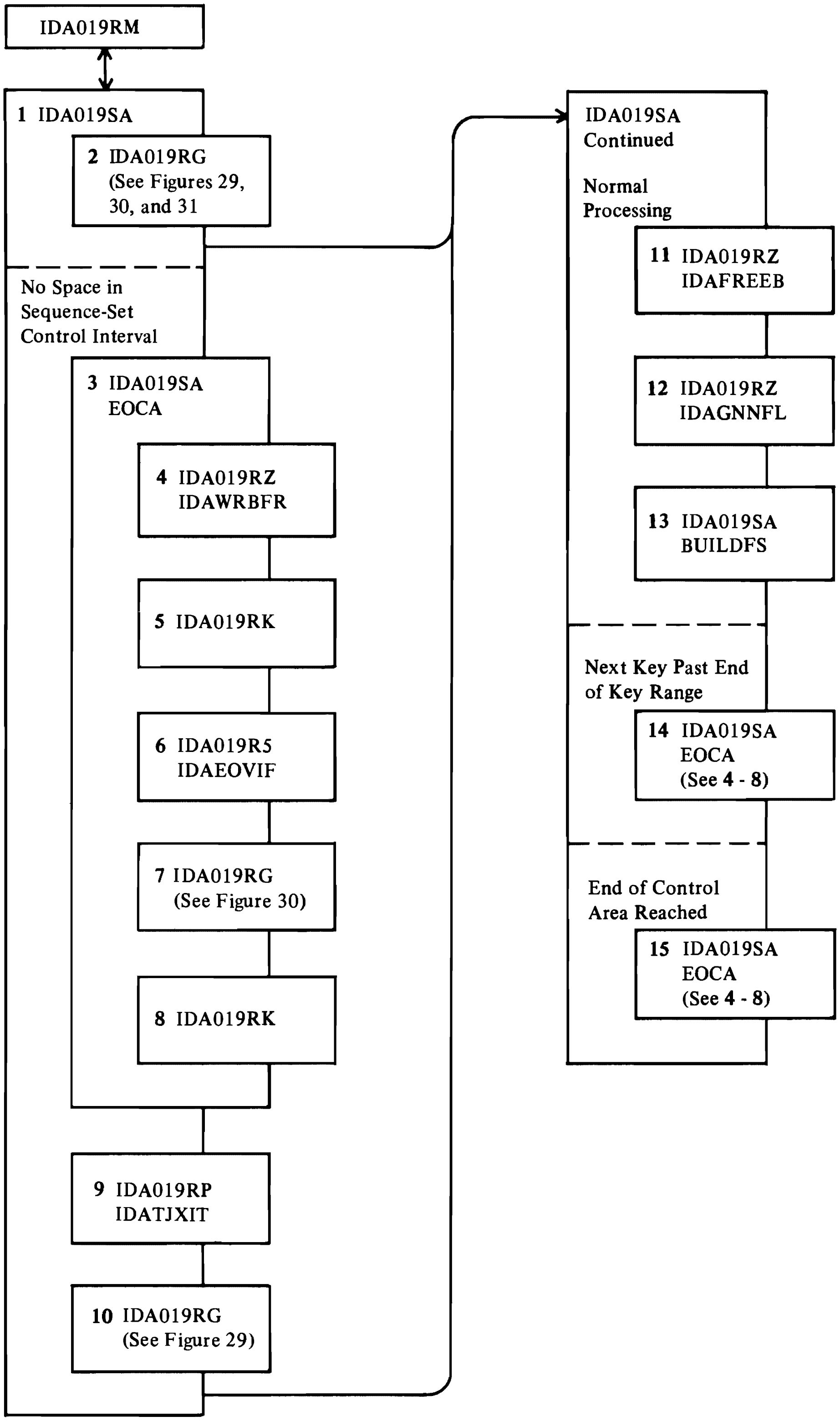

IDA019RM
1 IDA019SA
2 IDA019RG (See Figures 29, 30, and 31
No Space in Sequence-Set Control Interval
3 IDA019SA EOCA
4 IDA019RZ IDAWRBFR
5 IDA019RK
6 IDA019R5 IDAEOVIF
7 IDA019RG (See Figure 30)
8 IDA019RK
9 IDA019RP IDATJXIT
10 IDA019RG (See Figure 29)
IDA019SA Continued
Normal Processing
11 IDA019RZ IDAFREEB
12 IDA019RZ IDAGNNFL
13 IDA019SA BUILDFS
Next Key Past End of Key Range
14 IDA019SA EOCA (See 4 - 8)
End of Control Area Reached
15 IDA019SA EOCA (See 4 - 8)

Figure 26. Obtain the Next Control Interval: Create Processing and Entry-Sequenced Data Set Processing

Notes for Figure 26

1 IDA019SA obtains the next (sequential) control interval to contain the data records. IDA019SA selects this path when the data set (key-sequenced of entry-sequenced) is being created, or when an entry-sequenced data set is being processed.

2 IDA019RG builds an index entry (in the sequence-set control interval) for the full data control interval.

3 VSAM is designed so that the sequence set control interval can contain an index entry for each data control interval in a control area. Sometimes, when the keys are very long, the sequence-set control interval is filled even though some of the control intervals haven't been loaded with data records yet. When this occurs, IDA019RG (at **2**) returns a condition code indicating that the index entry for the full data control interval hasn't been built. EOCA writes each unused control interval in the control area (associated with the full sequence-set control interval) as a free control interval. EOCA then writes the full data control interval into the first control interval of the next control area.

4 IDAWRBFR writes the full buffer containing the data control interval into the data set (the first control interval of the next control area).

5 If the caller is creating the data set and specified the "speed option", the unused control intervals in the control area have not been preformatted. IDA019RK preformats them—rewrites them as free-space control intervals.

6 If the data set (or key range, if this describes step **14**'s EOCA) is out of space, IDAEOVIF calls End of Volume to obtain another secondary space allocation for the data set (or key range).

7 IDA019RG writes the full sequence set control interval into the index.

8 If the caller specified the "recovery option", IDA019RK preformats the next control area's control intervals.

9 IDATJXIT provides journal information about the data that is going into the new control area for the user's journal exit routine.

10 IDA019RG builds an index entry to describe the first control interval in the new control area and puts it into the new control area's sequence set control interval.

11 IDAFREEB frees the buffer that contains the full data control interval.

12 IDAGNNFL obtains an empty buffer to continue the caller's data set create proccessing.

13 BUILDFS initializes the buffer as a free-space control interval.

14 When the caller's key-sequenced data set is divided into key ranges and the key of the record being added is greater than the high key of the key range, EOCA writes the buffer containing the control area's last record into the control area. EOCA then writes each unused control interval in the control area as a free-space control interval. EOCA determines the RBA of the next key-range's first control area and writes the record into the new control interval.

15 When the caller's new record exceeds the capacity of the last control interval in the control area, EOCA determines the next control area and performs necessary processing to allow the caller to continue data set create processing.

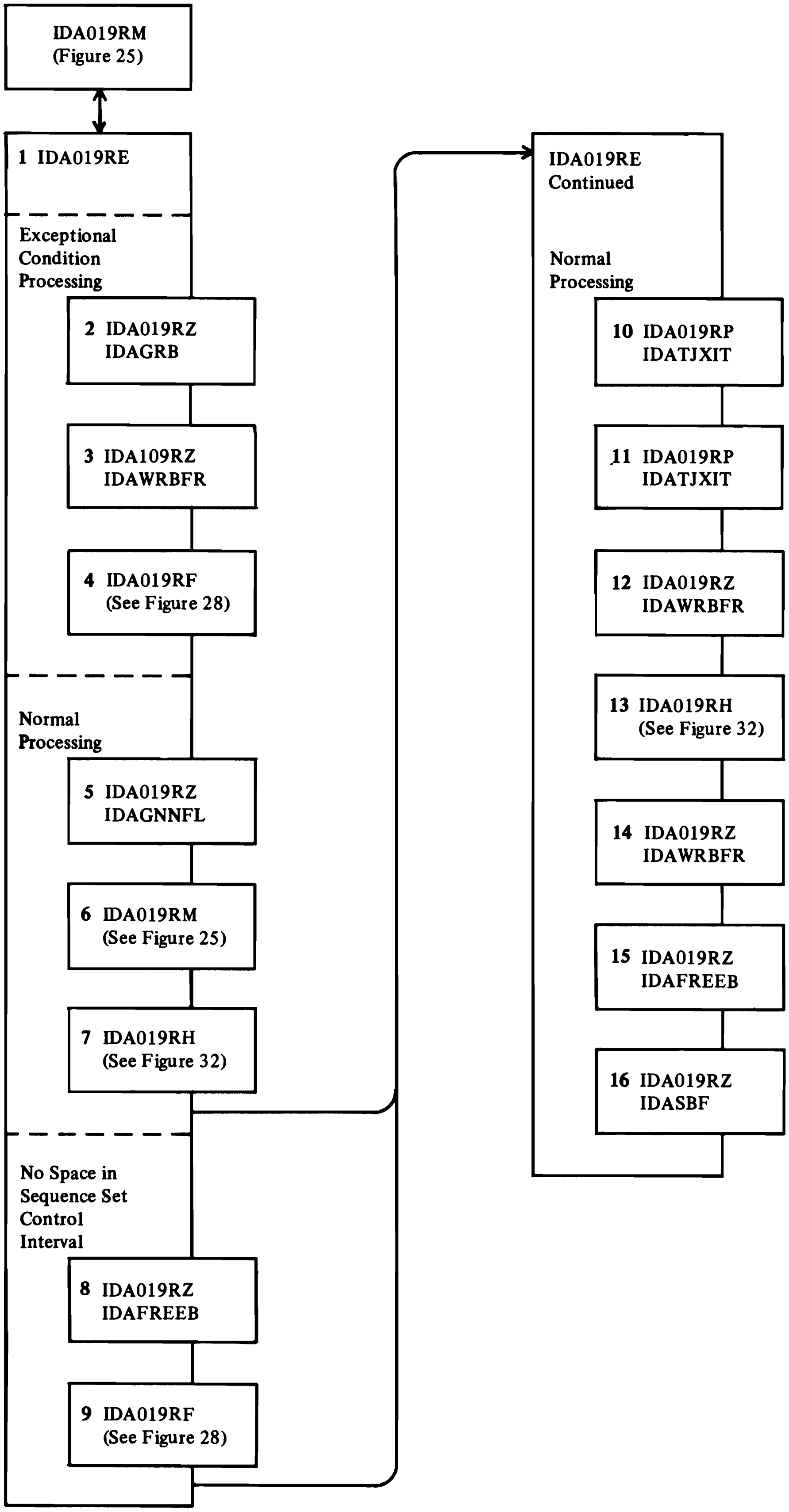


Figure 27. Split a Control Interval: Key-Sequenced Data Set, NonCreate-Time Processing

Notes for Figure 27

1 IDA019RE divides a control interval's records between the control interval and a free-space control interval.

2 If the sequence set record associated with the control interval has been modified by some other request, IDAGRB obtains a current copy of the sequence set control interval in a buffer.

3 If the data control interval has been modified by another request before it has been written back to the data set, IDAWRBFR writes the updated control interval into the data set.

4 If the control interval's control area doesn't contain a free-space control interval, IDA019RF splits the control area.

5 IDAGNNFL obtains an empty buffer. The buffer is used to build the new data control interval, using records from the control interval being split.

IDA019RE distributes the records between the current control interval (being split) and the new control interval (in the newly obtained buffer).

6 IDA019RM inserts the data record (the record that wouldn't fit and caused the control interval split) into the control interval.

7 IDA019RH builds an index entry for the new control interval. IDA019RH also puts the entry in the sequence set control interval associated with the control area.

8 If the entry won't fit in the sequence set control interval, IDA019RE forces a control area split. IDAFREEB frees the buffer that was obtained to contain the new data control interval.

9 IDA019RF splits the control area.

10 If the user's exit list contains an active journal exit address, IDATJXIT provides journaling information about the control area split and the data records that were moved from one control interval to another.

11 If the user's exit list contains an active journal exit address, IDATJXIT provides journaling information about the data records that were moved within the control interval to allow the new data record to be inserted.

12 IDAWRBFR writes the new control interval into the data set. Of the two control intervals that resulted from the control interval split, this control interval contains the records with the highest keys.

13 IDA019RH writes the updated sequence set record (from set **17**).

14 IDAWRBFR writes the updated (old) control interval into the data set.

15 IDAFREEB frees the buffer obtained during **5**. IDA019RE repositions the sequence set pointers to point to the data control interval into which the insert was made

16 IDASBF releases all other buffers associated with the placeholder (PLH).

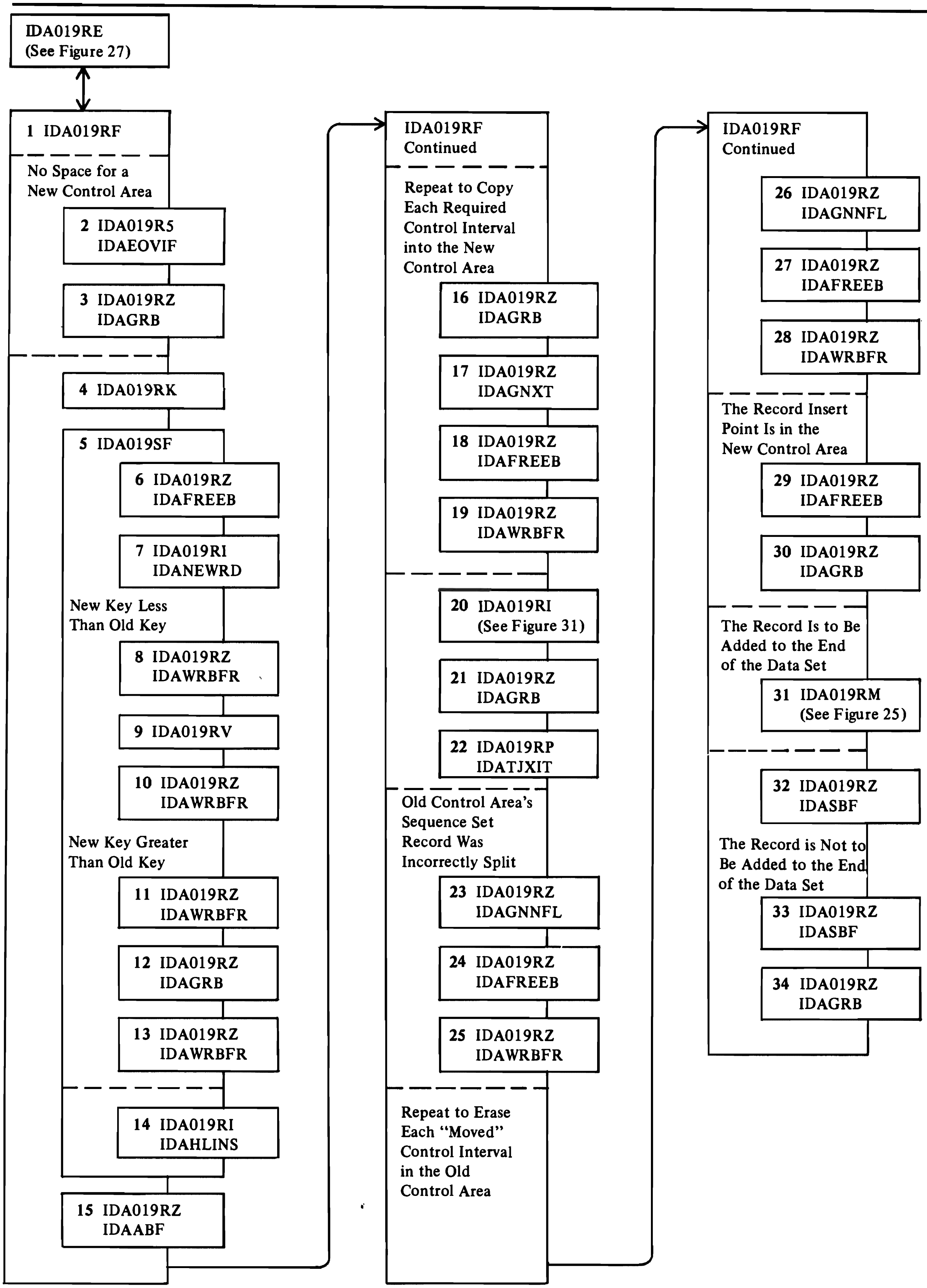


Figure 28. Split a Control Area

Notes for Figure 28

1 IDA019RF moves some of a control area's control intervals into a free-space control area.

2 If a free-space control area is not available, IDAEOVIF calls End of Volume to obtain more space for the data set.

3 IDAGRB obtains a current copy of the control area's sequence set record.

4 IDA019RK preformat's the free-space control area.

5 If the control area can't be split because it is filled with a single spanned record, IDA019SF builds a new sequence-set record and clears a data buffer to free space:

6 IDAFREED frees the current sequence-set buffer.

7 IDANEWRD initializes a new sequence-set record.

New Key Less Than Old Key

(The "old" key is the key of the spanned record that fills the control area.)

8 IDAWRBFR writes the new sequence-set record.

9 IDA019RV obtains the sequence-set record that precedes the sequence-set record of the control area filled with the spanned record. IDA019RV changes the sequence-set record's horizontal pointer to point to the new sequence-set record (step 7).

10 IDAWRBFR writes the sequence-set record (step 9).

New Key Greater Than Old Key

11 IDAWRBFR writes the new sequence-set record.

12 IDAGRB reads the sequence-set record of the control area filled with the spanned record and changes its horizontal pointer to point to the new sequence-set record.

13 IDAWRBFR writes the sequence-set record (step 12).

14 IDAHLINS changes the second-level index record to point to the new sequence-set record.

15 IDAABF obtains as many buffers as possible to allow the Control-Area-Split routine to function as smoothly as possible. The maximum number of buffers obtained is equal to the number of control intervals to be moved into the new control area. The buffers are used to copy control intervals from the old control area and rewrite them into the new control area.

16 IDAGRB obtains a copy of the first control interval that is to be copied into the new control area.

17 When this sequence is repeated for subsequent control intervals, IDAGNXT obtains the next sequential data control interval in the control area until all control intervals that are to be moved have been processed.

IDA019RF modifies the output RBA value in the control interval buffer's BUFC, so that the control interval is written into the new control area.

18 IDAFREEB frees the buffer. The buffer's contents will be written into the new control area; when it is used agáin to contain another control interval.

17 and **18** are repeated for each control interval in the old control area that is moved into the new control area.

19 IDAWRBFR writes all buffers not yet written into the new control control area.

20 IDA019RI builds a new sequence set record for the control area and adjusts other higher-level index records to point to the new sequence set record.

21 IDAGRB obtains a current copy of the old control area's sequence set record.

22 If the user's exit list contains an active journal exit, IDATJXIT provides journaling information about the control interval being moved—its old and new RBAs.

If the sequence set record could not be split at the point the data was split, some control intervals in the new control area are removed from the new control area so that both old and new seqeunce set records are accurate. These control interval's are rewritten as free-space control intervals in the new control area; they remain intact in the old control area. **23** through **25** process this exceptional condition.

23 IDAGNNFL obtains an empty buffer. IDA019RF builds a free-space control interval in it.

24 IDAFREEB frees the buffer, so that it will update the control area with a free-space control interval when the buffer is used next.

25 IDAWRBFR writes all buffers not yet written into the new control area.

26 IDAGNNFL obtains an empty buffer. IDA019RF builds a free-space control interval in it. The free-space control interval replaces each control interval in the old control area that has been copied into the new control area.

27 IDAFREEB frees the buffer, so that it will update the old control area with a free-space control interval when the buffer is used next. **26** and **27** are repeated until all control intervals in the old control area that have been copied are deleted.

28 IDAWRBFR writes all buffers not yet written into the old control area.

29 If the insert point for the record to be added to the data set is in the new control interval, IDAFREEB frees the buffer that contains the old sequence set control interval.

30 IDAGRB obtains a copy of the sequence set control interval associated with the new control area.

31 If the record is to be inserted at the end of the data set, IDA019RM inserts the record. No further control area split processing is performed.

32 If the record is not to be added to the end of the data set, IDASBF releases all buffers associated with the placeholder, except the buffers contained the data record's insert point and the sequence set control interval.

33 Same as **32.**

34 IDAGRB obtains a current copy of the data control interval that contains the data record's insert point.

IDA019RF returns to the control interval split routine to split the control interval and insert the data record.

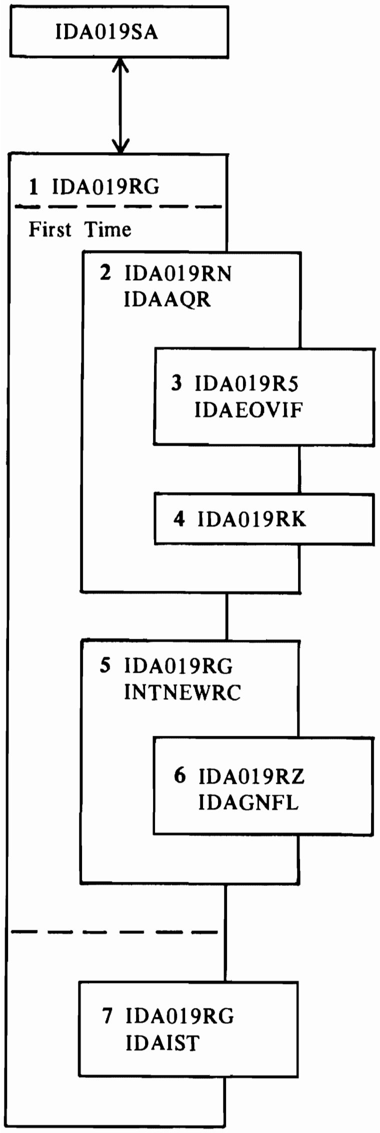


Figure 29. Create-Time Sequence-Set Record Processing: Build an Entry

Notes for Figure 29

IDA019RG is called by IDA019SA when a key-sequenced data set is being created.

1 This figure describes the addition of an index entry to the sequence-set control interval when a data control interval is full.

2 If IDA019RG is being called for the first time, IDAAQR obtains a control interval for a sequence-set record.

3 If all allocated space in the data set has been used, IDAEOVIF obtains another extent for the data set.

4 If the newly obtained extent must be preformatted before it can be used, IDA019RK preformats it.

5 INTNEWRC initializes the control interval as a sequence-set control interval.

6 IDAGNFL obtains a buffer for the sequence-set control interval.

7 IDAIST uses the high key value of the data control interval to build an index entry in the sequence-set control interval. The key is front and rear compressed before the entry is built.

If there is not enough room to insert the index entry in the sequence-set control interval, IDA019RG indicates this and returns to IDA019SA. The entry is not put in the sequence-set record.

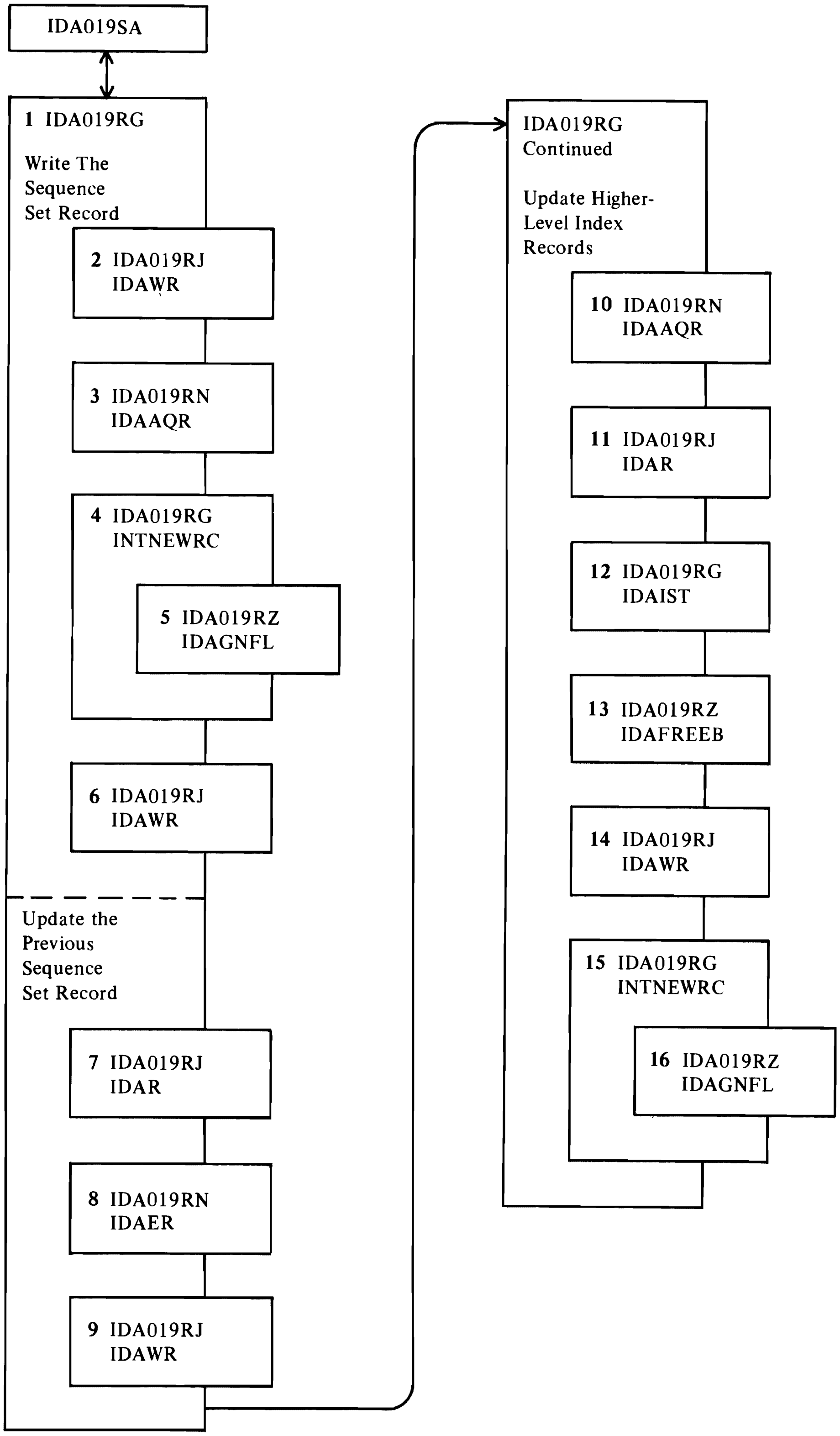


Figure 30. Create-Time Sequence-Set Record Processing: Write the Record (End of Control Area)

Notes for Figure 30

1 When the control area is full, IDA019RG writes its sequence-set record into the index and initializes a new sequence-set record for the new control area.

2 IDAWR writes the updated sequence-set record into the index. This is the sequence-set record associated with the old control area.

3 IDAAQR obtains the next control interval for a sequence-set record.

If all allocated space in the data set has been used, IDAAQR calls IDAEOVIF to obtain another extent for the data set.

If the newly obtained extent must be preformatted before it can be used, IDAAQR calls IDA019RK to preformat it.

4 INTNEWRC initializes the control interval as a sequence-set record.

5 IDAGNFL obtains a buffer for the sequence-set record.

6 IDAWR writes the new sequence-set record with a dummy index entry—an entry with length=0 and front-key compression=0.

7 Read obtains a copy of the previously written (from **2**) sequence-set record.

IDA019RG builds a horizontal pointer entry to allow the record to point to the newly created sequence-set record.

8 IDAER removes the dummy entry from the sequence-set record.

9 IDAWR writes the updated (previous, from step **7**) sequence-set record into the index. The sequence-set record now has the "proper" ending entry.

IDA019RG adjusts the higher-level index records to reflect the addition of a new sequence-set record.

10 When a higher-level index record is required, IDAAQR locates the control interval containing it.

If all allocated space in the data set has been used, IDAAQR calls IDAEOVIF to obtain another extent for the data set.

If the newly obtained extent must be preformatted before it can be used, IDAAQR calls IDA019RK to preformat it.

IDA019RG obtains more virtual storage (using GETMAIN) for another ICWA, if all other ICWAs are being used, and initializes it.

11 IDAR reads in the higher level index record.

12 IDAIST builds an index entry to describe the sequence-set index record and puts it into the higher level index record.

13 If the entry won't fit in the higher level record, IDAFREEB frees the buffer containing the higher level index record (from **11**).

14 IDAWR writes out the updated higher level index record, so that the index is always as current as possible. Steps **10** through **14** are repeated to update as many levels of the index as are required.

15 INTNEWRC initializes a buffer for the new sequence-set index record. index record.

16 IDAGNFL obtains an empty buffer for the new sequence-set index record.

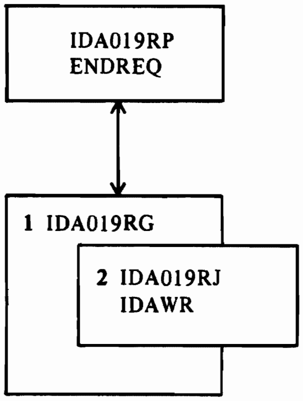


Figure 31. Create-Time Sequence-Set Record Processing: Write the Record (Closing the Data Set)

Notes for Figure 31

1 When the user closes the data set after he creates it, IDA019RG writes the last sequence-set record into the index.

2 IDAWR writes the sequence-set record into the index.

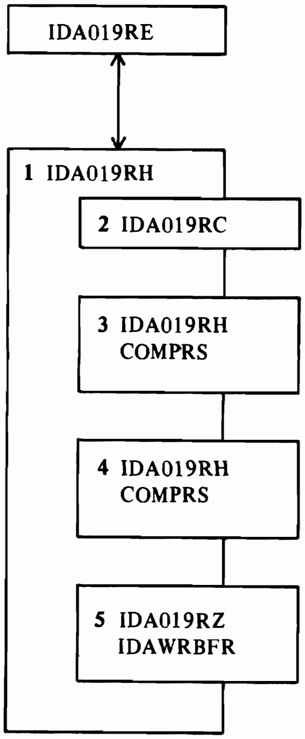


Figure 32. NonCreate-Time Sequence-Set Processing

Notes for Figure 32

1 IDA019RH builds an index entry and inserts it in the proper position in the sequence-set record when a control interval is split.

2 IDA019RC searches the compressed index entries in the sequence-set record to locate the insert point for the new index entry.

3 COMPRS performs rear key compression for the newly built index entry.

4 COMPRS modifies the front and rear key compression of index entries in the sequence-set record that might require modification as a result of inserting a new compressed key entry.

5 IDAWRBFR writes the updated sequence-set record into the sequence set.

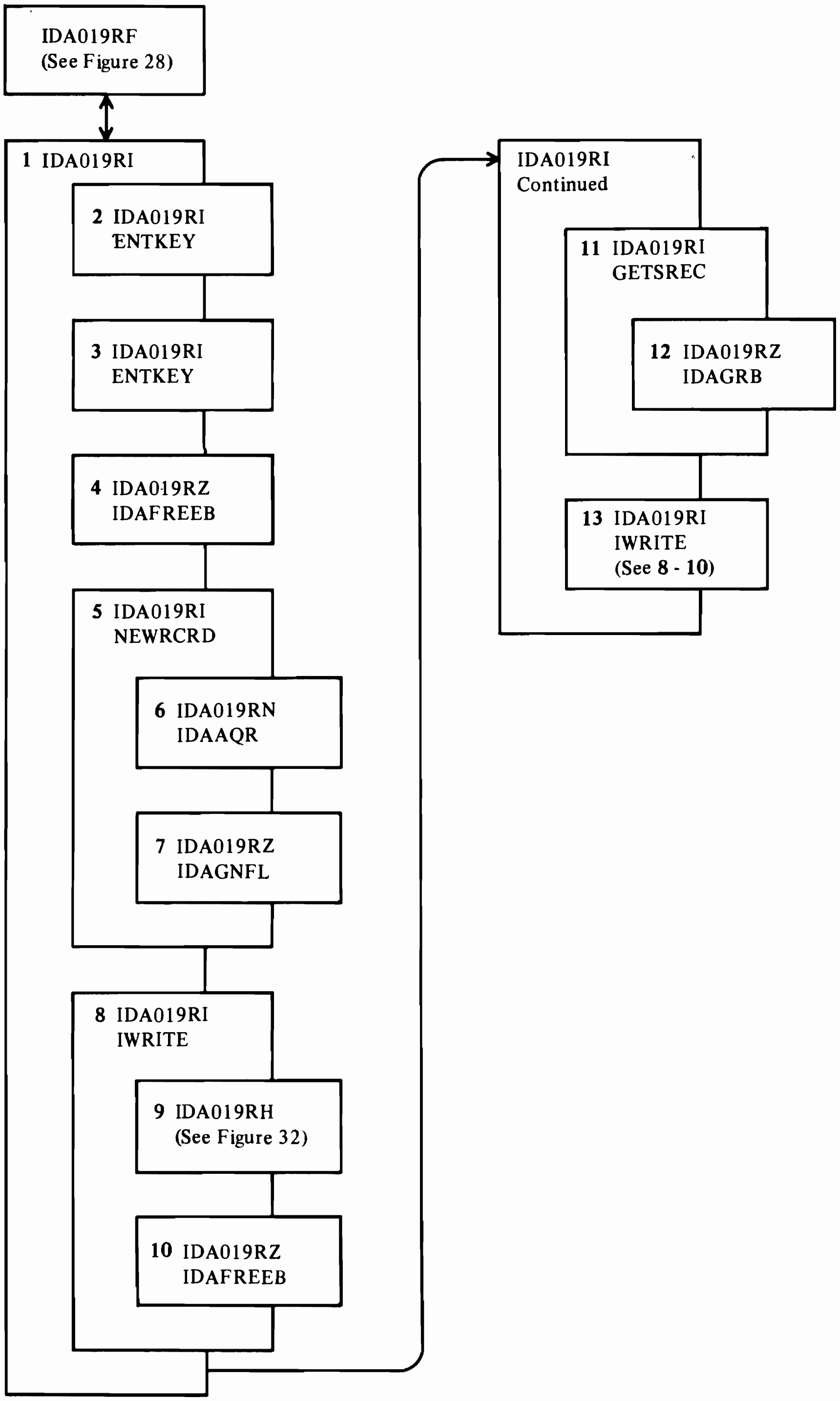


Figure 33. Update the Index: Adding to the End of a Key Range or Data Set

Notes for Figure 33

1 IDA019RI updates higher level index records when a control area is split. If the control area being split is at the end of a key range or data set, this figure describes the updating sequence.

2 ENTKEY locates and extracts the next to last section entry from the index record.

3 ENTKEY extracts the last section entry from the index record.

4 IDAFREEB frees the current index record.

5 NEWRCRD builds and initializes a new index record.

6 IDAAQR obtains a RBA value for the new index record.

If all allocated space in the data set has been used, IDAAQR calls IDAEOVIF to obtain another extent for the data set.

If the newly obtained extent must be preformatted before it can be used, IDAAQR calls IDA019RK to preformat it.

7 IDAGNFL obtains an empty index buffer for the new index record. When the record is built, it will be written into the index at the RBA obtained by IDAAQR.

NEWRCRD builds the new index record.

8 IWRITE writes the new index record into. the index.

9 IDA019RH writes the index record.

10 IDAFREEB frees the index record's buffer.

11 GETSREC obtains the previous sequence-set record.

12 IDAGRB retrieves the newly written index record.

GETSREC adjusts the index record, removing the last key entry from the record.

13 IWRITE rewrites the updated index record into the index.

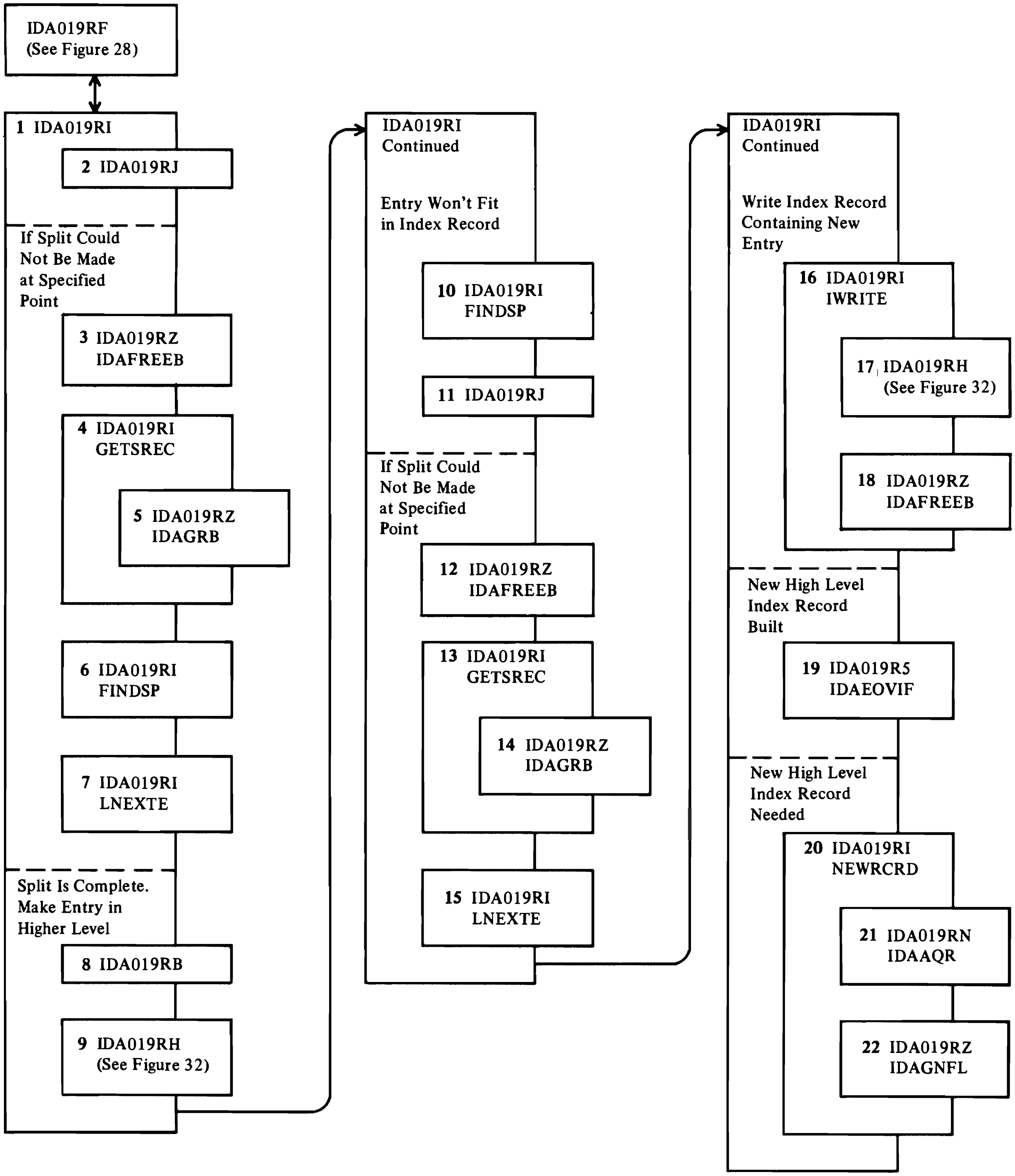


Figure 34. Update the Index: Splitting a Control Area (Not at the End of a Key Range or Data Set)

Notes for Figure 34

1 IDA019RI updates the higher level index records when a control area is split. If the control area being split is not at the end of a key range or data set, this figure describes the updating sequence.

2 IDA019RJ splits the current sequence-set record.

If the sequence-set record could not be split at the specified point, **3** through **7** adjust the split point so that it can be split.

3 IDAFREEB frees the index buffer.

4 GETSREC obtains the sequence-set record from the index (IDA019RJ destroyed the old copy during its processing.)

5 IDAGRB retrieves the sequence-set record.

6 FINDSP scans the sequence-set record to locate the split point.

7 LNEXTE adjusts the split point by one entry. **2** is retried, and **3** through **7** repeat, until the seqeunce-set record is split.

8 IDA019RB searches the index to locate the insert point in the next higher level of the index.

9 IDA019RH inserts the new entry in the higher level index record.

If the entry doesn't fit in the higher level index record, steps **10** and **11** attempt to split it.

10 FINDSP locates the midpoint of the index record entries in the higher level index record.

11 IDA019RJ splits the index record. If the split could not be made at the specified point, **12** through **15** adjust the split point so that the record can be split.

12 IDAFREEB frees the index record's buffer.

13 GETSREC obtains the higher level index record from the index (IDA019RJ destroyed the copy in the buffer during its processing).

14 IDAGRB retrieves the index record.

15 LNEXTE adjusts the split point by one entry. **11** is retried, and **12** through **15** repeat, until the index record is split. When the split is correct, **8** and **9** insert the entry that would not fit before.

16 IWRITE writes the index record containing the new entry into the index.

17 IDA019RH writes the index record.

18 IDAFREEB frees the index record's buffer.

19 If a new high-level index record was built by this index upgrading processing, IDAEOVIF updates the catalog information for the index.

20 If a new high-level index record is needed, NEWRCRD obtains a RBA and buffer for the record. NEWRCRD builds the new record and does **16** through **19** to write the record and adjust the index's catalog information.

21 IDAAQR obtains a RBA value for the new high-level index record.

If all allocated space in the data set has been used, IDAAQR calls IDAEOVIF to obtain another extent for the data set.

If the newly obtained extent must be preformatted before it can be used, IDAAQR calls IDA019RK to preformat it.

22 IDAGNFL obtains an empty index buffer for the new index record. When the record is built, it will be written into the index at the RBA obtained by IDAAQR.

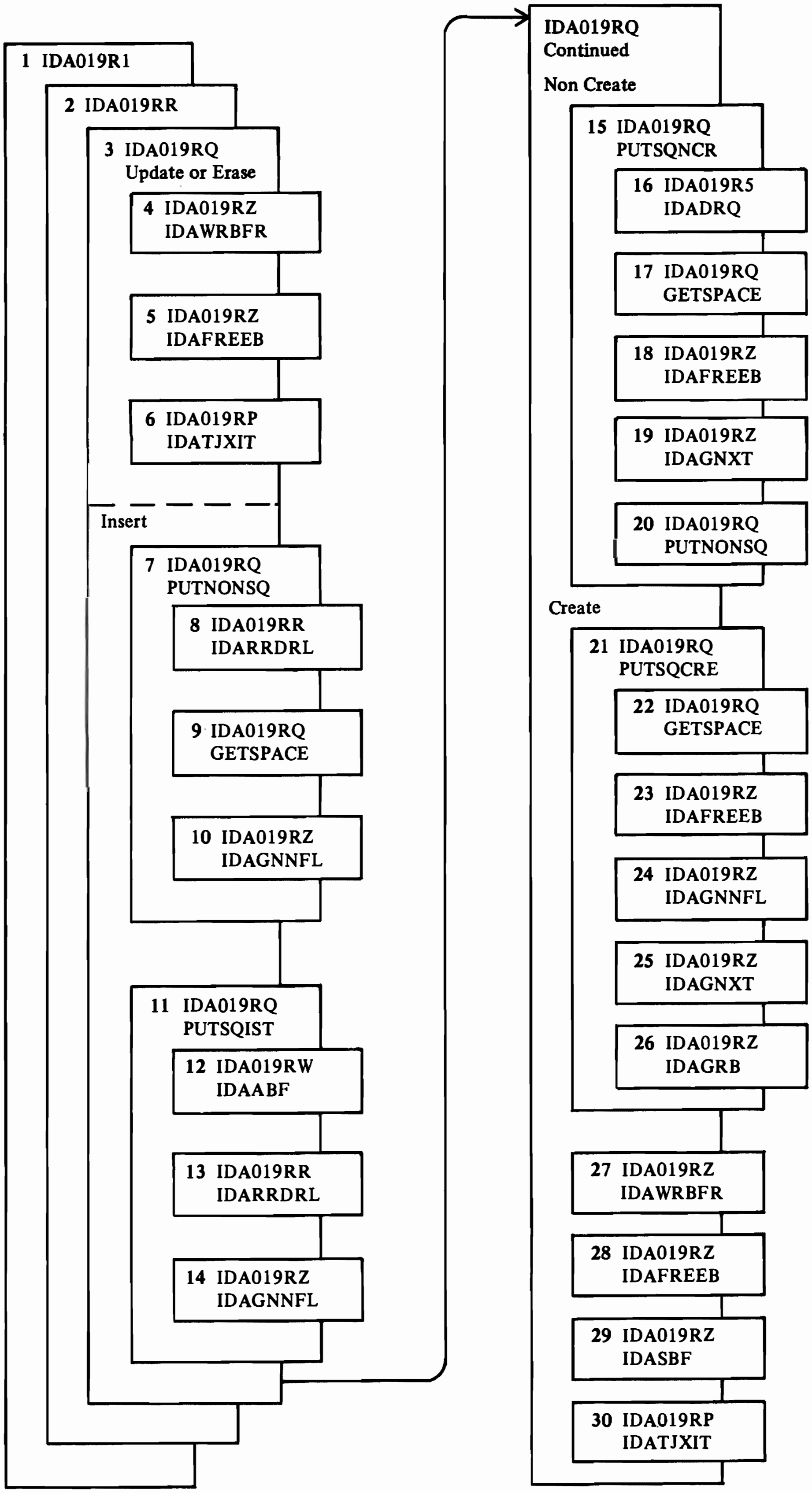


Figure 35. PUT/ERASE Processing (RRDS)

Notes for Figure 35

1 IDA019R1 is the common Record Management request module. It verifies that the request is valid and checks for keyed processing of a relative record data set.

2 IDA019RR selects the processing path for GET, PUT, POINT, or ERASE and for direct, sequential, or skip sequential access.

Update or Erase

PUT-update or ERASE requires that a GET-update was previously issued. Therefore, the control interval that contains the record to be updated or deleted is in the data buffer, and the PLH points to the record.

3 For PUT-update, IDA019RQ lays the updated record over the old record. For ERASE, IDA019RQ fills the slot with binary zeros and changes the RDF to indicate an empty slot.

4 For a direct request that is not to have string position noted, IDAWRBFR writes the data buffer to the control interval.

5 IDAFREEB frees the data buffer.

6 If the user's EXLST contains an active journal exit address, IDATJXIT provides the necessary journaling information for the user's journal exit routine.

Insert

The slot indicated by the search argument or by current positioning must be empty. If it isn't, the record to be inserted isn't inserted, because of duplicate record.

7 PUTNONSQ locates the control interval for a direct or skip sequential request. The search argument (relative record number) is converted to the RBA of the control interval that contains it and the offset of the record in the control interval.

8 For skip sequential access, IDARRDRL verifies that the search argument is greater than the previous one, indicated by positioning. It retrieves the control interval by RBA and sets the PLH pointer to the indicated slot.

9 If the indicated relative record number is in a control interval beyond the last control interval currently in the data set, GETSPACE calls IDA019RK to preformat the next control area. If processing is for creation (the data set was empty when opened) with the SPEED option, the rest of the control intervals in the current control area are preformatted before a new control area is preformatted. Control intervals are preformatted until the one that contains the indicated relative record number has been preformatted. GETSPACE calls IDAEOVIF when additonal space is needed for control areas.

10 To insert the record into a slot in a control interval not currently in the data set, no control interval is read. IDAGNNFL gets an empty data buffer and formats it with empty slots.

PUTSQ1ST locates the first control interval of the data set when the first request after OPEN is sequential.

12 IDAABF adds additional buffers to the buffer chain for read-ahead buffering.

13 If processing is not for creation (that is the data set contained formatted control areas when opened), IDARRDRL retrieves the first control interval and sets the PLH pointer to the first slot in the control interval.

14 If processing is for creation, IDAGNNFL gets an empty data buffer and formats it with empty slots.

NonCreate

15 PUTSQNCR processes sequential requests when processing is not for creation. If the previous request was POINT with KGE (key greater than or equal), the control interval identified by the search argument of the POINT is retrieved. Otherwise PUTSQNCR advances the PLH pointer to the next slot. If there are no more slots in the control interval, the next control interval is retrieved.

16 When additional space is allocated, IDADRQ gets exclusive use of the data set for extension.

17 When the next control interval is in the next control area, GETSPACE calls IDA019RK to preformat the next control area. If additional space is needed for the next control area, GETSPACE calls IDAEOVIF to allocate the space and preformat the first control area in it.

18 When there are no more slots in the current control interval, IDAFREEB frees the current data buffer.

19 IDAGNXT retrieves the next sequential control interval.

20 If the previous request was POINT with KGE, PUTNONSQ retrieves the control interval identified by the search argument of the POINT.

Create

21 PUTSQCRE processes sequential requests when processing is for creation. PUTSQCRE advances the PLH pointer to the next slot in the current data buffer.

22 When the next control interval is in the next control area, GETSPACE calls IDA019RK to preformat the next control area. If additional space is needed for the next control area, GETSPACE calls IDAEOVIF to allocate the space. Unless the SPEED option is indicated, IDAEOVIF preformats the first control area in the newly allocated space.

23 When there are no more slots in the current control interval, IDAFREEB frees the current data buffer.

24 When the next control interval hasn't been preformatted, IDAGNNFL gets an empty data buffer and formats it with empty slots.

25 When the next control interval has been preformatted and the RECOVERY option is indicated, IDAGNXT retrieves the next control interval and puts it in the data buffer.

26 When the next control interval has been preformatted and the SPEED option is indicated, IDAGRB retrieves the next control interval by RBA and puts it in the insert buffer. Using the insert buffer causes an update-write channel program to be used when the control interval is written.

IDA019RQ moves the record to be inserted into its slot, unless the slot already contains a record. The record to be inserted is considered a duplicate.

27 For a direct request that is not to have string position noted, IDAWRBFR writes the data buffer to the control interval.

28 IDAFREEB frees the data buffer.

29 For a direct request that is not to have string position noted, where the current data buffer is the insert buffer, IDASBF writes the insert buffer and removes it from the normal buffer chain.

30 If the user's EXLST contains an active journal exit address, IDATJXIT provides the necessary journaling information for the user's journal exit routine.

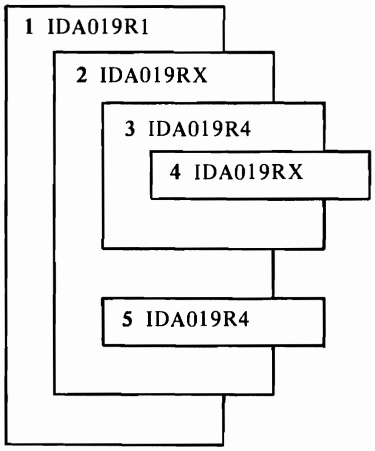


Figure 36. Path Processing

Notes for Figure 36

1 IDA019R1 checks the user's RPL for validity and assigns a PLH to it. It detects a request for access to a base cluster by way of an alternate index.

2 IDA019RX builds an inner RPL to be used in retrieving the alternate-index record needed for the request.

3 IDA019R4 retrieves the alternate-index record needed for the request.

4 If IDA019R4 detected that the user's data area was too small for the alternate-index record, IDA019RX increases the size of the area.

IDA019RX builds an inner RPL to be used for the request for access to the base cluster.

5 IDA019R4 issues the request for access to the base cluster.

IDA019RX transfers any return code from the inner RPL to the user's RPL.

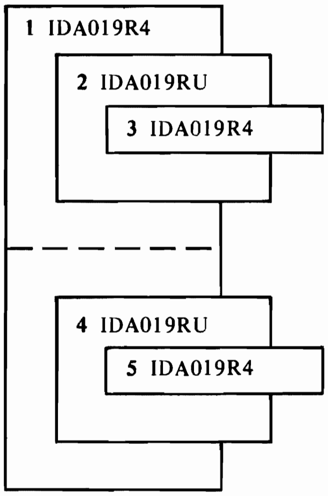


Figure 37. Upgrade Processing

Notes for Figure 37

1 For a PUT or ERASE, when there is an upgrade table (UPT)—which indicates that the base cluster has an upgrade set, IDA019R4 calls IDA019RU for upgrade processing.

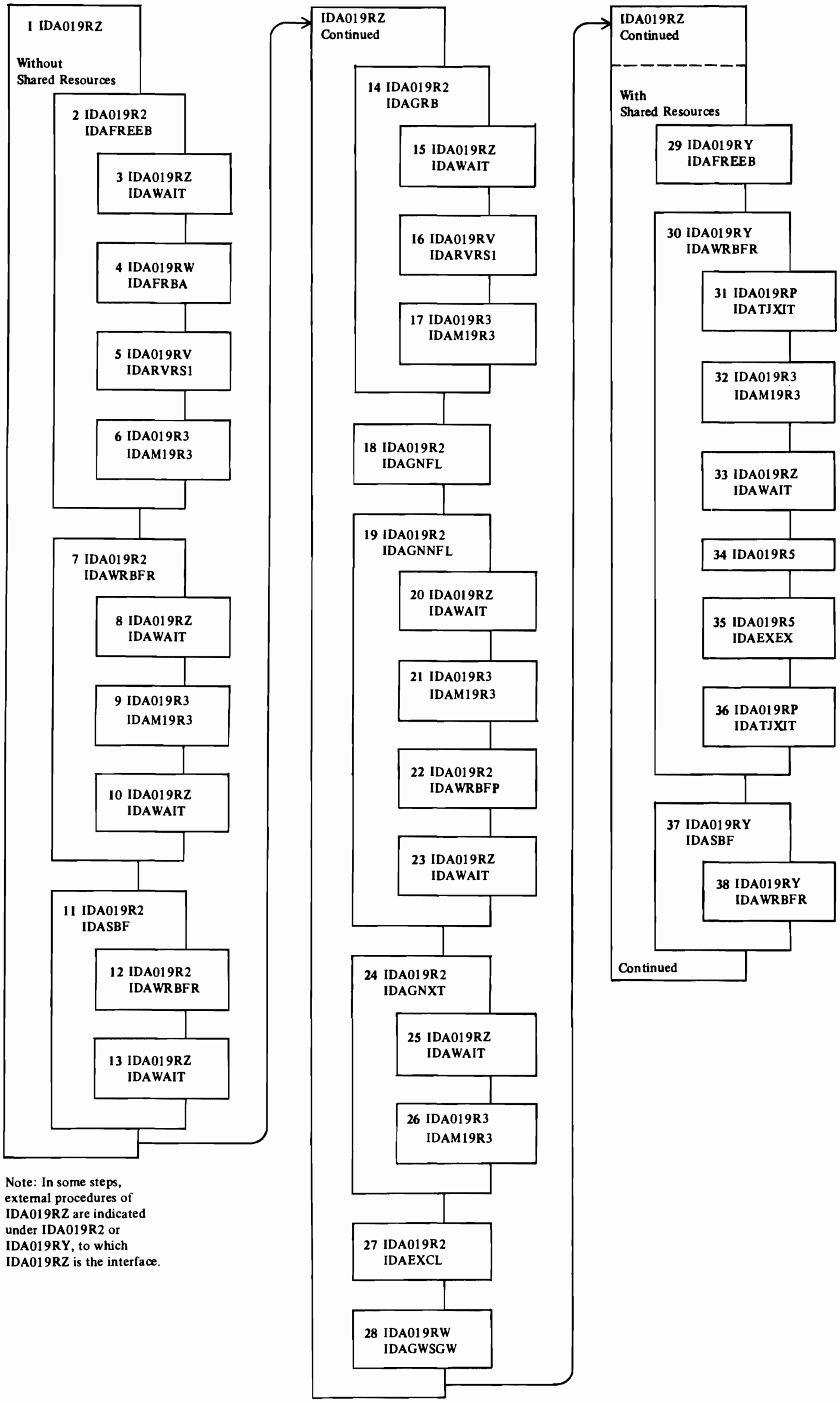
2 For each alternate index in the upgrade set, IDA019RU determines whether the PUT or ERASE requires an alternate-index record or a pointer in an alternate-index record to be added or removed.

3 For each alternate index that requires upgrading, IDA019R4 does the I/O to accomplish upgrading.

If each alternate index was upgraded successfully, IDA019R4 does the I/O for the PUT or ERASE.

4 If the I/O for the PUT or ERASE failed, IDA019RU backs out (undoes) the upgrading for each alternate index.

5 For each alternate index whose upgrading was backed out, IDA019R4 does the I/O to accomplish backing out.



Note: In some steps, external procedures of IDA019RZ are indicated under IDA019R2 or IDA019RY, to which IDA019RZ is the interface.

Figure 38. (Part 1 of 3) Buffer Management

Notes for Figure 38 (Part 1 of 3)

1 IDA019RZ is entered for all frequently used Buffer Management functions. It sets a code in a register that indicates the requested function. For requests *without* shared resources specified, it calls IDA019R2; for requests *with* shared resources specified, it calls IDA019RY. Some procedures (such as IDAFREEB) are literally part of IDA019RZ, but their processing actually takes place in IDA019R2 or IDA019RY. (For example, in this figure IDAFREEB is shown as a procedure of both IDA019R2 and IDA019RY.)

Without Shared Resources

2 IDAFREEB makes an index or insert buffer available for reassignment. For sequential retrieval, when IDAFREEB frees a data buffer, it initiates read-ahead buffering if enough free buffers are available for it.

3 For read-ahead buffering, IDAWAIT lets any previously started I/O finish.

4 IDAFRBA determines the RBA of the next control interval.

5 When one or more of the RBAs in the I/O chain are not in ascending sequence, IDARVRS1 puts them in ascending sequence.

6 IDA019R3 (I/O Management) issues I/O for read-ahead buffering.

7 IDAWRBFR writes the buffer (s) in the current I/O chain.

8 IDAWAIT lets any previously started I/O finish.

9 IDA019R3 (I/O Management) issues I/O for the current chain.

10 IDAWAIT lets the I/O started in step 9 finish.

11 IDASBF moves buffer (s) from the I/O chain back to the buffer pool.

12 Before IDASBF moves a buffer back to the buffer pool, IDAWRBFR ensures that no writes are pending against the buffer.

13 IDAWAIT lets any I/O pending against the buffer finish.

14 IDAGRB reads an index or a data control interval.

15 IDAWAIT lets any previously started I/O finish.

16 IDARVRS1 puts in ascending sequence any RBAs in the I/O chain that are out of order.

17 Unless the index or data control interval is already in the buffer pool, IDA019R3 (I/O Management) issues I/O to read it.

18 IDAGNFL supplies a work buffer for index processing or for a control-interval split.

19 IDAGNNFL supplies an empty data buffer for sequential output processing.

20 IDAWAIT lets any previously started I/O finish.

21 When enough buffers are already flagged for output, IDA019R3 (I/O Management) issues I/O to write them.

22 If the current buffer's contents have been modified, IDAWRBFR write it.

23 IDAWAIT lets any I/O pending against the buffer finish.

24 IDAGNXT ensures that the next data control interval has been read and provides a pointer to the buffer that contains it.

25 IDAWAIT lets any pending I/O finish.

26 IDA019R3 (I/O Management) issues I/O to read a buffer that was not read previously because another request had exclusive control of it.

27 IDAEXCL obtains exclusive control of a control interval identified by RBA.

28 IDAGWSGW obtains an empty data buffer from the current I/O chain.

With Shared Resources

29 IDAFREEB makes a buffer available for reassignment.

30 IDAWRBFR writes a buffer.

31 If the user's EXLST contains an active journal exit address, IDATJXIT notifies the journal exit routine of an impending write.

32 IDA019R3 (I/O Management) issues I/O for the write.

33 IDAWAIT lets I/O for the write finish.

34 If an I/O error occurred, IDA019R5 builds an error message.

35 If an I/O error occurred and the AMB contains an exception exit address, IDAEXEX passes control to the exception exit routine.

36 If an I/O error occurred and the user's EXLST contains an active journal exit address, IDATJXIT passes control to the journal exit routine.

37 IDASBF frees the current buffer.

38 If the buffer's contents have been modified, IDAWRBFR writes it.

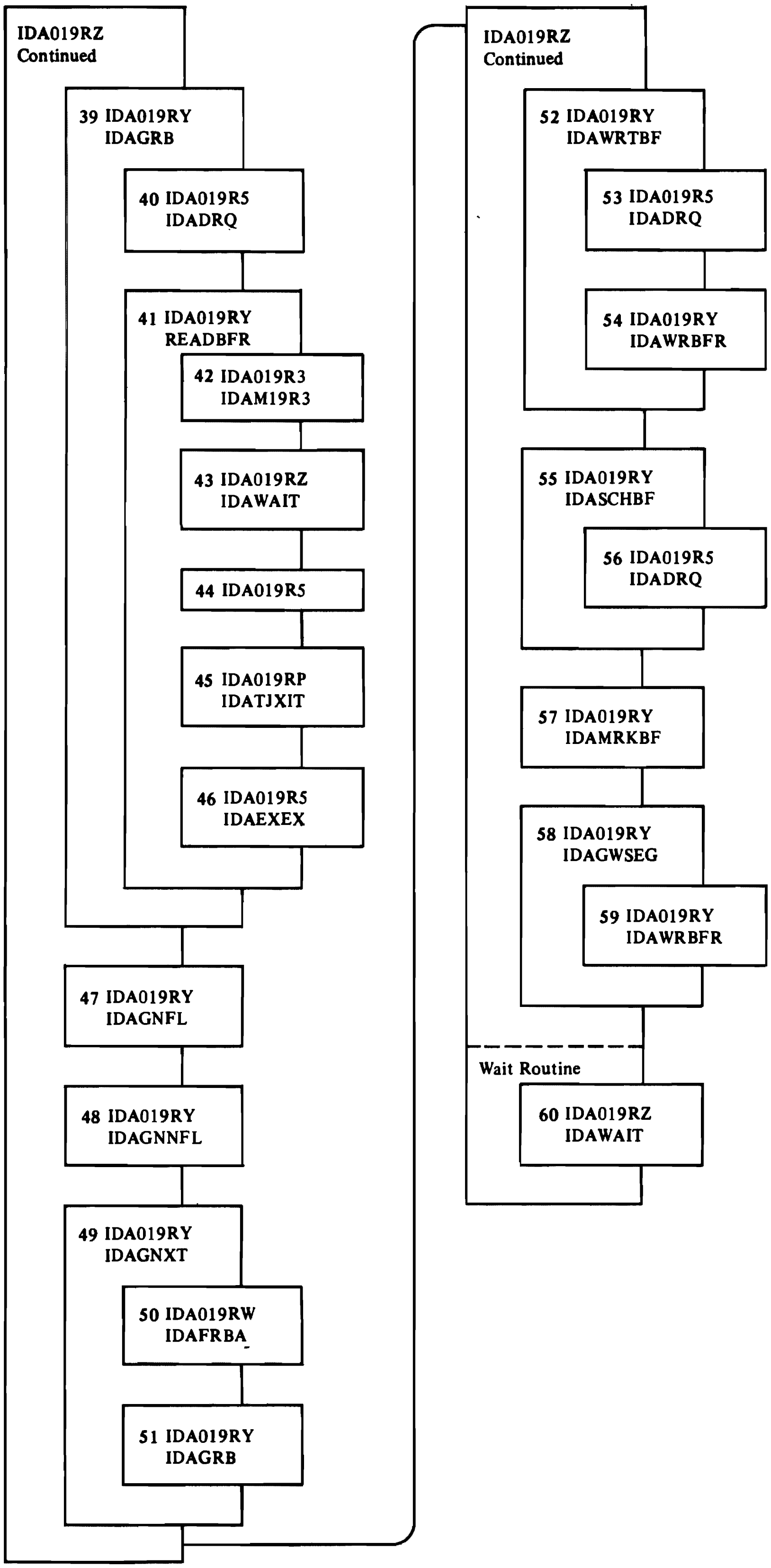


Figure 38. (Part 2 of 3) Buffer Management

Notes for Figure 38 (Part 2 of 3)

39 IDAGRB reads an index or a data control interval.

40 If the buffer is already being read, IDADRQ suspends processing for the current request.

41 Unless the index or data control interval is already in the buffer pool, READBFR reads it.

42 IDA019R3 (I/O Management) issues I/O for the read.

43 IDAWAIT lets the I/O started in step 42 finish.

44 If an I/O error occurred, IDA019R5 builds an error message.

45 If an I/O error occurred and the user's EXLST contains an active journal exit address, IDATJXIT passes control to the journal exit routine.

46 If an I/O error occurred and the AMB contains an exception exit address, IDAEXEX passes control to the exception exit routine.

47 IDAGNFL supplies a work buffer for index processing or for a control-interval split.

48 IDAGNNFL supplies an empty data buffer for sequential output processing.

49 IDAGNXT ensures that the next data control interval has been read and provides a pointer to the buffer that contains it.

50 IDAFRBA determines the RBA of the next control interval.

51 IDAGRB obtains the control interval.

52 IDAWRTBF processes a WRTBFR macro to write the buffer(s) indicated by the caller.

53 If any of the buffers to be written are being used by another request, IDADRQ suspends processing for the current request until the other request makes the buffers available.

54 IDAWRBFR writes the buffers.

55 IDASCHBF processes a SCHBFR macro to search the buffer pool for the RBA indicated by the user.

56 If a buffer contains the indicated RBA but is in the process of having the control interval read into it, IDADRQ suspends processing for the current request until reading is finished.

57 IDAMRKBF processes a MRKBFR macro to mark a buffer to be released or for output.

58 IDAGWSGW obtains an empty data buffer from the current I/O chain.

59 If the buffer's contents have been modified, IDAWRBFR writes it.

60 For a synchronous request, IDAWAIT issues a WAIT macro for the I/O to finish. For an asynchronous request, IDAWAIT sets a flag for the I/O Manager's Asynchronous Routine to pass control to IDAWAIT after the I/O is finished.

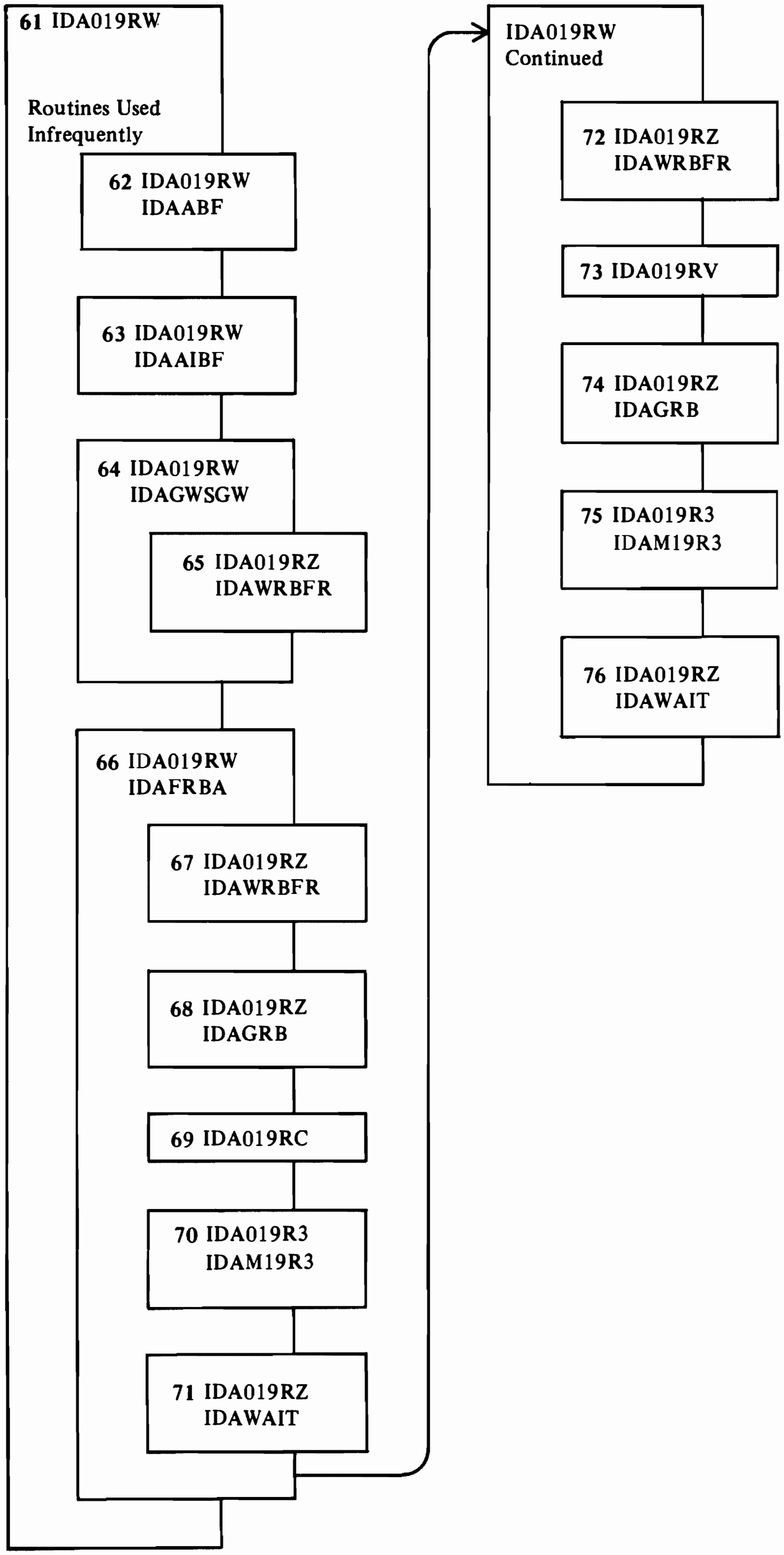


Figure 38. (Part 3 of 3) Buffer Mannagement

Notes for Figure 38 (Part 3 of 3)

61 IDA019RW receives requests for Buffer Management functions that are used only infrequently.

62 For processing without shared resources, IDAABF adds buffers to a string's I/O chain to shorten processing time.

63 For processing without shared resources, IDAAIBF adds the insert buffer to a string's I/O chain for a control-area split or for updating or inserting a spanned record.

64 For processing without shared resources, IDAGWSGW locates empty buffer(s) in the string's I/O chain so that a spanned record can be inserted or lengthened (with additional segments) without using buffers that are being used for read-ahead buffering.

65 IDAWRBFR writes empty buffers whose contents have been modified.

66 IDAFRBA determines the RBA of the next control interval.

67 When the next RBA in sequence is in the next control area, IDAWRBFR prevents subsequent repositioning to a preceding control area for writing.

68 For processing with shared resources, IDAGRB reads the index control interval that contains the current sequence-set record.

69 When sequence-set pointers become invalid (because of a control-interval split or processing with shared resources), IDA019RC searches the sequence set for the current key.

70 For processing without shared resources, IDA019R3 (I/O Management) issues I/O to read a sequence-set record.

71 For processing without shared resources, IDAWAIT lets the I/O started in step 70 finish.

72 When the next RBA in sequence is in the next control area, IDAWRBFR prevents subsequent repositioning to a preceding control area for writing.

73 For backward processing, IDA019RV obtains the sequence-set record preceding the current sequenct-set record.

74 For processing with shared resources, IDAGRB obtains the next sequence-set record.

75 For processing without shared resources, IDA019R3 (I/O Management) issues I/O to read the next sequence-set record.

76 IDAWAIT lets the I/O started in step 75 finish.

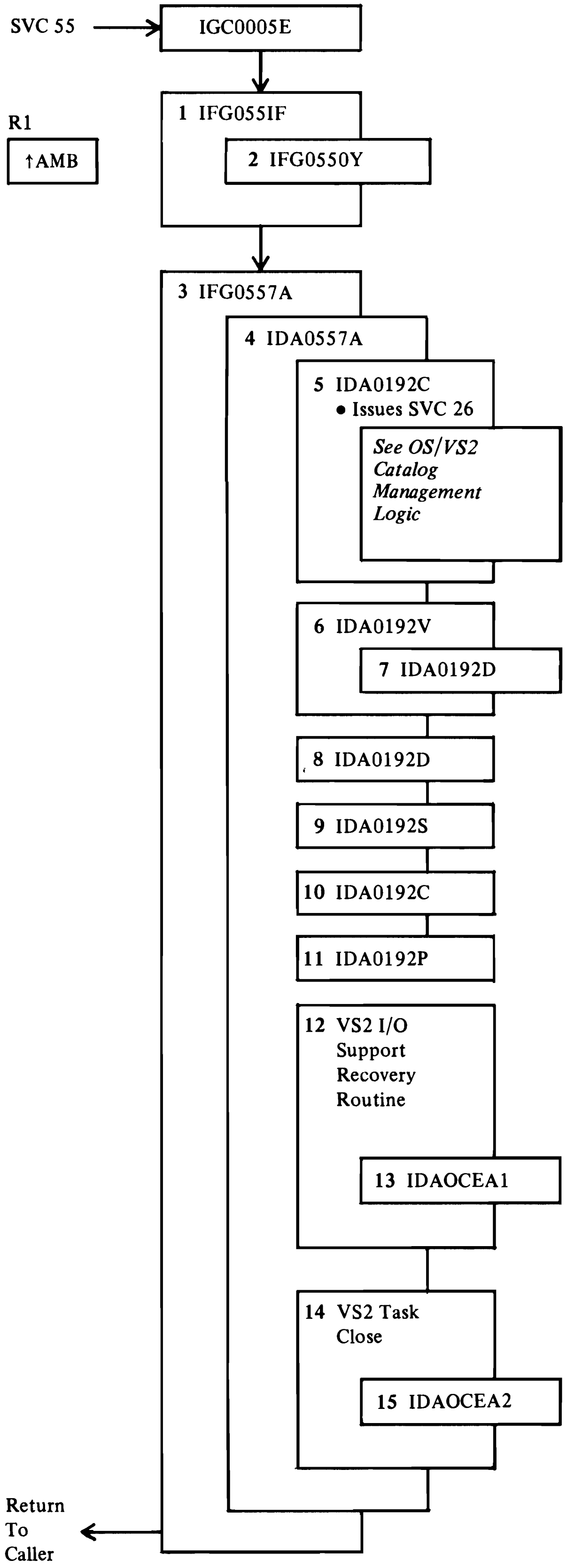


Figure 39. VSAM End of Volume (from VSAM Record Management: IDAEOVIF Procedure (in Module IDA019R5))

Notes for Figure 39

1 IGC0005E and IFG0551F are VS2 End-of-Volume modules (see *OS/VS2 Open/Close/EOV Logic* for details).

2 IFG0550Y is an alias-name for IFG0200N. It performs special processing for the VS2 catalog's ACB and is called when End of Volume is called for a catalog.

3 IFG0557A is an alias-name for IFG0192A.

4 IDA0557A is the VSAM End-of-Volume module.

5 IDA0192C calls VS2 Catalog Management (LOCATE) to retrieve the volume time stamp from the volume entry.

6 IDA0192V ensures that the required volumes are mounted for the VSAM object.

7 If the data set is stored on a mass storage volume, IDA0192D stages (via a Mass Storage System ACQUIRE) the new volume to a direct-access storage device.

8 If the data set is stored on a mass storage volume, IDA0192D stages any new extents to a direct-access storage device.

9 IDA0192S writes SMF record(s) type 64.

10 IDA0192C calls VS2 Catalog Management (LOCATE and UPDATE) to locate and update information in the object's catalog record.

11 Whenever End of Volume detects an error, IDA0192P issues a diagnostic message and traces VSAM control blocks if the Generalized Trace Facility (GTF) is active.

12-13
IDAOCEA1 runs as an ESTAE exit when an error occurs in Open. It logs system information and returns to the VS2 I/O Support Recovery Routine to continue with termination.

14-15
IDAOCEA2 locates and frees storage used for VSAM data sets in the system queue area and the common service area.

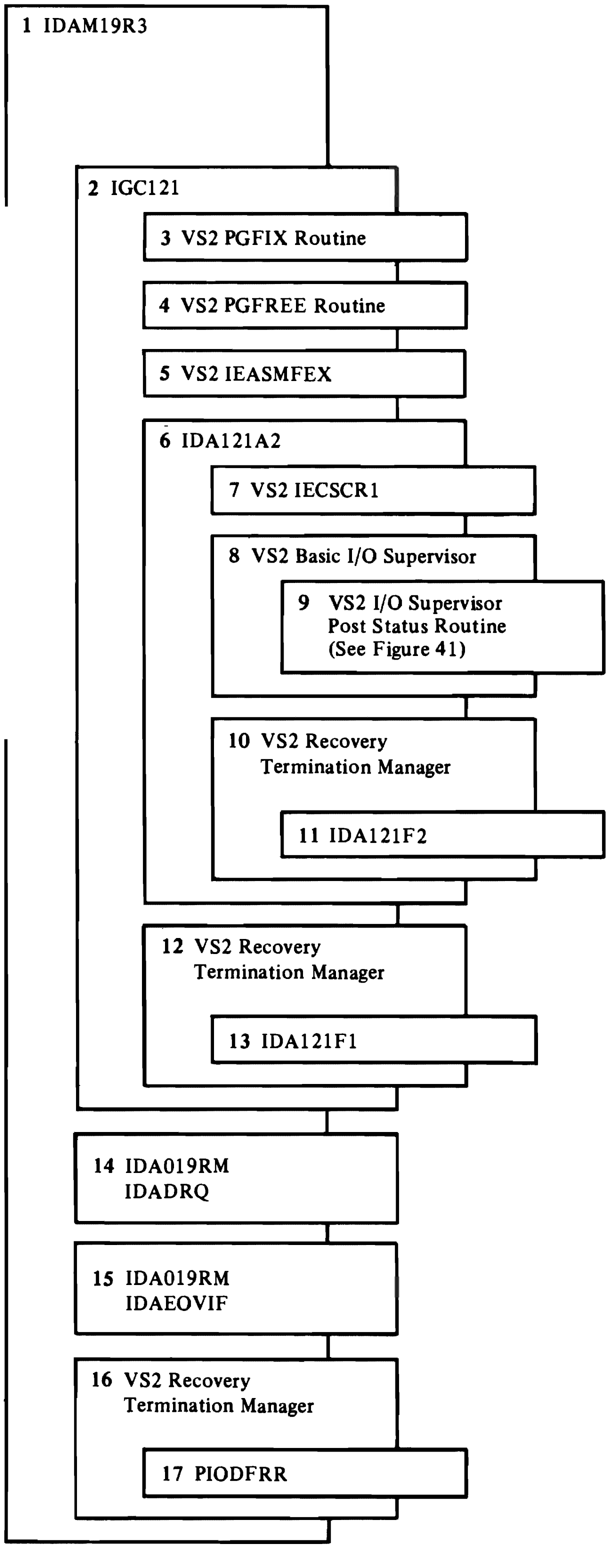

1 IDAM19R3
2 IGC121
3 VS2 PGFIX Routine
4 VS2 PGFREE Routine
5 VS2 IEASMFEX
6 IDA121A2
7 VS2 IECSCR1
8 VS2 Basic I/O Supervisor
9 VS2 I/O Supervisor Post Status Routine (See Figure 41)
10 VS2 Recovery Termination Manager
11 IDA121F2
12 VS2 Recovery Termination Manager
13 IDA121F1
14 IDA019RM IDADRQ
15 IDA019RM IDAEOVIF
16 VS2 Recovery Termination Manager
17 PIODFRR

Figure 40. I/O Management: Translating Virtual Addresses to Real Addresses and Completing a Channel Program for I/O

Notes to Figure 40

1 Buffer Management requests I/O Management to read or write data. IDAM19R3 prepares for supervisor-state processing and passes a request on to IGC121.

2 IGC121 chains together the CPAs required for the request and determines whether the RBAs of records or control intervals in the request are covered by the extent definition blocks (EDBs) that exist for the data set. IGC121 builds lists of virtual and real addresses for use by the channel program that the VS2 I/O Supervisor will execute to do I/O.

3 The VS2 PGFIX Routine fixes in real storage the virtual pages that contain the buffers required to do the I/O.

4 The VS2 PGFREE Routine releases pages that have been fixed in real storage in the case where an error occurs in virtual-to-real address translation and IGC121 cannot continue with the request.

5 VS2 IEASMFEX sets up for counting EXCPs used by the VS2 I/O Supervisor in doing I/O.

6 IDA121A2 completes the segments of channel programs passed to it by IGC121 (or by the OS/VS Auxiliary Storage Manager, which enters I/O Management at IDA121A2) and chains them together to form a single channel program for use by the VS2 I/O Supervisor.

7 VS2 IECSCR1 converts the physical-record number in the first channel program segment into a sector value for use with devices having rotational position sensing.

8 The VS2 Basic I/O Supervisor receives a STARTIO request from IDA121A2. It schedules the I/O and returns to IDA121A2.

9 After I/O completion, the VS2 I/O Supervisor Post Status Routine determines whether the I/O was successful and decides which VSAM end appendage should get control.

10 The VS2 Recovery Termination Manager gets control when an error occurs in VS2. If a functional recovery routine has been set up to get control in case of an error, the Recovery Termination Manager gives control to it.

11 IDA121F2 is the functional recovery routine for IDA121A2. It frees pages fixed in real storage by IGC121, releases the local lock that IDA121A2 obtains for storage protection, and issues an SDUMP macro to record information in SYS1.DUMP.

12 See step-**10** note.

13 IDA121F1 is the functional recovery routine for IGC121. Its processing is similar to that described in step-**11** note.

14 When IDAM19R3 defers a request because End of Volume is processing and the request requires that the processing be completed or that End of Volume do other processing for the request, IDADRQ awaits an indication that End of Volume is finished.

15 IDAEOVIF handles a request from IDAM19R3 for End-of-Volume processing. IDAM19R3 expects End of Volume to create an EDB to cover some RBA in the request from Buffer Management.

16 See step-**10** note.

17 PIODFRR is the functional recovery routine for IDAM19R3. It releases the local lock that IDAM19R3 may have obtained for storage protection and issues an SDUMP macro to record information in SYS1.DUMP.

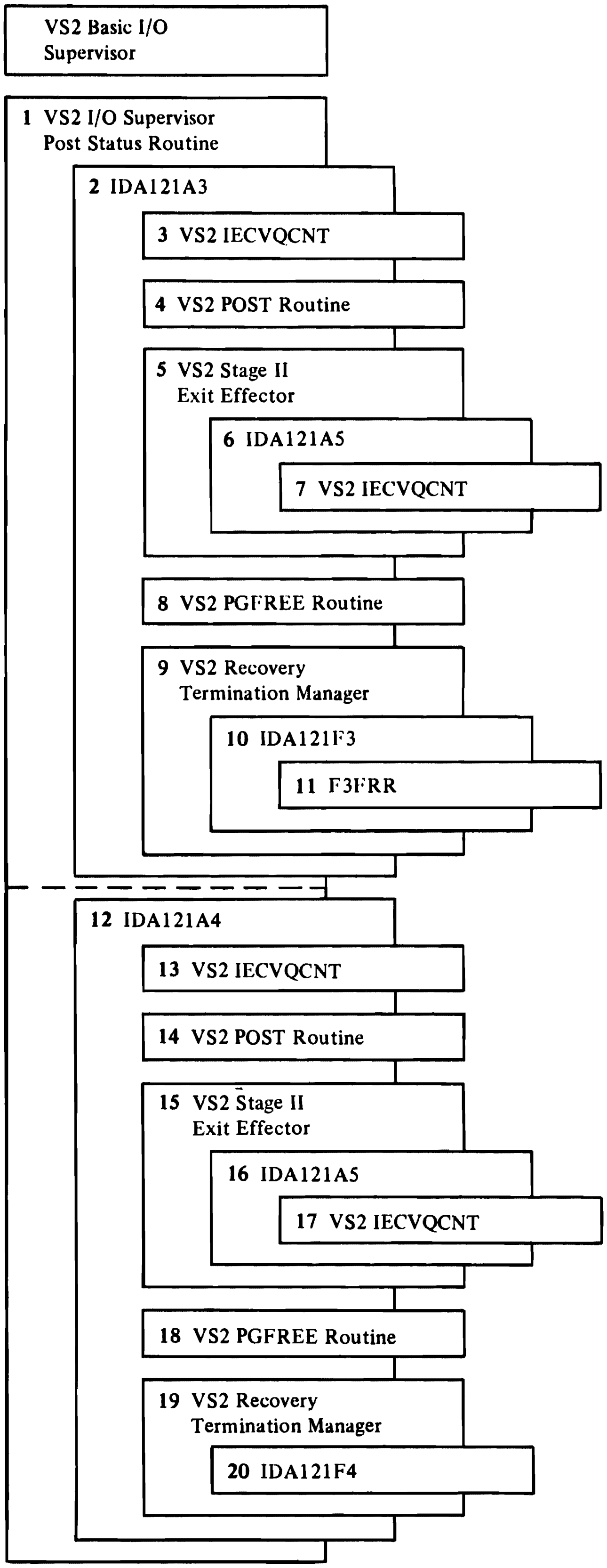


Figure 41. I/O Management: Processing after VS2 I/O Supervisor Completes I/O

Notes for Figure 41

1 After I/O completion, the VS2 I/O Supervisor Post Status Routine determines whether the I/O was successful and decides which VSAM end appendage should get control.

2 IDA121A3 indicates I/O completion in the BUFCs and does housekeeping following successful I/O and provides for the caller of I/O Management to get control back.

3 In the case where I/O activity is being allowed to continue to completion, but no new I/O is being started (for quiescing the task), IECVQCNT decrements the count of outstanding I/O. (When the count is reduced to 0, the task can be closed down.)

4 For a synchronous request, the VS2 POST Routine indicates in an ECB that I/O has completed.

5 For an asynchronous request or for a synchronous request when End of Volume is waiting to process, the VS2 Stage II Exit Effector schedules IDA121A5.

6 IDA121A5 itself gets control asynchronously in relation to the end appendage to give control back to the requester of I/O or to End of Volume.

7 See step-3 note.

8 The VS2 PGFREE Routine releases pages fixed in real storage by IGC121.

9 The VS2 Recovery Termination Manager gets control when an error occurs in OS. If a functional recovery routine has been set up to get control in case of an error, the Recovery Termination Manager gives control to it.

10 IDA121F3 is the functional recovery routine for IDA121A3. It duplicates the processing of IDA121A3, in order to continue processing if possible.

11 F3FRR is a functional recovery routine within IDA121F3. It frees pages fixed in real storage by IGC121 and issues an SDUMP macro to record information in SYS1.DUMP.

12 IDA121A4 indicates completion in BUFCs whose I/O completed and error in the BUFC whose I/O failed. It retries I/O that yet has a chance to complete successfully. It does housekeeping and provides for the caller of I/O Management to get control back.

13-19
See notes for steps 3-9.

20 IDA121F4 is the functional recovery routine for IDA121A4. It frees pages that were fixed in real storage by IGC121 and issues an SDUMP to record information in SYS1.DUMP.

DIRECTORY

This directory identifies the method of operation diagrams and program organization compendiums for the modules and external procedures of VSAM. The module directory and the external procedure directory contain the same information, ordered differently.

The microfiche for VSAM contains a "Control Flow Report" that lists each module and procedure of VSAM, along with the modules and procedures that call it or that it calls.

Module Directory

The module directory is organized alphabetically by symbolic module name. It lists the module's descriptive name, its external procedure names (external entry points), the component to which it belongs, and the method of operation diagrams and program organization figures that refer to it.

The components are identified by:

C	Close
C/R	Checkpoint/Restart
CBM	Control Block Manipulation
EOV	End of Volume
II	ISAM Interface
IOM	I/O Management
O	Open
RM	Record Management

		External		**Method of**	**Program**
Module	**Descriptive**	**Procedure**		**Operation**	**Organization**
Name	**Name**	**Names**	**Component**	**Diagrams**	**Figures**
AMDUSRF9	IMDPRDMP Format Appendage	IMDUSRF9	—	—	—
IDACKRA1	ESTAE	IDACKRA1	C/R	AJ, AK, AL1, AM	17, 18
IDAICIA1	ISAM-Interface: Data-Set Management Recovery Routine	IDAICIA1	II	AG	11
IDAIIFBF	ISAM Interface: FREEDBUF Processing	IDAIIFBF	II	BU2	—
IDAIIPM1	ISAM Interface: QISAM Load-Mode Processing	IDAIIPM1	II	BU1	—
IDAIIPM2	ISAM Interface: QISAM Scan-Mode Processing	IDAIIPM2	II	BU1	—
IDAIIPM3	ISAM Interface: BISAM Processing	IDAIIPM3	II	BU2	—
IDAIISM1	ISAM Interface: SYNAD Processing	IDAIISM1	II	BU2	—
IDAM19R3	Problem-State I/O Driver	IDA019R3	IOM	BG1, BS1, BS2, BS3, BS4, DA1, DA2, DA3	20, 22, 38, 40
		PIODFRR	IOM	DA1	40
IDAOCEA1	Data-Set Management Recovery Routine (Force Close Executor)	IDAOCEA1	O/C/EOV	AF, AG	8, 10, 11, 12, 16, 39
IDAOCEA2	Task Close Executor	IDAOCEA2	O/C/EOV	AH1, AH2, AH3	8, 39
IDAOCEA4	BLDVRP/DLVRP ESTAE Routine	IDAOCEA4	O/C/EOV	AF, AG	10, 16
IDA0A05B	Restart	IDA0A05B	C/R	AJ, AL1, AL2	18
IDA0C05B	SSCR and Initial DEB Processing	IDA0C05B	C/R	AK	18

Module Directory

Module Name	Descriptive Name	External Procedure Names	Component	Method of Operation Diagrams	Program Organization Figures
IDA0C06C	Checkpoint	IDA0C06C	C/R	AI, AJ	17
IDA0I96C	SSCR Build and Cleanup	IDA0I96C	C/R	AJ	17
IDA019C1	Control Block Manipulation	IDA019C1	CBM	CA, CB1, CB2	—
IDA019RA	Direct Record Locate	IDA019RA	RM	BC, BE, BH1, BJ	20, 21, 22, 24
IDA019RB	Index Search	IDA019RB	RM	BC, BH1, BH8, BJ, BM	22, 34
IDA019RC	Search Compressed Index Block	IDA019RC	RM	BC, BH1, BH2, BH6, BI, BJ, BM, BS4	22, 24, 25, 32
IDA019RD	DD DUMMY Processing	IDA019RD	RM	—	—
IDA019RE	Control-Interval Split	IDA019RE	RM	BH1, BH3, BH7	24, 25, 27, 28, 32
		IDAREPOS	RM	—	—
IDA019RF	Control-Area Split	IDA019RF	RM	BH1, BH2, BH3, BH4, BH5	24, 25, 26, 27, 33, 34
IDA019RG	Index Create	IDA019RG	RM	BG1, BG2, BG3, BG4, BG5, BK2	26, 29, 30, 31
		IDAIST	RM	BG3, BG4, BG5, BH9	29, 30
IDA019RH	Index Insert	IDA019RH	RM	BH3, BH6, BH7, BH8	27, 32, 33, 34
		IDAIVIXB	RM	—	—
		IDASPACE	RM	—	—
IDA019RI	Index Upgrade	IDA019RI	RM	BH4, BH5, BH7, BH8, BH9	28, 33, 34
		IDAHLINS	RM	BH4	28
		IDANEWRD	RM	BH4	28
IDA019RJ	Split Index Record	IDA019RJ	RM	BH4, BH7, BH8, BH9, BH10	34
		IDAR	RM	BG5, BH9, BH10	30
		IDAWR	RM	BG4, BG5, BH10	30, 31
IDA019RK	Preformat	IDA019RK	RM	BG2, BH4, BH8, BH9, BK2, BN2, BO1	24, 26, 28, 29
IDA019RL	Data Modify	IDA019RL	RM	BH2, BI	24, 25
IDA019RM	Data Insert	IDA019RM	RM	BE, BF, BH1, BH2, BH3	24, 25, 26, 27, 28, 40
		IDACHKKR	RM	—	—
IDA019RN	Indexing Subroutines	IDA019RN	RM	—	—
		IDAAQR	RM	BG3, BG4, BG5, BH8, BH9	29, 20, 33, 34
		IDAER	RM	BG5	30
IDA019RO	Verify	IDA019RO	RM	BM	—

Module Name	Descriptive Name	External Procedure Names	Component	Method of Operation Diagrams	Program Organization Figures
IDA019RP	ENDREQ and JRNAD	IDA019RP	RM	BK1, BK2	—
		IDAENDRQ	RM	BK1, BK2	31
		IDATJXIT	RM	BN1, BN3	20, 21, 24, 25, 26, 27, 35, 38
IDA019RQ	Relative Record Subroutines	IDA019RQ	RM	BO1, BO2	35
IDA019RR	Relative Record Driver	IDA019RR	RM	BC, BD, BJ, BO1	23, 25, 35
		IDARRDRL	RM	BJ, BO1	23, 35
IDA019RS	Spanned Record Data Modify	IDA019RS	RM	BH2, BI	24, 25
		IDAADSEG	RM	BH1, BH2	24, 25
		IDAMVSEG	RM	BH1, BH2	24, 25
IDA019RT	Spanned Record Data Insert	IDA019RT	RM	BE, BF, BH1	24
		IDADARTV	RM	BC, BD	20, 21
		IDAJRNSR	RM	—	—
		IDASPNPT	RM	—	—
IDA019RU	Alternate-Index Upgrade Driver	IDA019RU	RM	BC, BD, BE, BH1, BI, BR	37
		IDAXGPLH	RM	BB1	—
IDA019RV	Locate Previous Sequence-Set Record	IDA019RV	RM	—	28
		IDAADVPH	RM	BD	21
		IDARVRS1	RM	—	38
IDA019RW	Buffer Management, Part 2	IDA019RW	RM	—	38
		IDAABF	RM	BH4	28, 35, 38
		IDAAIBF	RM	—	38
		IDAFRBA	RM	BN3, BS1, BS2, BS4	22, 38
		IDAGWSGW	RM	—	38
IDA019RX	Path Processing Driver	IDA019RX	RM	BQ	36
		IDAGETWS	RM	—	—
		IDARELWS	RM	—	—
		IDARXBD	RM	—	—
IDA019RY	Shared Resources Buffer Management	IDA019RY	RM	BP1, BP2, BP3, BS1, BS2, DA1	38
IDA019RZ	Buffer Management Interface	IDA019RZ	RM	BC, BS1, BS2, BS4	38
		IDAEXCL	RM	—	38
		IDAFREEB	RM	BC, BD, BE, BG1, BG5, BH3, BH4, BH5, BM, BN1, BN2, BN3, BO1, BO2, BS1	20, 21, 22, 23, 24, 26, 27, 28, 30, 33, 34, 35, 38
		IDAGNFL	RM	BG3, BH3, BH8	29, 30, 33, 34, 38

Module Name	Descriptive Name	External Procedure Names	Component	Method of Operation Diagrams	Program Organization Figures
IDA019RZ	(Continued)	IDAGNNFL	RM	BE, BF, BG1, BG4, BH4, BH5, BN2, BO1	24, 26, 27, 38, 35, 38
		IDAGNXT	RM	BC, BD, BH4, BN1, BO1	20, 21, 22, 28, 35, 38
		IDAGRB	RM	BC, BE, BH1, BH3, BH4, BH5, BH9, BH10, BJ, BM, BN1, BO1, BS2, BS3	21, 22, 23, 27, 28, 33, 34, 38
		IDAGWSEG	RM	—	38
		IDAMRKBF	RM	—	38
		IDASBF	RM	BE, BH5, BK1, BN1, BN2	22, 23, 27, 28, 35, 38
		IDASCHBF	RM	—	—
		IDAWAIT	RM	BS2, BS3, BS4	22, 38
		IDAWRBFR	RM	BE, BG2, BH3, BH4, BH5, BH8, BH9, BK1, BK2, BN2, BN3, BO1, BO2	20, 24, 25, 26, 27, 28, 32, 35, 38
		IDAWRTBR	RM	—	—
IDA019R1	Decode and Validate	IDA019R1	RM	AD7, BB1, BK1, BK2, BL	12, 14, 20, 21, 23, 24, 35, 36
IDA019R2	Buffer Management, Part 1	IDA019R2	RM	BS1, BS2, BS3, DA1	38
IDA019R4	Keyed/Addressed Request Driver	IDA019R4	RM	BC, BD, BE, BF, BH1, BH3, BQ, BR	20, 21, 22, 24, 25, 36, 37
IDA019R5	I/O-Error Analysis	IDA019R5	RM	BB3, BK1, BL	38
		IDADRQ	RM	BP1, BP3, BS2, DA1	24, 35, 38, 40
		IDAEOVIF	RM	BE, BG2, BN2, BO1, DA1, DA2, DA4	26, 28, 29, 34, 40
		IDAERROR	RM	—	—
		IDAEXEX	RM	—	38
		IDAEXITR	RM	BK1, BL	—
		IDARSTRT	RM	—	—
IDA019R8	Control-Interval Processing	IDA019R8	RM	BM, BN1, BN2 BN3	—
IDA019SA	Control-Interval Initialization—Create Entry-Sequenced Data Set	IDA019SA	RM	BE, BF, BG1, BG2, BG3, BK2	26, 29, 30
IDA019SB	Dynamically Build Channel Program Area for Shared Resources	IDA019SB	RM	—	—
IDA019SF	Control-Area Split-Spanned Records	IDA019SF	RM	BH4	28
IDA019S1	Improved Control-Interval Processing Driver	IDA019S1	RM	BN1, BN3	—

Module Directory

Module Name	Descriptive Name	External Procedure Names	Component	Method of Operation Diagrams	Program Organization Figures
IDA019S3	Improved Control-Interval Processing—I/O Management	IDA019S3	RM	BN1, BN3	—
IDA0192A	VSAM Open String	IDA0192A	O	AC1, AC2, AC7, AG	5, 6, 7, 11
IDA0192B	Open a Cluster	IDA0192B	O	AC4, AL1	8, 18
IDA0192C	Catalog Interface	IDA0192C	O	AC2, AC4, AD6 AD7, AE2, AL1, BT1	5, 6, 8, 12, 14, 15, 18, 39
IDA0192D	Stage/Destage (ACQUIRE/RELINQUISH)	IDA0192D	O/C/EOV	AD6, AE2, BT2	8, 12, 14, 18, 39
IDA0192F	Open Base Cluster, Path, and Upgrade Alternate Index	IDA0192F	O	AC3, AC4, AC5, AC6	8,18
IDA0192G	Data-Space Security Verification	IDA0192G	O	—	15
IDA0192I	ISAM Interface: Open Processing	IDA0192I	II	AC1, AC7	7
IDA0192M	Virtual-Storage Manager	IDA0192M	O/C/EOV	AL2	8, 10, 16
IDA0192P	VSAM Open/Close/EOV: Problem Determination	IDA0192P	O	AD5, AD6, AE2, AH3	8, 12, 14, 18, 39
IDA0192S	VSAM Open/Close/EOV: SMF Record Build	IDA0192S	O	AC7, AD6, AE2	8, 12, 14, 39
IDA0192V	Volume Mount and Verify	IDA0192V	O	AC3, BT1	5, 8,18, 39
IDA0192W	Channel-Program-Area Build	IDA0192W	O	AC1, AC4	8, 10
IDA0192Y	String Build and Shared-Resource Processor	IDA0192Y	O/C	AC1, AC4, AC5, AD5, AF1, AH3, AL1, AL2	8, 10, 12, 14, 16, 18
IDA0192Z	Control-Block Build	IDA0192Z	O	AC4, AL1	8, 18
IDA0195A	VSAM SNAP Format Routine	IDA0195A	O/C/EOV	—	—
IDA0200B	Close a Cluster	IDA0200B	C	AD1, AD3, AD4, AD6	12
IDA0200S	ISAM Interface: Close Processing	IDA0200S	II	AD1, AD7	11
IDA0200T	VSAM Close String	IDA0200T	C	AD1, AD2, AD3, AD4, AD5, AD7	12
IDA0231B	Close (TYPE=T) a Cluster	IDA0231B	C	AE1, AE2	14
IDA0231T	VSAM Close (TYPE=T) String	IDA0231T	C	AE1, AE2	14
IDA0557A	VSAM End of Volume	IDA0557A	EOV	BT1, BT2	—
IDA121A2	Actual Block Processor	IDA121A2	IOM	DA2, DA3, DA4	40
		IDA121F2	IOM	DA3	40
IDA121A3	Normal End Appendage	IDA121A3	IOM	DA4	41
		F3FRR	IOM	DA4	41
		IDA121F3	IOM	DA4	41
IDA121A4	Abnormal End Appendage	IDA121A4	IOM	DA5	41
		IDA121F4	IOM	DA5	41
IDA121A5	Asynchronous Routine	IDA121A5	IOM	DA4, DA5	41
IDA121A6	Purge Routine	IDA121A6	IOM	—	—
IDA121CV	Communication Vector Table (IEZABP)	—	IOM	—	—

Module Directory

Module Name	Descriptive Name	External Procedure Names	Component	Method of Operation Diagrams	Program Organization Figures
IEFVAMP	AMP Parameter Interpreter	IEFNB902	—	—	—
IFG0192A	VSAM Open/Close/EOV String Load (Interface between OS/VS and VSAM O/C/EOV)	IFG0192A	O/C/EOV	AF	5, 7, 8, 11, 12, 14
IFG0192Y	BLDVRP/DLVRP Load Routine	IFG0192Y	O/C/EOV	—	10, 16
IGC121	Supervisor-State I/O Driver	IGC121	IOM	DA1, DA2, DA3	40
		IDA121F1	IOM	DA2	40

External Procedure Directory

Procedure Name	Module Name	Descriptive Name	Method of Operation Diagrams	Program Organization Figures
F3FRR	IDA121A3	Functional Recovery Routine (of IDA121F3)	DA4	41
IDAABF	IDA019RW	Buffer Management: Add Buffer to Placeholder (PLH)	BH4	28, 35, 38
IDAADSEG	IDA019RS	Insert a Spanned-Record-Segment Entry into a Sequence-Set Record	BH1, BH2	24, 25
IDAADVPH	IDA021RV	Advance Placeholder Backwards	BD	21
IDAAIBF	IDA019RW	Add Insert Buffer to Chain	—	38
IDAAQR	IDA019RN	Split Index Record: Assign RBA to the Index Record	BG3, BG4, BG5, BG8, BH9	29, 30, 33, 34
IDACHKKR	IDA019RM	Check Key for Proper Key Range	—	—
IDACI96C	IDA0I96C	Free VSAM Checkpoint/Restart Storage	AJ, AL2	17, 18
IDACKRA1	IDACKRA1	Checkpoint/Restart: ESTAE	AJ, AK, AL1, AM	17, 18
IDADARTV	IDA019RT	Retrieve a Spanned Record	BC, BD	20, 21
IDADRQ	IDA019R5	Data Insert: Defer the Request until the Device is Available	BP1, BP3, BS2, DA1	24, 35, 38, 40
IDAENDRQ	IDA019RP	ENDREQ Request	BK1, BK2	31
IDAEOVIF	IDA019R5	Data Insert: Interface to VSAM End of Volume	BE, BG2, BN2, BO1, DA1, DA2, DA4	26, 28, 29, 34, 40
IDAER	IDA019RN	Index Create: Erase Dummy Entry from the Index Record	BG5	30
IDAERROR	IDA019R5	Determine Which Exit to Take	—	—
IDAEXCL	IDA019RZ	Exclusive Control	—	38
IDAEXEX	IDA019R5	Exit to User Exception Routine	—	38
IDAEXITR	IDA019R5	Exit to User Routine	BK1, BL	—
IDAFRBA	IDA019RW	Buffer Management: Determine next RBA for Sequential Processing	BN3, BS1, BS2, BS4	22, 38
IDAFREEB	IDA019RZ	Free a Buffer	BC, BD, BE, BG1, BG5, BH3, BH4, BH5, BM, BN1, BN2, BN3, BO1, BO2, BS1	20, 21, 22, 23, 24, 25, 27, 28, 30, 33, 34, 35, 38
IDAGETWS	IDA019RX	Get Working Storage	—	—
IDAGNFL	IDA019RZ	Buffer Management: Obtain an Empty Buffer	BG3, BH3, BH8	29, 30, 33, 34, 38
IDAGNNFL	IDA019RZ	Buffer Management: Obtain Next Empty Buffer for the Placeholder	BE, BF, BG1, BG4, BH4, BH5, BN2, BO1	24, 26, 27, 28, 35, 38
IDAGNXT	IDA019RZ	Buffer Management: Obtain Next Buffer in Sequence	BC, BD, BH4, BN1, BO1	20, 21, 23, 28, 35, 38
IDAGRB	IDA019RZ	Buffer Management: Obtain the Buffer That Contains the Specified RBA	BC, BE, BH1, BH3, BH4, BH5, BH9, BH10, BJ, BM, BN1, BO1, BS2, BS3	21, 22, 23, 27, 28, 33, 34, 38
IDAGWSEG	IDA019RZ	Get a Work Segment (for Shared Resources)	—	38
IDAGWSGW	IDA019RW	Get a Work Segment	—	38

External Procedure Directory

Procedure Name	Module Name	Descriptive Name	Method of Operation Diagrams	Program Organization Figures
IDAHLINS	IDA019RI	Insert Entry into Index-Set Record	BH4	28
IDAICIA1	IDAICIA1	ISAM Interface: Data-Set Management Recovery Routine	AG	11
IDAIIFBF	IDAIIFBF	ISAM Interface: FREEDBUF Processing	BU2	—
IDAIIPM1	IDAIIPM1	ISAM Interface: Load-Mode Processing	BU1	—
IDAIIPM2	IDAIIPM2	ISAM Interface: QISAM Scan-Mode Processing	BU1	—
IDAIIPM3	IDAIIPM3	ISAM Interface: BISAM Processing	BU2	—
IDAIISM1	IDAIISM1	ISAM Interface: SYNAD Processing	BU2	—
IDAIST	IDA019RG	Index Create: Insert Entry into Index Record	BG3, BG4, BG5, BH9	29, 30
IDAIVIXB	IDA019RH	Index Insert: Invalidate Buffers Containing a Copy of the Modified Index Record	— —	
IDAJRNSR	IDA019RT	Journal a Spanned-Record Segment	—	—
IDAMRKBF	IDA019RZ	Mark a Buffer	—	38
IDAMVSEG	IDA019RS	Move a Segment	BH1, BH2	24, 25
IDANEWRD	IDA019RI	Initialize a New Sequence-Set Record	BH4	28
IDAOCEA1	IDAOCEA1	Data-Set Management Recovery Routine (Force Close Executor)	AF, AG	8, 10, 11, 12, 16, 39
IDAOCEA2	IDAOCEA2	Task Close Executor	AH1, AH2, AH3	8, 39
IDAOCEA4	IDAOCEA4	BLDVRP/DLVRP ESTAE Routine	AF, AG	10, 16
IDAR	IDA019RJ	Split Index Record: Read the Record	BG5, BH9, BH10	30
IDARELWS	IDA019RX	Release Working Storage	—	—
IDAREPOS	IDA019RE	Reposition Placeholder	—	—
IDARRDRL	IDA019RR	Direct Record Locate for Relative Record	BJ, BO1	23, 35
IDARSTRT	IDA019R5	Restart	—	—
IDARVRS1	IDA019RV	Order Buffers	—	38
IDARXBD	IDA019RX	Increase Working Buffer Length	—	—
IDASBF	IDA019RZ	Buffer Management: Remove Buffers from Placeholder	BE, BH5, BK1, BN1, BN2	22, 23, 27, 28, 35, 38
IDASCHBF	IDA019RZ	Share a Buffer	—	—
IDASPACE	IDA019RH	Check an Index Record to Ensure It Can Be Split	—	—
IDASPNPT	IDA019RT	Make an INDEX Entry for a Spanned-Record Segment	—	—
IDATJXIT	IDA019RP	Control-Interval Request: Take the Journal Exit	BN1, BN3	20, 21, 24, 25, 26, 27, 35, 38
IDAWAIT	IDA019RZ	Buffer Management: Wait for Completion of I/O Operations	BS2, BS3, BS4	22, 38
IDAWR	IDA019RJ	Split Index Record: Write the Index Record	BG4, BG5, BH10	30, 31

External Procedure Directory

Procedure Name	Module Name	Descriptive Name	Method of Operation Diagrams	Program Organization Figures
IDAWRBFR	IDA019RZ	Buffer Management: Write the Buffer	BE, BG2, BH3, BH4, BH5, BH8, BH9, BK1, BK2, BN2, BN3, BO1, BO2	20, 24, 25, 26, 27, 28, 32, 35, 38
IDAWRTBF	IDA019RZ	Write a Buffer	—	—
IDAXGPLH	IDA019RU	Get a Placeholder	BB1	—
IDA0A05B	IDA0A05B	Checkpoint/Restart: Restart	AJ, AL1, AL2	18
IDA0C05B	IDA0C05B	Checkpoint/Restart: SSCR and Initial DEB Processing	AK	18
IDA0C06C	IDA0C06C	Checkpoint/Restart: Checkpoint	AI, AJ	17
IDA0I96C	IDA0I96C	Checkpoint/Restart: SSCR Build and Cleanup	AJ	17
IDA019C1	IDA019C1	Control Block Manipulation	CA, CB1, CB2	—
IDA019RA	IDA019RA	Direct Record Locate	BC, BE, BH1, BJ	20, 21, 22, 24
IDA019RB	IDA019RB	Index Search	BC, BH1, BH8, BJ, BM	22, 34
IDA019RC	IDA019RC	Search Compressed Index Block	BC, BH1, BH2, BH6, BI, BJ, BM, BS4	22, 24, 25, 32
IDA019RD	IDA019RD	DD DUMMY Processing	—	—
IDA019RE	IDA019RE	Control-Interval Split	BH1, BH3, BH7	24, 25, 27, 28, 32
IDA019RF	IDA019RF	Control-Area Split	BH1, BH2, BH3, BH4, BH5	24, 25, 26, 27, 33, 34
IDA019RG	IDA019RG	Index Create	BG1, BG2, BG3, BG4, BG5, BK2	26, 29, 30, 31
IDA019RH	IDA019RH	Index Insert	BH3, BH6, BH7, BH8	27, 32, 33, 34
IDA019RI	IDA019RI	Index Upgrade	BH4, BH5, BH7, BH8, BH9	28, 33, 34
IDA019RJ	IDA019RJ	Split Index Record	BH4, BH7, BH8, BH9, BH10	34
IDA019RK	IDA019RK	Preformat	BG2, BH4, BH8, BH9, BK2, BN2, BO1	24, 26, 28, 29
IDA019RL	IDA019RL	Data Modify	BH2, BI	24, 25
IDA019RM	IDA019RM	Data Insert	BE, BF, BH1, BH2, BH3	24, 25, 26, 27, 28, 40
IDA019RN	IDA019RN	Indexing Subroutines	—	—
IDA019RO	IDA019RO	Verify	BM	—
IDA019RP	IDA019RP	ENDREQ and JRNAD	BK1, BK2	—
IDA019RQ	IDA019RQ	Relative Record Subroutines	BO1, BO2	35
IDA019RR	IDA019RR	Relative Record Driver	BC, BD, BJ, BO1	23, 25, 35
IDA019RS	IDA019RS	Spanned Record Data Modify	BH2, BI	24, 25
IDA019RT	IDA019RT	Spanned Record Data Insert	BE, BF, BH1	24

External Procedure Directory

Procedure Name	Module Name	Descriptive Name	Method of Operation Diagrams	Program Organization Figures
IDA019RU	IDA019RU	Alternate-Index Upgrade Driver	BC, BD, BE, BH1, BI, BR	37
IDA019RV	IDA019RV	Locate Previous Sequence-Set Record	—	28
IDA019RW	IDA019RW	Buffer Management, Part 2	—	38
IDA019RX	IDA019RX	Path Processing Driver	BQ	36
IDA019RY	IDA019RY	Shared Resources Buffer Management	BP1, BP2, BP3, BS1, BS2, DA1	38
IDA019RZ	IDA019RZ	Buffer Management Interface	BC, BS1, BS2, BS4	38
IDA019R1	IDA019R1	Record Management: Request Decode and Validate	AD7, BB1, BK1, BK2, BL	12, 14, 20, 21, 23, 24, 35, 36
IDA019R2	IDA019R2	Buffer Management, Part 1	BS1, BS2, BS3, DA1	38
IDA019R3	IDAM19R3	I/O Management: Problem-State I/O Driver	BG1, BS1, BS2, BS3, BS4, DA1, DA2, DA3	20, 22, 38, 40
IDA019R4	IDA019R4	Keyed/Addressed Request Driver	BC, BD, BE, BF, BH1, BH3, BQ, BR	20, 21, 22, 24, 25, 36, 37
IDA019R5	IDA019R5	I/O Error Analyais	BB3, BK1, BL	38
IDA019R8	IDA019R8	Control-Interval Processing	BM, BN1, BN2, BN3	—
IDA019SA	IDA019SA	Control-Interval Initialization-Create Entry-Sequenced Data Set	BE, BF, BG1, BG2, BG3, BK2	26, 29, 30
IDA019SB	IDA019SB	Dynamically Build Channel Program Area for Shared Resources	—	—
IDA019SF	IDA019SF	Control-Area Split-Spanned Records	BH4	28
IDA019S1	IDA019S1	Improved Control-Interval Processing Driver	BN1, BN3	—
IDA019S3	IDA019S3	Improved Control-Interval Processing-I/O Management	BN1, BN3	—
IDA0192A	IDA0192A	VSAM Open String	AC1, AC2, AC7, AG	5, 6, 7, 11
IDA0192B	IDA0192B	Open a Cluster	AC4, AL1	8, 18
IDA0192C	IDA0192C	VSAM Open/Close: Catalog Interface	AC2, AC4, AD6, AD7, AE2, AL1, BT1	5, 6, 8, 12, 14, 15, 18, 39
IDA0192D	IDA0192D	Stage/Destage (ACQUIRE/RELINQUISH)	AD6, AE2, BT2	8, 12, 14, 18, 39
IDA0192F	IDA0192F	Open Base Cluster, Path, and Upgrade Alternate Index	AC3, AC4, AC5, AC6	8, 18
IDA0192G	IDA0192G	Data-Space Security Verification	AL2	15
IDA0192I	IDA0192I	ISAM Interface: Open Processing	AC1, AC7	7
IDA0192M	IDA0192M	Virtual Storage Manager	—	8, 10, 16
IDA0192P	IDA0192P	VSAM Open/Close/EOV: Problem Determination	AD5, AD6, AE2, AH3	8, 12, 14, 18, 39

External Procedure Directory

Procedure Name	Module Name	Descriptive Name	Method of Operation Diagrams	Program Organization Figures
IDA0192S	IDA0192S	VSAM Open/Close/EOV: SMF Record Build	AC7, AD6, AE2	8, 12, 14, 39
IDA0192V	IDA0192V	Volume Mount and Verify	AC3, BT1	5, 8, 18, 39
IDA0192W	IDA0192W	Channel-Program-Area Build	AC1, AC4	8, 10
IDA0192Y	IDA0192Y	String Build and Shared-Resource Processor	AC1, AC4, AC5, AD5, AF1, AH3, AL1, AL2	8, 10, 12, 14, 16, 18
IDA0192Z	IDA0192Z	Control Block Build	AC4, AL1	8, 18
IDA0200B	IDA0200B	Close a Cluster	AD1, AD3, AD4, AD6	12
IDA0200S	IDA0200S	ISAM Interface: Close Processing	AD1, AD7	11
IDA0200T	IDA0200T	VSAM Close String	AD1, AD2, AD3 AD4, AD5, AD7	12
IDA0231B	IDA0231B	Close (TYPE=T) a Cluster	AE1, AE2	14
IDA0231T	IDA0231T	VSAM Close (TYPE=T) String	AE1, AE2	14
IDA0557A	IDA0557A	VSAM End of Volume	BT1, BT2	—
IDA121A2	IDA121A2	I/O Management: Actual Block Processor	DA2, DA3, DA4	40
IDA121A3	IDA121A3	I/O Management: Normal End Appendage	DA4	41
IDA121A4	IDA121A4	I/O Management: Abnormal End appendage	DA5	41
IDA121A5	IDA121A5	I/O Management: Asynchronous Routine	DA4, DA5	41
IDA121A6	IDA121A6	I/O Management: Purge Routine	—	—
IDA121F1	IGC121	Functional Recovery Routine	DA2	40
IDA121F2	IDA121A2	Functional Recovery Routine	DA3	40
IDA121F3	IDA121A3	Functional Recovery Routine	DA4	41
IDA121F4	IDA121A4	Functional Recovery Routine	DA5	41
IEFNB902	IEFVAMP	AMP Parameter Interpreter	—	—
IFG0192A	IFG0192A	VSAM Open/Close/EOV String Load (Interface between OS/VS and VSAM O/C/EOV)	AF	5, 7, 8, 11, 12, 14
IFG0192Y	IFG0192Y	BLDVRP/DLVRP Load Routine	—	10, 16
IGC121	IGC121	I/O Management: Supervisor State I/O Driver	DA1, DA2, DA3	40
IMDUSRF9	AMDUSRF9	IMDPRDMP Format Appendage	—	—
PIODFRR	IDAM19R3	Functional Recovery Routine	DA1	40

Module Packaging

Most VSAM modules reside in pageable virtual storage; some I/O-Management modules reside in the nucleus. The following table lists the VSAM load modules and transients that are resident in the SVCLIB or LPALIB library or in the nucleus. Those in the libraries are loaded into the pageable supervisor or link-pack area by nucleus initialization (NIP) at initial program load (IPL). Those in the nucleus are link-edited there when the system is generated.

Name	**Description**	**VSAM Modules**
Record Management		
IDA019L1	Main Record Management	IDAM19R3, IDA019RA, IDA019RB, IDA019RC, IDA019RD, IDA019RE, IDA019RF, IDA019RG, IDA019RH, IDA019RI, IDA019RJ, IDA019RK, IDA019RL, IDA019RM, IDA019RN, IDA019RO, IDA019RP, IDA019RQ, IDA019RR, IDA019RU, IDA019RV, IDA019RW, IDA019RX, IDA019RY, IDA019RZ, IDA019R1, IDA019R2, IDA019R4, IDA019R5, IDA019R8, IDA019SA, IDA019SB, IDA019SF
IDA019L2	Improved Control-Interval Processing	IDA019S1, IDA019S3
IDA019RS	Spanned Record Data Modify	IDA019RS
IDA019RT	Spanned Record Data Insert	IDA019RT
Open/Close/End of Volume and Checkpoint/Restart		
IDA0192A	Open/Close/End of Volume	IDACKRA1,IDA0A05B, IDA0C05B, IDA0C06C, IDA0I96C, IDA0192A, IDA0192B, IDA0192C, IDA0192D, IDA0192F, IDA0192G, IDA0192I, IDA0192M, IDA0192P, IDA0192S, IDA0192V, IDA0192W, IDA0192Y, IDA0192Z, IDA0200B, IDA0200S, IDA0200T, IDA0231B, IDA0231T, IDA0557A
ISAM Interface		
IDAIIFBF	FREEDBUF	IDAIIFBF
IDAIIPM1	QISAM Load	IDAIIPM1
IDAIIPM2	QISAM Scan	IDAIIPM2
IDAIIPM3	BISAM	IDAIIPM3
IDAIISM1	SYNAD	IDAIISM1
I/O Management		
IDAM19R3	Problem-State I/O Driver	IDAM19R3 (Packaged in Main Record Management IDA019L1)
IDA121A2	Actual Block Processor	IDA121A2 (Nucleus)
IDA121A3	Normal End Appendage	IDA121A3 (Nucleus)
IDA121A4	Abnormal End Appendage	IDA121A4 (Nucleus)
IDA121A5	Asynchronous Routine	IDA121A5
IDA121A6	Purge Routine	IDA121A6 (Nucleus)
IDA121CV	Communication Vector Table (IEZABP)	IDA121CV (Nucleus)
IGC121	Supervisor-State I/O Driver	IGC121 (Nucleus)

Name	Description	VSAM Modules
Control Block Manipulation		
IDA019C1	Control Block Manipulation	IDA019C1
VSAM Transient Routine		
IFG0192A	VSAM Open/Close/End of Volume Loader	IDAICIA1, IDAOCEA1, IDAOCEA2, IDAOCEA4, IFG0192A
Miscellaneous Routines		
AMDUSRF9	IMDPRDMP format appendage	AMDUSRF9
IDA0195A	VSAM SNAP format routine	IDA0195A
IEFVAMP	AMP parameter interpreter	IEFVAMP

DATA AREAS

"Data Areas" describes a VSAM data set and index and their record formats. "Data Areas" also describes each VSAM control block and shows the relationships between VSAM control blocks. *OS/VS2 VSAM Cross Reference* (microfiche) has a "Symbol Where Used Report" that lists alphabetically all the symbols used in VSAM modules, along with all the modules that use them.

VSAM Data Set Format

A VSAM data set is a collection of records grouped into control intervals. Control intervals are grouped into larger units called control areas. The VSAM stored record, control interval, and control area are described in the sections that follow.

VSAM Record

VSAM records are ordered according to key, in the case of a key-sequenced data set, according to when the records were stored, in the case of an entry-sequenced data set, or according to record numbers that serve as keys, in the case of a relative record data set.

Data records are put in the low-address portion of the control interval. Control information about each data record is put in the high-address portion of the control interval. The combination of a data record and its control information, though they are not physically adjacent, is called a stored record.

In a key-sequenced or entry-sequenced data set, records can be variable in length and can span control intervals. Each segment of a spanned record is stored in its own control interval.

Control Interval

A control interval is a continuous area of auxiliary storage that VSAM uses for storing records. The control interval is the unit of information that VSAM transfers between virtual and auxiliary storage.

The length of each control interval is an integral multiple of block size. The size of a control interval is determined by the system from the size of the records, user-specified minimum buffer size, device characteristics, and the user-specified percentage of free space. The user can specify the size of the control interval, but it must be within limits acceptable to VSAM.

Figure 42 shows the format of a control interval.

When a data set is created, records are put into control intervals.

For an *entry-sequenced data set,* records are ordered according to when they were stored in the data set. The first record to be stored, therefore, has the lowest RBA. A control interval is filled until there is insufficient space in it for the next record. Records are always added at the end of an entry-sequenced data set.

For a *key-sequenced data set,* records are ordered according to key. Records of a key-sequenced data set are put into control intervals; the percentage of free space specified is reserved in each control interval and in each control area for use by records to be added to the data set. As records are added to

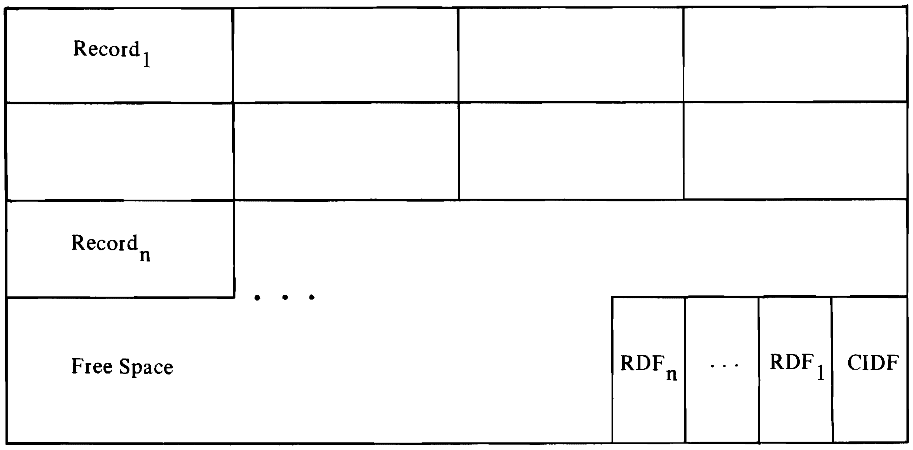


Figure 42. Control Interval Format

the data set, records that have higher keys are moved to higher RBA locations; the free space within the control interval is reduced.

Distributed free space is used to simplify the insertion of records. If there is enough free space in the control interval to accommodate the record to be inserted, higher-keyed records are moved within the control interval to keep the records in key sequence.

If the space needed for directly inserted records is greater than the amount of free space available in a control interval, the control interval is split: VSAM moves some of the stored records (data records and their control information) to an empty control interval in the same control area. For mass insert (sequential insert at the end of a control interval), the percentage of free space defined by the user is maintained.

When a control interval has reached its defined packing factor, a new control interval is obtained. No data is moved.

Note that it is possible for the physical sequence of records to be different from their key sequence after control-interval splits. The sequence will be according to key in each control interval, but the control intervals involved in the split need not be adjacent. Thus, it is possible to have 1-2-3, 4-5-6, 9-10, 7-8 in each of four control intervals. The sequence-set index records, however, reflect the key sequence.

For a *relative record data set,* records are ordered according to their relative record number. Each control interval has as many fixed-length slots as will fit (and allow room for control information). If each control interval has ten slots, the first control interval has slots for relative records 1 through 10, the second for 11 through 20, and so on.

RDF—Record Definition Field

The record definition field (RDF) describes a record, record slot, or record segment within the control interval. RDFs are put into the control interval right to left so that the rightmost RDF describes the leftmost data record. The format of the RDF is:

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

CIDF—Control Interval Definition Field

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

The control interval definition field (CIDF) describes the control interval. The format of the CIDF is:

Offset	Bytes and Bit Pattern	Description
0(0)	2	The displacement from the beginning of the control interval to the beginning of the unused space, or, if there is no unused space, to the beginning of the control information. This number is equal to the length of the data (records, record slots, or record segment). In a control interval without data, the number is 0.
2(2)	2	The length of the unused space. This number is equal to the length of the control interval, minus the length of the control information, minus the 2-byte number in the preceding field. In a control interval without data (records, record slots, or record segment), the number is the length of the control interval, minus 4 (the length of the CIDF—there are no RDFs). In a control interval without unused space, the number is 0.

Control Area

A control area consists of control intervals; the number of control intervals in a control area is determined by VSAM. The control area is the amount of space that VSAM preformats so that data integrity is ensured for records added to a data set.

Control areas are also used to simplify and localize the movement of records when records are inserted in a key-sequenced data set. If an insertion requires a free control interval and there isn't one, a control-area split results. VSAM establishes a new control area and moves the contents of approximately half of the full control area to free control intervals in the new control area. The new records, as their keys dictate, are then inserted into one of the two control areas.

Index Format

There are two types of indexes in VSAM: the *prime index* of a key-sequenced data set, and *alternate indexes* of either a key-sequenced or an entry-sequenced data set.

A key-sequenced data set is a cluster composed of a data component, which contains the control intervals that contain data records, and an index component, which contains the control intervals that contain the records of the prime index.

An alternate index is itself a key-sequenced data set. Its data component contains index records that give the location of data records within its *base cluster* (the key-sequenced or entry-sequenced data set for which it is the alternate index).

Format of Records in a Prime Index

The format of records in the index component of a key-sequenced cluster is fully compatible with the format of VSAM data records; that is, index records, regardless of their level within the index, are treated by Record-Management modules in the same way that any other VSAM record is treated. Each index record and associated control information resides in an index control interval. Figure 43 shows the basic format of an index control interval. The RDF and CIDF fields are described under "Control Interval" earlier in this chapter.

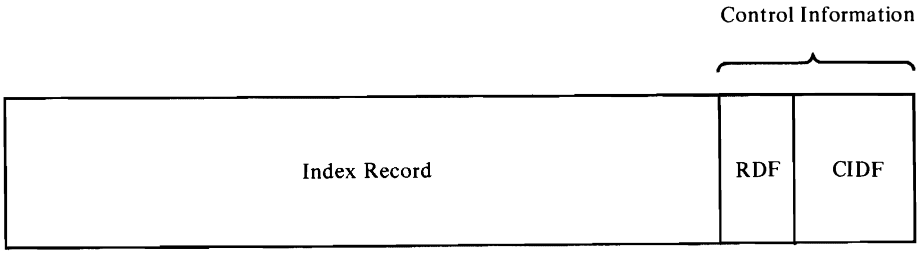


Figure 43. Index Control Interval Format

Figure 44 shows an expansion of the record portion of the index control interval.

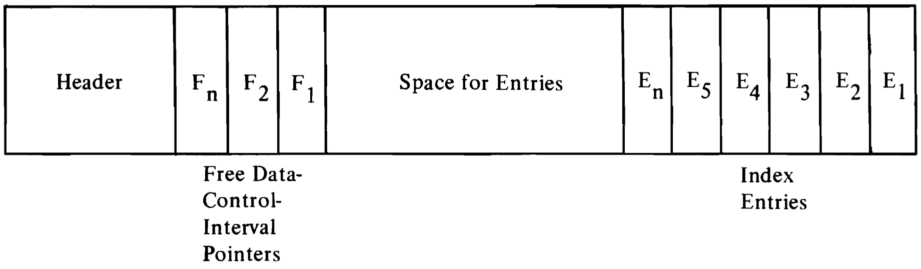


Figure 44. Index Record Format

The header portion of the index record contains, for example, the information required to insert index entries, to locate entries within the index record, and to convert pointers within entries to RBAs. The free data-control-interval pointers are used to locate data control intervals that have not yet been used; these entries exist only in sequence-set index records. Both the index entries and the free data-control-interval pointers are placed in the index record from right to left, as indicated in the figure.

Index entries are grouped into sections. When an index entry is to be located, the search for it begins at the section level. The high-key entry of each section is examined to locate the section that contains the specified entry. VSAM determines the number of sections on the basis of the total number of entries within the index record. Figure 45 shows the index entry portion of the index record divided into sections.

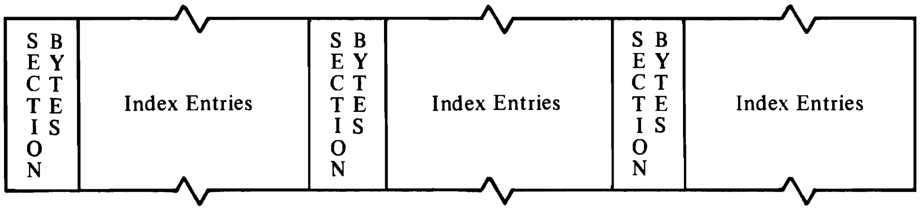


Figure 45. Index Entries Grouped into Sections

The parts of an index record—header, free data-control-interval pointers, and entry sections—are described in the paragraphs that follow.

Index Record Header

The format of the index record header is:

Index Record Header Format

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	2	IXHLL	Length, in bytes, of the index record, including this field. Equals control-interval length minus 7.
2 (2)	1	IXHFLPLN	Length, in bytes, of the control information (the IBFLPF, IBFLPL, and IBFLP3 fields) in each index entry.
3 (3)	1	IXHPTLS	Length of the vertical pointers in this index record. In the sequence set, vertical pointers point to data control intervals; in the index set, they point to index control intervals in a lower level of the index.[1] This field is used as a mask for insert character (store character) under mask instructions that are used to access pointers. The value contained in this field specifies the length of these pointers, as follows: X'01' 1-byte pointer X'03' 2-byte pointer X'07' 3-byte pointer
4 (4)	4	IXHBRBA	For a sequence-set index record, the RBA of a data control area that contains data to be referenced. This RBA and index-entry pointers are used together to calculate the 4-byte RBA of another index record or of a data control interval.
8 (8)	4	IXHHP	Pointer to the logically next index record in this index level. (Horizontal pointer.)
12 (C)	4	IXHXX	Reserved (0).
16 (10)	1	IXHLV	Index level number. A sequence-set index is assigned a value of 1; the next higher-level index is assigned a value of 2; etc.
17 (11)	1	IXHFLGS	Reserved (0).
18 (12)	2	IXHFSO	Displacement from the beginning of this record to the space available for inserting index entries. For higher-level indexes, the entry space immediately follows the record header; for sequence-set indexes, the entry space follows the record header and free data-control-interval pointers.

Index Record Header Format

Offset	Bytes and Bit Pattern	Field Name	Description
20 (14)	2	IXHLEO	Displacement from the beginning of this record to the last (high-key) entry in the index record.[2] (The leftmost entry.)
22 (16)	2	IXHSEO	Displacement from the beginning of this record to the last (high-key) entry in the first section in the index record.[2] (The leftmost entry in the first section.)

[1] Pointers vary in length to conserve index space. For example, if the number of items to be referenced is less than 256, a one-byte pointer is used; if the number is greater than 256, a two-byte pointer is used; and if the number is greater than 65,536, a three-byte pointer is used.

[2] This displacement is to the IBFLPF (front-key compression count) byte of the entry, not to the beginning of the entry.

Free-Data-Control-Interval Pointers

Free data-control-interval pointers, which exist only in sequence-set index records, are used to calculate the RBAs of available data control intervals. The length of a pointer is specified in the record header.

VSAM always uses the rightmost free data-control-interval pointer when a data control interval is needed. The value of the pointer is set to 0 when the control interval is used. As pointers are set to 0, the displacement to space that is available for index entries (contained in the record header) is adjusted by the length of the free data-control-interval pointer. In this way, space used by free data-control-interval pointers is made available for index entries when the pointers are no longer required.

Index Entries

The format of an index entry is:

Length (in Bytes)	Field Name	Description
Variable	IXKEY	Key characters that determine the sequence of records in a key-sequenced data set.
1	IBFLPF	Front-key compression count, that is, the number of characters by which the beginning of the key has been compressed.
1	IBFLPL	Length of the IXKEY field.
1-3	IBPLP3	Pointer to an index or data control interval. The length of the pointer is specified in the record header.

The last (high key) index entry in each index level is a dummy entry: it contains no key characters and the IBFLPF and IBFLPL fields are set to 0. The pointer in this entry is used to calculate the RBA of the last control

Each segment of a spanned record has its own entry in a sequence-set index record. Only the leftmost entry (the entry for the last segment) contains the IXKEY field. In all of the other entries, IBFLPF contains the spanned record's key length, and IBFLPL contains 0.

Index-Entry Sections

Index entries are grouped into sections. A section is defined by a 2-byte field that precedes the high-key index entry. This 2-byte field links a section with a higher-keyed section. This field contains the displacement from the IBFLPF field of the high-key entry in this section to the IBFLPF field of the high-key entry in the next higher-key section. Figure 46 shows how these pointers work. Section 1 indicates the number of bytes between the high-key entry in section 1 and the high-key entry in section 2; section 2 indicates the number of bytes between the high-key entry in section 2 and the high-key entry in section 3; etc.

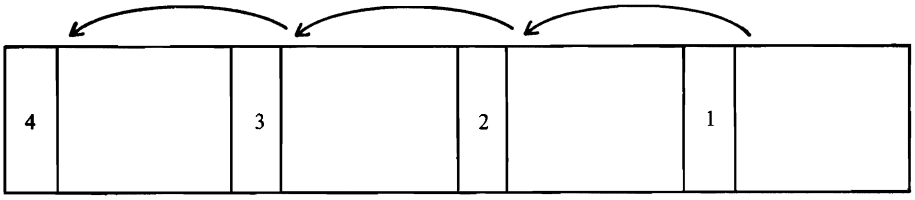


Figure 46. Index-Entry Section Pointers

When the index is searched, the high key of each section is examined to locate the section that contains the specified entry. When the section that contains the entry is found, it is searched.

When an index is originally built, the sections within a record usually contain the same number of entries. As index entries are added and deleted, however, the number of entries per section varies. All of the entries for the segments of a spanned record are grouped into the same section.

Format of Records in an Alternate Index

The index component of an alternate index is the same as the index of any key-sequenced data set. The data component, too, is the same in form: data records, which can be spanned, are stored in control intervals, and control intervals are grouped into control areas.

A data record in an alternate index contains:

- Header information
- A key field that contains the alternate key of the base cluster over which the alternate index is defined
- One or more pointers to data records in the base cluster that contain the alternate key in the alternate-index record's key field

In an alternate index defined with *unique* keys, data records are fixed in length—they contain only one pointer to a base record. In an alternate index defined with *nonunique* keys, data records are variable in length—they can contain more than one pointer to base records.

A pointer to a record in an entry-sequenced base cluster is an *RBA pointer.* It gives the location of the base record by RBA. A pointer to a record in a key-sequenced base cluster is a *prime-key pointer.* That is, it identifies the base record by its prime key. VSAM uses the prime-key pointer to go to the index of the key-sequenced base cluster to find the base record's location.

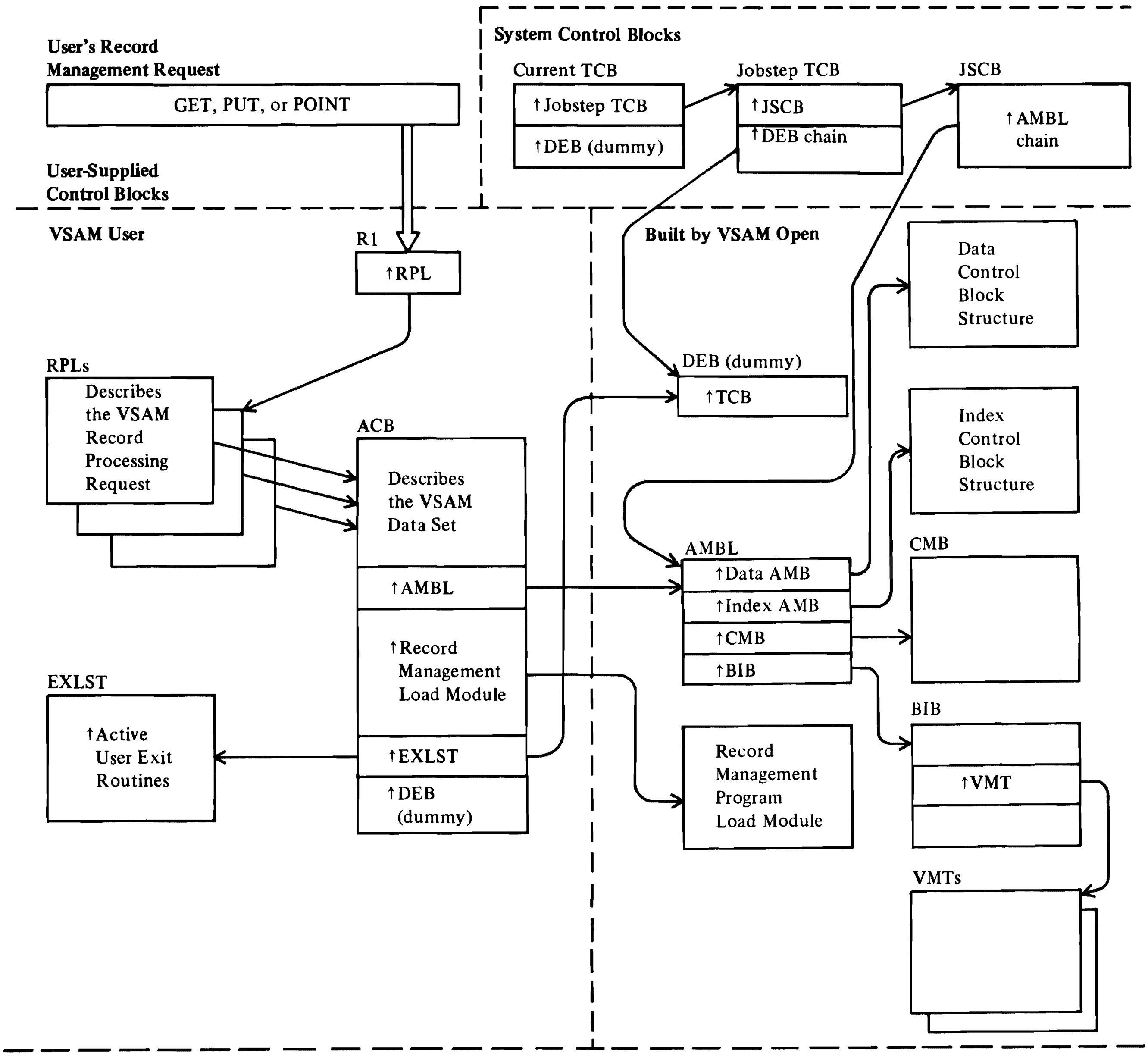
The format of a data record in an alternate index is:

	Bytes and		
Offset	**Bit Pattern**	**Field Name**	**Description**
Header			
0 (0)	1	AIXFG	Flags that indicate what kind of pointer (s) the record contains:
	0		RBA pointer(s).
	1		Prime-key pointer(s).
	xxxx xxx.		Reserved.
1 (1)	1	AIXPL	Length of a pointer. An RBA pointer is four bytes long. A prime-key pointer is as long as the prime key of the key-sequenced base cluster.
2 (2)	2	AIXPC	Number of pointers in the record.
4 (4)	1	AIXKL	Length of the key field in the record. The length is the same as the length of the alternate key field in base records. (That is, the key field in an alternate-index record is *not* compressed.)
Key Field			
5 (5)	VL	AIXKY	The key field of the record. It contains the alternate key of the base record(s) governed by the record.
Pointer(s)			
VL	VL	AIXPT	A pointer to a base record. This field is repeated for each base record that contains the alternate key in AIXKY.

Control Block Interrelationships

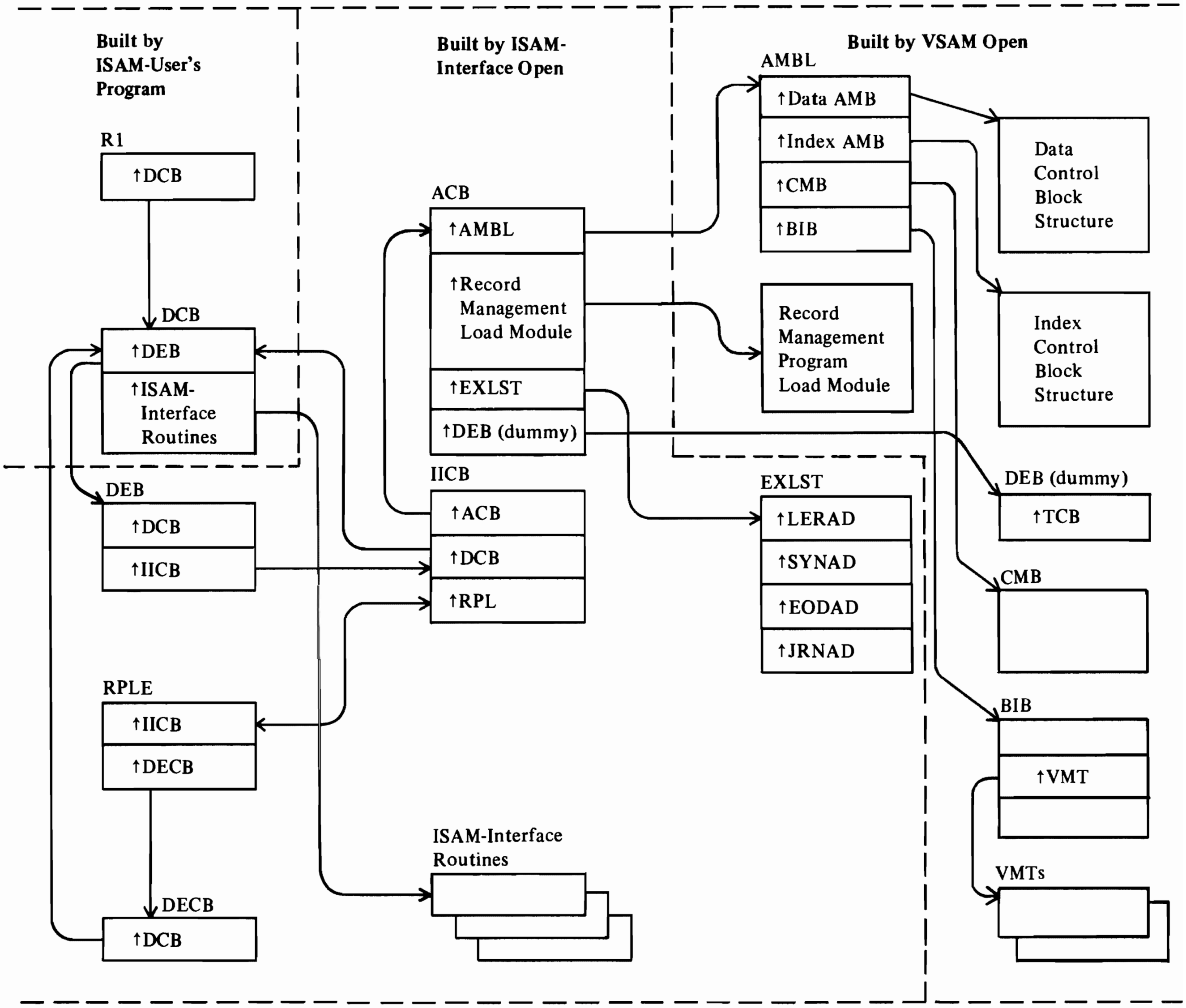
Figure 47 and 48 show the VSAM control blocks built when a key-sequenced data set is opened.

The role of the BIB and CMB in virtual-storage management is described in "Virtual-Storage Management" in "Diagnostic Aids."


User's Record Management Request
GET, PUT, or POINT
User-Supplied Control Blocks
VSAM User
System Control Blocks
Current TCB
↑Jobstep TCB
↑DEB (dummy)
Jobstep TCB
↑JSCB
↑DEB chain
JSCB
↑AMBL chain
Built by VSAM Open
R1
↑RPL
RPLs
Describes the VSAM Record Processing Request
ACB
Describes the VSAM Data Set
↑AMBL
↑Record Management Load Module
↑EXLST
↑DEB (dummy)
EXLST
↑Active User Exit Routines
DEB (dummy)
↑TCB
Data Control Block Structure
Index Control Block Structure
AMBL
↑Data AMB
↑Index AMB
↑CMB
↑BIB
CMB
BIB
↑VMT
VMTs
Record Management Program Load Module

Note: The data control block structure is shown in Figure 52. The index control block structure is shown in Figure 54.

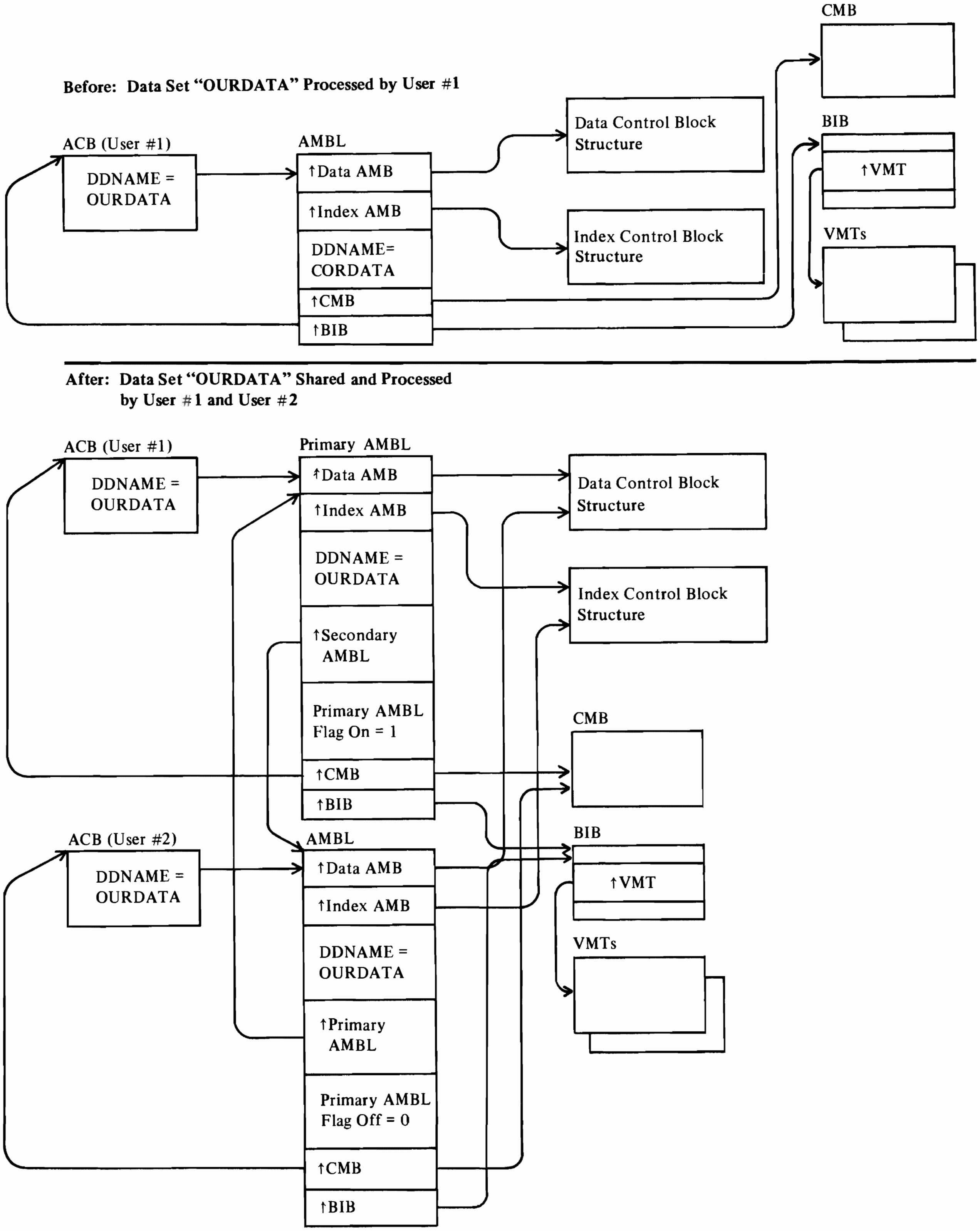
Figure 47. VSAM Control Block Structure for a Key-Sequenced Data Set (VSAM User)



Note: The data control block structure is shown in Figure 52. The index control block structure is shown in Figure 54.

Figure 48. VSAM Control Block Structure for a Key-Sequenced Data Set (ISAM User)

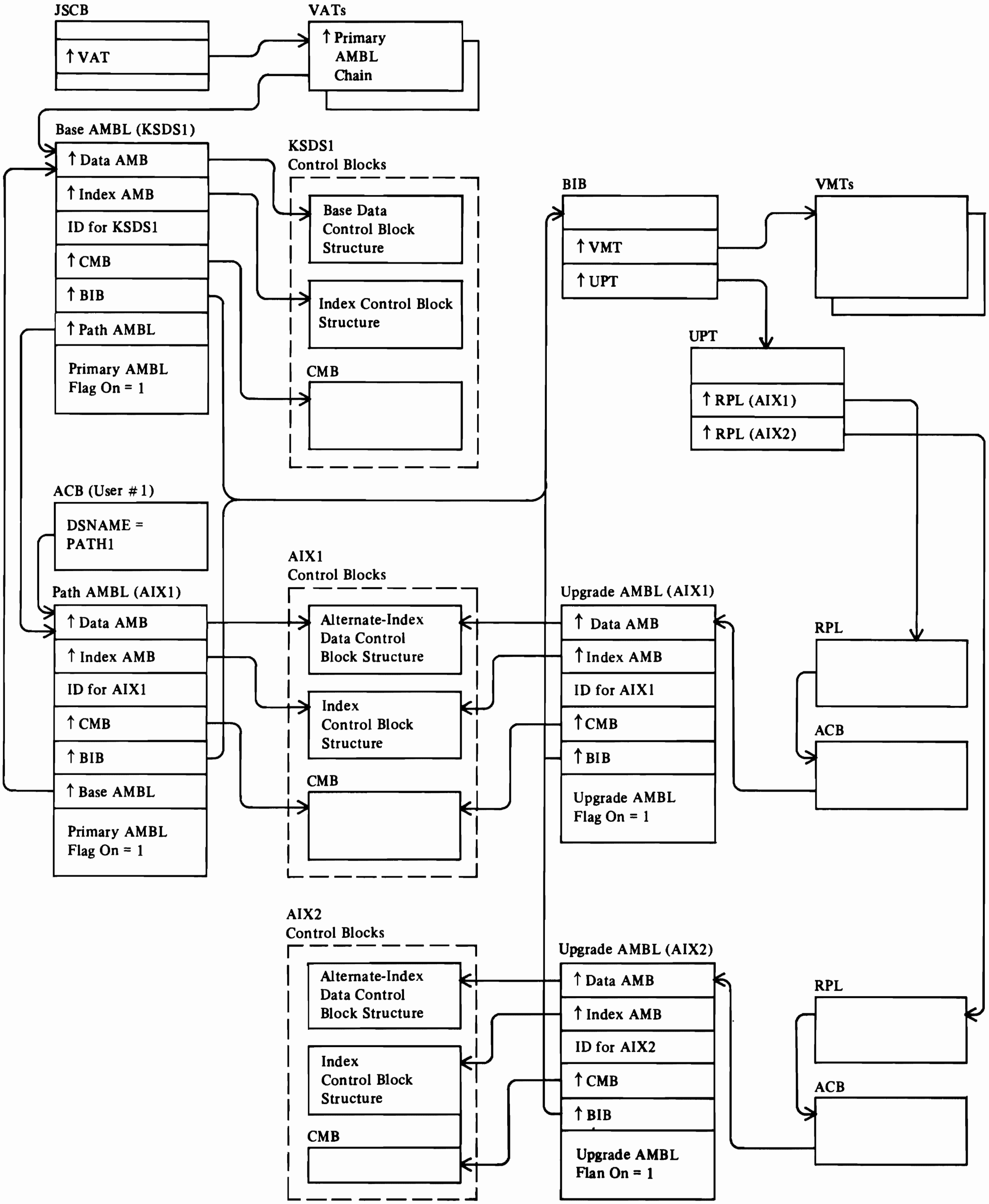
Figure 49 shows how a VSAM cluster (OURDATA) is shared between two subtasks (User#1 and User#2). When the cluster is opened by User#1, VSAM control blocks are built to describe the cluster to VSAM routines. When the cluster is opened by User#2, an AMBL is built to link User#2's ACB to the cluster's VSAM control blocks. When either subtask closes the cluster, the subtask's AMBL is deleted. When the last subtask that is sharing the cluster closes it, the VSAM control blocks that describe the cluster to the VSAM routines are deleted.



Note: The data control block structure is shown in Figure 52. The index control block structure is shown in Figure 54.

Figure 49. VSAM Data Set Control Blocks Before and After Data Set Sharing

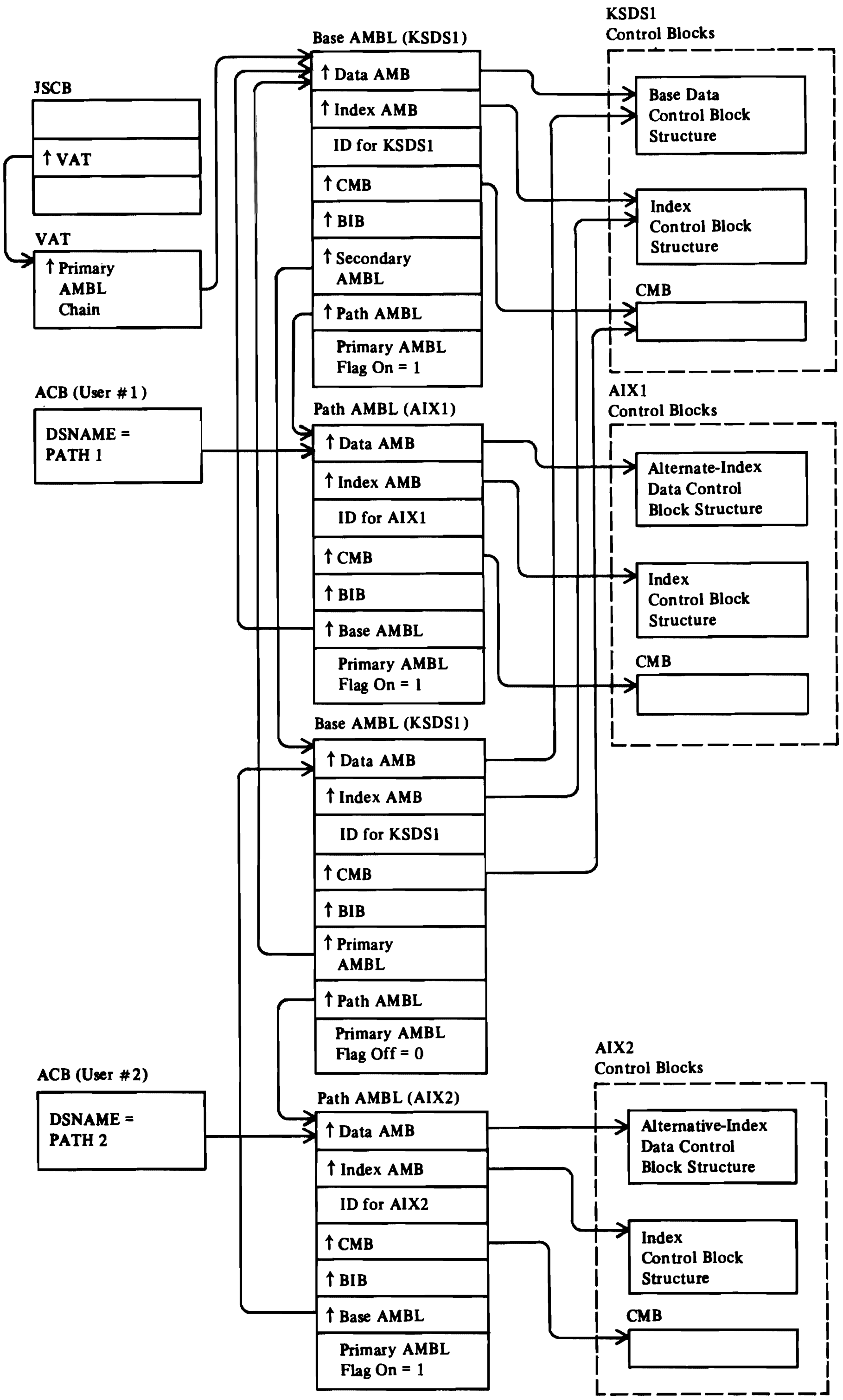
Figure 50 shows the VSAM control blocks built when a key-sequenced data set (KSDS1) is opened for access through a path (PATH1). The path alternate index (AIX1) and a second alternate index (AIX2) are members of the upgrade set for KSDS1.



Note: The base data control block structure is shown in Figure 52. The alternate-index data control block structure is shown in Figure 53. The index control block structure is shown in Figure 54.

Figure 50. VSAM Control Block Structure for a Key-Sequenced Data Set Accessed through a Path

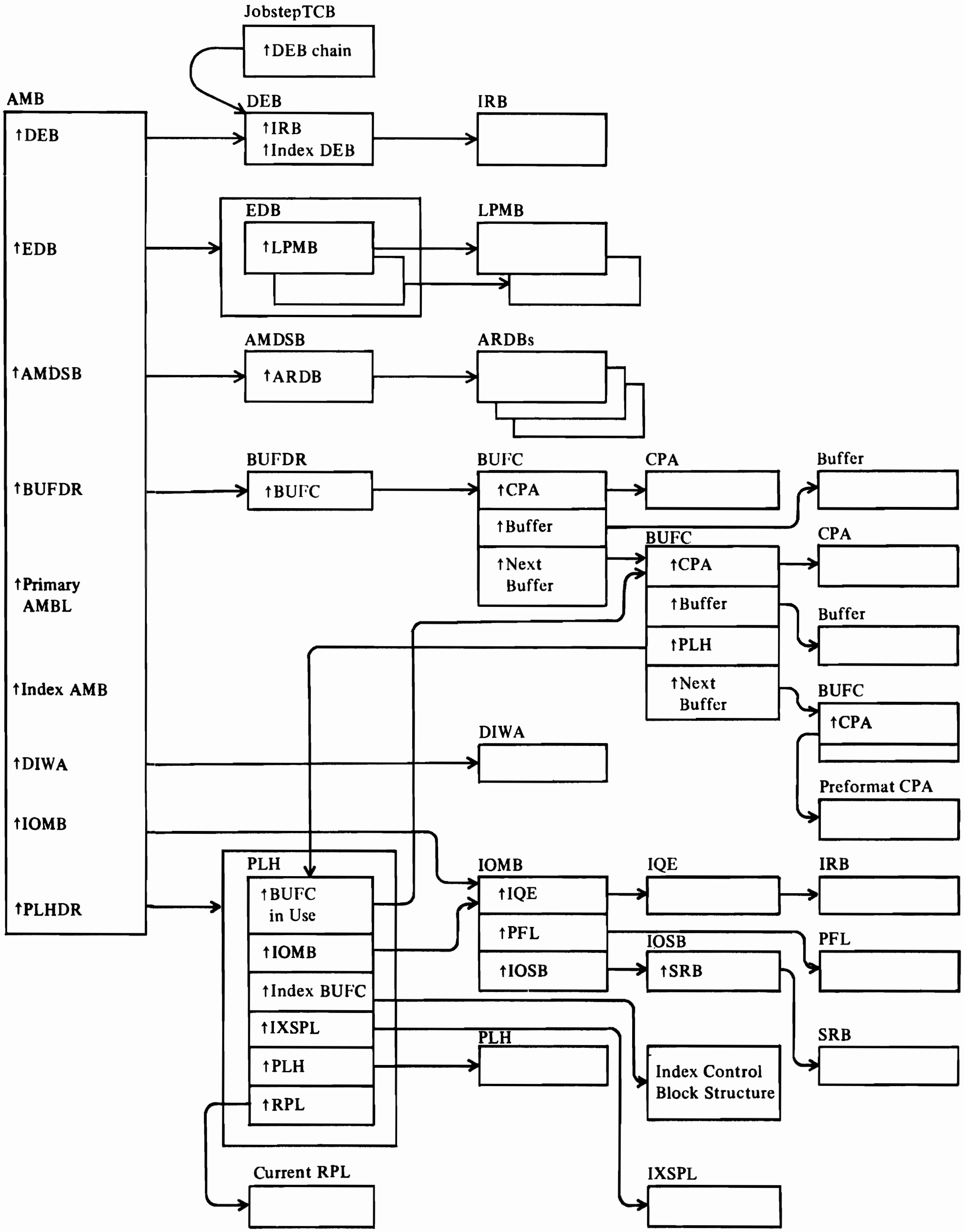
Figure 51 shows the sharing of VSAM control blocks when the key-sequenced data set (KSDS1) shown in Figure 50 is opened for access through a path (PATH2) with the second alternate index (AIX2). AMBLs are built to link User #2's ACB to AIX2 and KSDS1. When either user closes his path, the associated AMBLs are deleted.



Note: The base data control block structure is shown in Figure 52. The alternate-index data control block structure is shown in Figure 53. The index control block structure is shown in Figure 54. The BIB-UPT-RPL-ACB-upgrade AMBL structure (not shown) is the same as in Figure 50.

Figure 51. Shared VSAM Control Block Strucutre for a Key-Sequenced Data Set Accessed through Two Paths

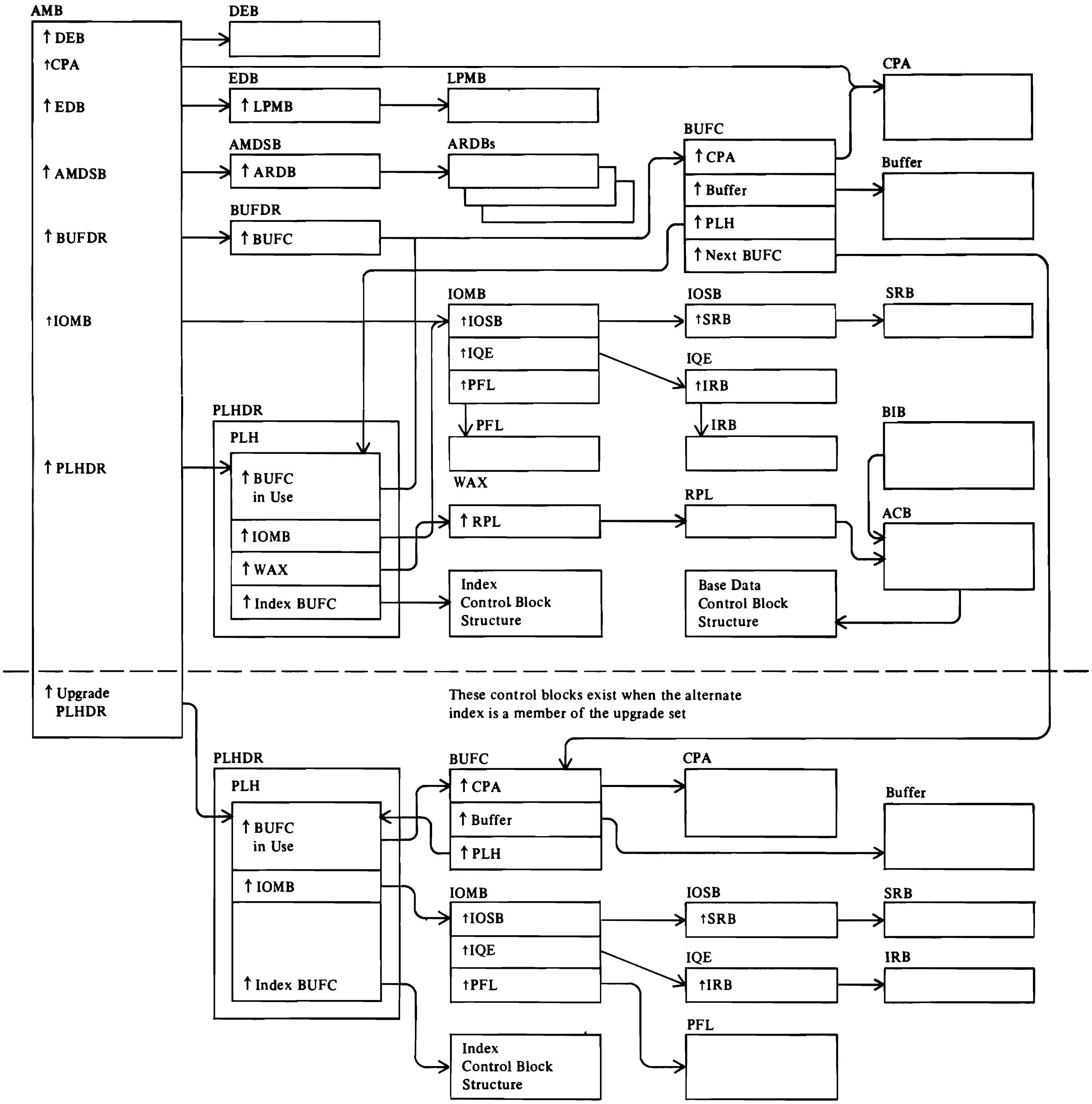
Figure 52 shows the control blocks that describe a cluster's data component set to VSAM Record-Management routines.



Note: The AMBL Control block structure is shown in Figures 49, 50, and 51. The index control block structure is shown in Figure 54.

Figure 52. Data AMB Control Block Structure

Figure 53 shows the control blocks that describe an alternate index's data component to VSAM Record-Management routines.

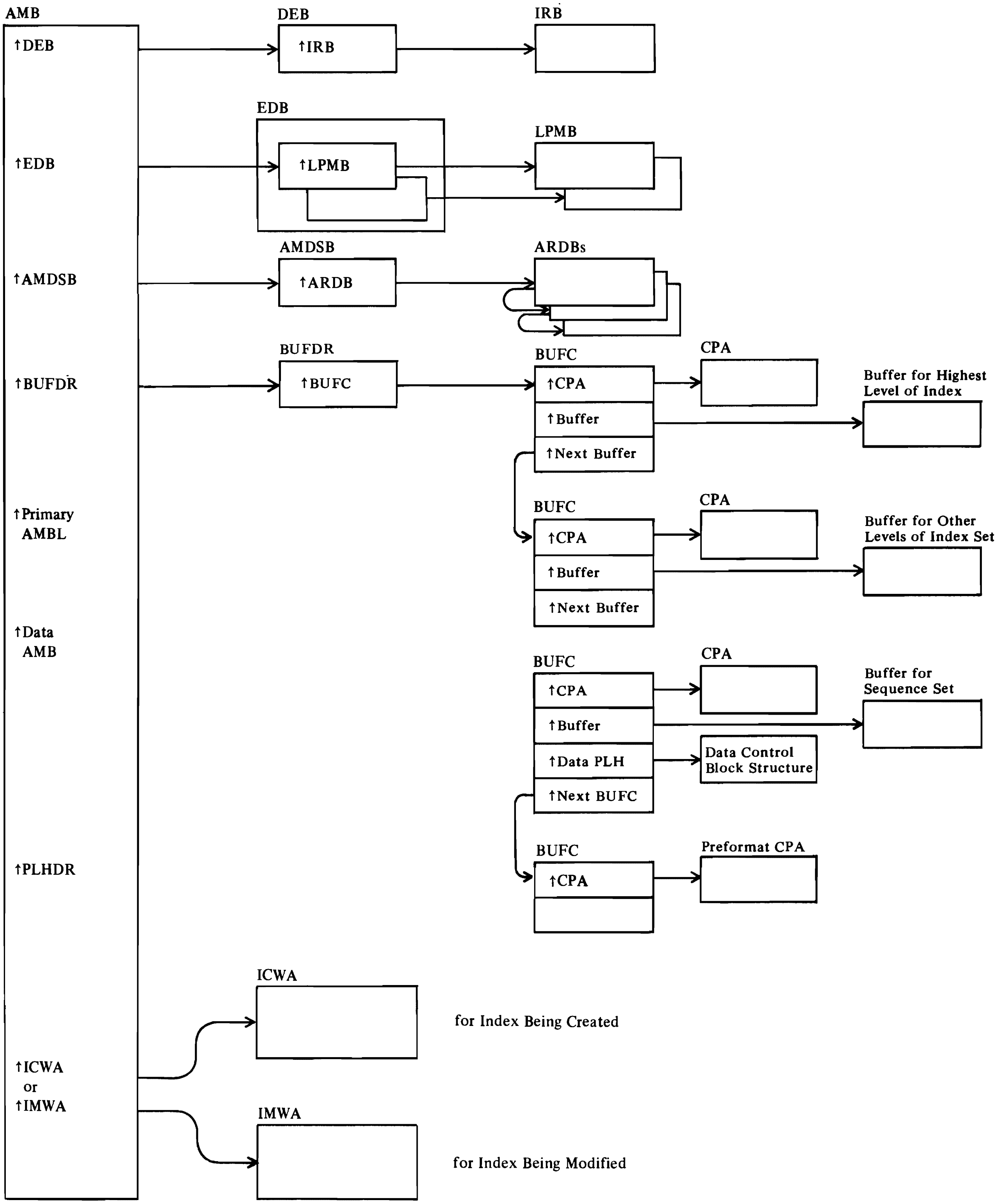


These control blocks exist when the alternate index is a member of the upgrade set

Note: The base data control block structure is shown in Figure 52. The index control block structure is shown in Figure 54.

Figure 53. Alternate-Index Data AMB Control Block Structure

Figure 54 shows the control blocks that describe a key-sequenced cluster's index component to VSAM Record-Management routines.



Note: The AMBL control block structure is shown in Figures 49, 50, and 51. The data control block structure is shown in Figure 52.

Figure 54. Index AMB Control Block Structure

Figure 55 shows the VSAM control blocks built for processing with shared resources. These control blocks describe the VSAM resource pool. With global shared resources (GSR), they are in global storage. With local shared resources (LSR), all of them except the IOSBs, SRBs, and PFLs are in the user's address space. For accessibility to the IOSBs, SRBs, and PFLs in case of a failure in the address space that contains the local resource pool, the ASCB and a chain of VGTTs give their location.

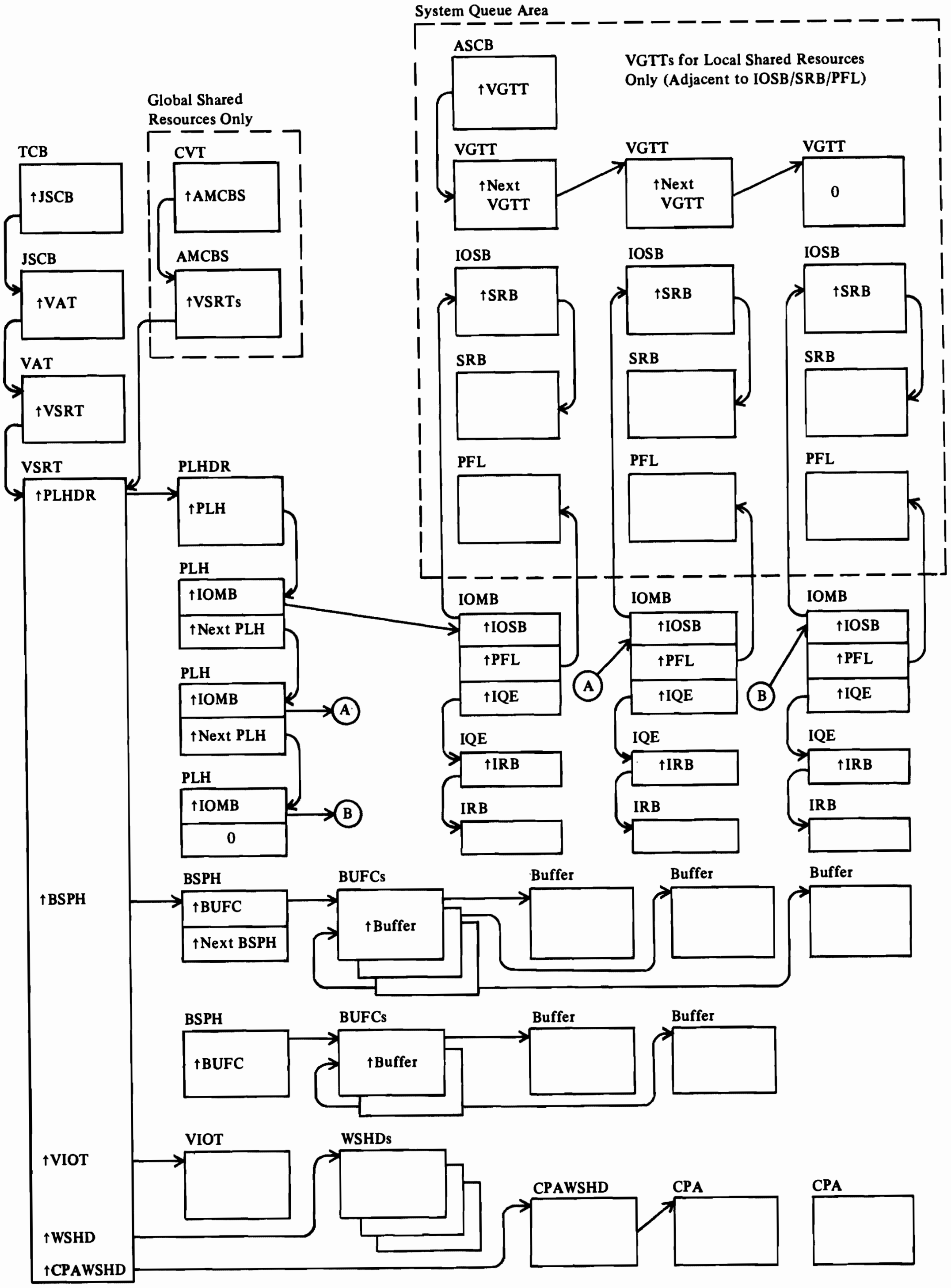


Figure 55. Shared Resources Control Block Structure

Figure 56 shows the AMB control block structure for processing with shared resources. It differs from the structure for processing without shared resources, which is shown in Figures 52, 53, and 54.

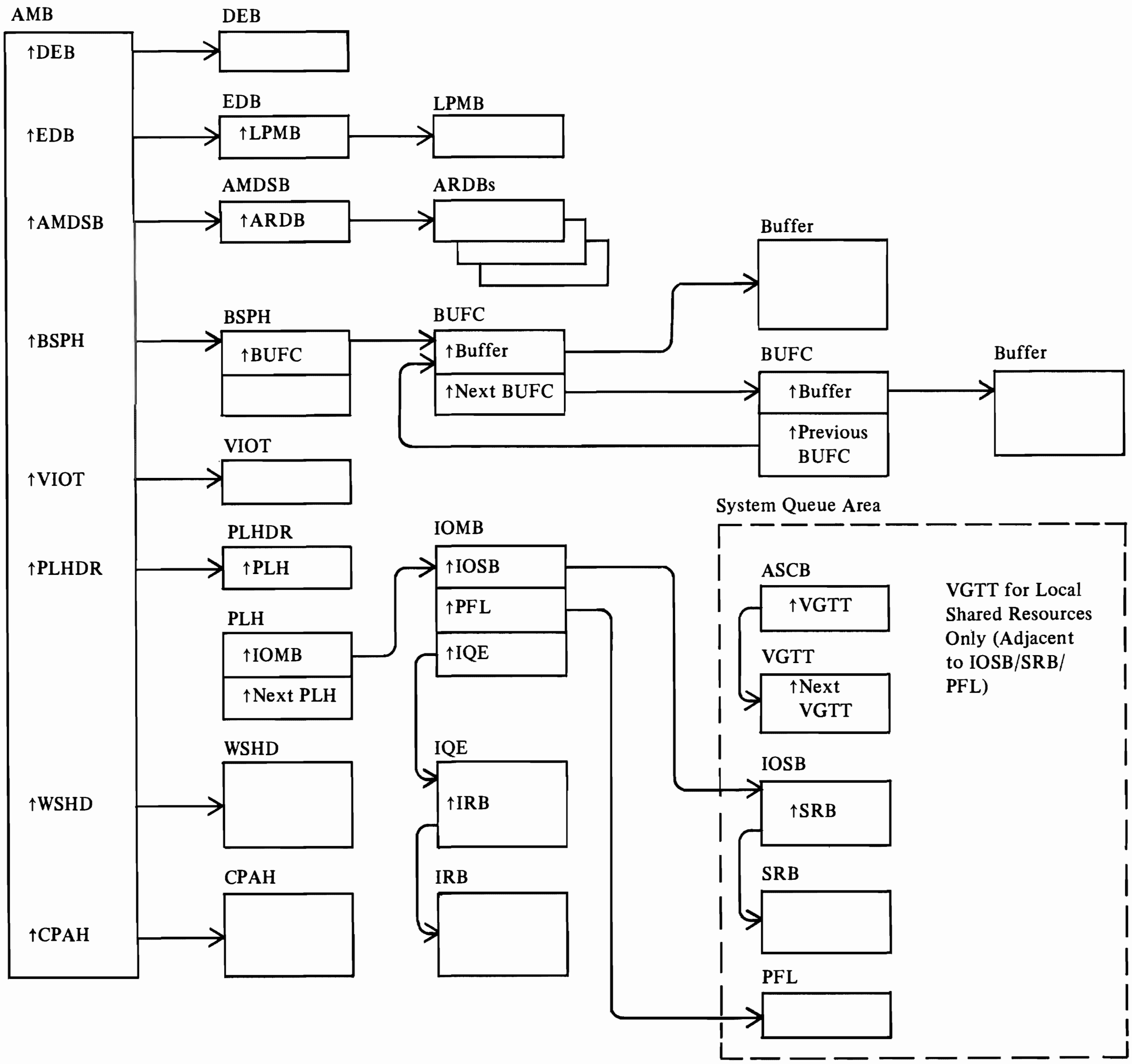


Figure 56. AMB Control Block Structure with Shared Resources

Control Block Subpool Assignment

Subpool 230 in the high end of the user's private address space contains the DEBs, to be consistent with OS/VS I/O Support and Checkpoint/Restart. Subpool 245 in the system queue area (SQA, in global storage) contains the IOSBs and SRBs because I/O Supervisor, running in any user's private address space, needs them in a place where it can always address them. Subpool 252 in the low end of the private area contains the I/O control blocks that must be protected from user alteration (including alteration by Record Management). The AMBXN is *not* in 252 (which has storage protection key of 0) because it contains fields from the AMB and the IOMB that Record Management needs to update. The AMBXN is in subpool 250, along with other unprotected control blocks.

All control blocks for a catalog are kept in global storage, in either Subpools 231 and 241 in the common service area (CSA) or Subpool 245 in SQA.

"Virtual-Storage Management" in "Diagnostic Aids" indicates the subpools in which storage for particular control blocks is indicated.

Control Block Formats

This section discusses VSAM control blocks and (except for those adequately covered in *OS/VS2 Data Areas*) gives their format.

OS/VS2 VSAM Cross Reference (microfiche) has a "Symbol Where Used Report" that lists alphabetically all the symbols used in VSAM modules, in particular the labels of the control blocks discussed here. With each symbol is listed all the modules that use it, along with a code that tells how it is used:

D defined
R read (that is, referenced without alteration)
W written (that is, altered)
C compared

ABP—Actual Block Processor (I/O-Management Communication Vector Table)

The ABP is a communication vector table that contains entry points for I/O Management modules located in the nucleus. It is link-edited in the nucleus as IDA121CV, along with the modules.

The ABP is created by NIP and pointed to by the system CVT (CVTIOBP).

Actual Block Processor (ABP)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	1	ABPID	Control block identifier, X'C1'
1 (1)	1	ABPLEN	Length of the ABP
2 (2)	2	ABPBR14	Unconditional branch, register 14
4 (4)	4	ABPSIQD	Address of the Supervisor-State I/O Driver (IGC121)
8 (8)	4	ABPABP	Address of the Actual Block Processor routine (IDA121A2)
12 (C)	4	ABPNE	Address of the Normal End Appendage (IDA121A3)
16 (10)	4	ABPAE	Adddress of the Abnormal End Appendage (IDA121A4)

ACB—Access Method Control Block

The VSAM ACB describes a VSAM cluster. It is built by the user's program with the ACB or GENCB macro. Before the cluster is opened, the ACB can be modified by the user's DD statements and by the MODCB macro. After the cluster is opened, the ACB is pointed to by the RPL (RPLDACB) that describes the user's record processing request.

Access Method Control Block (ACB)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	1	ACBID	Control block identifier, X‘A0’
1 (1)	1	ACBSTYP	Subtype: X‘10’ = VSAM X‘20’ = VTAM
2 (2)	2	ACBLENG	Length of the ACB
4 (4)	4	ACBAMBL ACBIXLST ACBJWA ACBIBCT	Address of the AMBL Address of the index list
8 (8)	4	ACBINRTN	Address of the VSAM Interface routine (IDA019R1)
12 (C)	2	ACBMACRF	MACRF flags:
		ACBMACR1	MACRF flag byte 1:
	1...	ACBKEY	The record is identified by a key—keyed processing
	.1..	ACBADR ACBADD	The record is identified by a RBA (relative byte address)—addressed processing
	..1.	ACBCNV ACBBLK	Control interval processing
	...1	ACBSEQ	Sequential processing
	.... 1...	ACBDIR	Direct processing
	.... .1..	ACBIN	Input (GET, READ) processing
	.... ..1.	ACBOUT	Output (PUT, WRITE) processing
	1	ACBUBF	User-supplied buffer space
13 (D)		ACBMACR2	MACRF flag byte 2:
	...1	ACBSKP	Skip sequential processing
	.... 1...	ACBLOGON	VTAM LOGON indicator
	.... .1..	ACBRST	Set data set to empty state
	.... ..1.	ACBDSN	Basic subtask shared control-block connection on common DSNAMEs
	1	ACBAIX	Object to be processed is the alternate index of the path specified in the given DDNAME
	xxx.		Reserved
14 (E)	1	ACBBSTNO	Number of concurrent strings for alternate-index path
15 (F)	1	ACBSTRNO	Number of RPL strings
16 (10)	2	ACBBUFND	Number of buffers requested for data
18 (12)	2	ACBBUFNI	Number of buffers requested for index
20 (14)	4	ACBBUFPL	Address of the buffer header (BUFC)

Access Method Control Block (ACB)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
20 (14)	1	ACBMACR3	MACRF flag byte 3:
	.1..	ACBLSR	Local shared resource
	..1.	ACBGSR	Global shared resources
	...1	ACBICI	Improved control-interval access
	.... 1...	ACBDFR	Write operations are to be deferred
	.... .1..	ACBSIS	Sequential insert strategy
	.... ..1.	ACBNCFX	Control blocks are fixed in real storage
	x... ...x		Reserved
21 (15)	1	ACBMACR4	Reserved
22 (16)	2	ACBJBUF	Number of buffers requested for journal
24 (18)	1	ACBRECFM	Record format:
	1...	ACBRECAF	JES format
25 (19)	1	ACBCCTYP	Control character:
	xxxx		Reserved
	 xxxx	ACBASA	Control character type
26 (1A)	2	ACBOPT	Non-user options:
			Byte 1:
	xx..	ACBCROPS	Checkpoint/restart options:
	1...	ACBCRNCK	Restart hasn't checked for modification since last checkpoint
	.1..	ACBCRNRE	Data added since last checkpoint hasn't been erased by restart, and no reposition to last checkpoint takes place
	..xx xxxx		Reserved
			Byte 2:
	.... 1...	ACBDSORG	Match with DCBDSORG
	xxxx .xxx		Reserved
28 (1C)	4	ACBMSGAR	Message area
32 (20)	4	ACBPASSW	Address of the user-supplied password
36 (24)	4	ACBEXLST ACBUEL	Address of the user exit list
Before OPEN			
40 (28)	8	ACBDDNM	DD name
After OPEN			
40 (28)	2	ACBTIOT	Offset to the TIOT
42 (2A)	1	ACBINFL	Indicator flags
43 (2B)	1	ACBAMETH	Access method type
44 (2C)	1	ACBERFL	Error flags
45 (2D)	3	ACBDEB	Address of the DEB
Not Changed by OPEN			
48 (30)	1	ACBOFLGS	Open/Close flags:
	..1.	ACBEOV	EOV concatenation
	...1	ACBOPEN	The ACB is open
	.... 1...	ACBDSERR	No further requests are possible against the ACB
	.... ..1.	ACBEXFG	An ACB Exit routine exists
	1	ACBIOSFG	The Open or Close routine is in control
	xx.. .x..		Reserved

Access Method Control Block (ACB)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
49 (31)	1	ACBERFLG	Error flags **Note:** See "Open and Close Return Codes" in "Diagnostic Aids" for details on the ACBERFLG error flags.
50 (32)	2	ACBINFLG	Indicator flags:
	.1..	ACBJEPS	JEPS processing
	..1.	ACBIJRQE	RQE being held by JAM
	...1	ACBCAT	The ACB describes a VSAM catalog
	.... 1...	ACBSCRA	Catalog recovery area is built in system storage
	.... .1..	ACBUCRA	Catalog recovery area is built in user's storage
	.... ..1.	ACBSDS	A VSAM data set is being opened as a system data set
	x... ...x		Reserved
51 (33)	1		Reserved
52 (34)	4	ACBUJFCB	Address of the user JFCB
56 (38)	4	ACBBUFSP	Amount of space available for the buffers
60 (3C)	2	ACBBLKSZ	Length of the physical DASD record
		ACBMSGLEN	Message length
62 (3E)	2	ACBLRECL	Length of the user's record
64 (40)	4	ACBUAPTR	Address of the user's work area
68 (44)	4	ACBCBMWA	Address of the work area for control block manipulation
72 (48)	4	ACBAPID	Address of application ID

AMB—Access Method Block

The AMB describes a VSAM data set or index and points to control blocks needed to process data set and index records, such as the BUFC, the PLH, the catalog's ACB, and the AMDSB. An AMB is built for a cluster's data set and, if the cluster is key-sequenced, an AMB is built for the index. Each AMB associated with the cluster is pointed to by the AMBL (AMBLDTA points to the data AMB; AMBLIX points to the index AMB). When a data set's or index's record is being processed by VSAM record management, register 3 (RAMB) points to the data set's or index's AMB.

Access Method Block (AMB)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	1	AMBID	Control block identifier, X'40'
1 (1)	1	AMBRSC	Resource TS byte
2 (2)	2	AMBLEN	Length of the AMB
4 (4)	4	AMBLINK	Address of the next AMB in the AMB chain
8 (8)	4	AMBBUFC	Address of the BUFC associated with the AMB
12 (C)	4	AMBPH	Address of the PLH associated with the AMB
16 (10)	4	AMBCACB	Address of the VSAM catalog's ACB (the ACB of the catalog that contains the object's catalog record)

Offset	Bytes and Bit Pattern	Field Name	Description
20 (14)	4	AMBDSB	Address of the AMDSB
24 (18)	1	AMBEOVR	End-of-Volume request type (See AMBXN control block: AMBXEOVR field)
25 (19)	1	AMBFLG1	Indicator flags:
	1...	AMBCREAT	The object is being created
	.0..	AMBTYPE	The AMB describes a data set
	.1..		The AMB describes the index of a key-sequenced data set
	..1.	AMBMCAT	The AMB describes the master catalog
	...1	AMBUCAT	The AMB describes a user catalog
	.... 1...	AMBSPEED	Speed option: Control intervals are not preformatted before the user's data records are written (only applies when the data set is created.)
	.... .1..	AMBUBF	The user's EXLST contains a buffer handling exit routine's address
	.... ..1.	AMBJRN	The user's EXLST contains a journaling exit routine's address
	1	AMBINBUF	The data set is shared—a direct buffer request has been issued
26 (1A)	2	AMBDSORG	Data set organization indicators:
26 (1A)	1		Reserved
27(1B)	1		
	.... 1...	AMBDORGA	VSAM access method
	xxxx.xxx		Reserved
28 (1C)	4	AMBIOBAD AMBIOMB	Address of the IOMB Address of the IOMB
32 (20)	3	AMBCDSN	Data set name of the catalog
35 (23)	3	AMBDDSN	Data set name of the object associated with the AMB
38 (26)	2		Reserved
40 (28)	2	AMBTIOT	Address of the TIOT
42 (2A)	1	AMBINFL	Indicator flags:
	...1	AMBCAT	The AMB describes a catalog
	.... 1...	AMBSCRA	Catalog recovery area is is in system storage
	.... .1..	AMBUCRA	Catalog recovery area is in user's storage
	.... ..1.	AMBUPX	An upgrade table (UPT) exists
	xxx. ...x		Reserved
43 (2B)	1	AMBAMETH	VSAM access method indicator
	...1 ...1	AMBVSAM	VSAM
	xxx. xxx.		Reserved
44 (2C)	1	AMBIFLGS	Error flags
45 (2D)	3	AMBDEBAD	Address of the DEB
48 (30)	1	AMBOFLGS	Open status flags:
	...1	AMBOPEN	The AMB is open
	.... ..1.	AMBEXFG	User exit routines are active
	1	AMBBUSY	Busy bit
	xxx. xx..		Reserved

Offset	Bytes and Bit Pattern	Field Name	Description
49 (31)	1	AMBFLG2	Flag byte 2:
	1...	AMBPUG	The data set described by this AMB is an alternate index in an upgrade set
	.xxx xxxx		Reserved
50 (32)	2	AMBRPT	
52 (34)	4	AMBEDB	Address of the EDB
56 (38)	4	AMBEOVPT	Address of the AMBXN for an End-of-Volume request
60 (3C)	4	AMBWKA	Address of the AMB work area
64 (40)	4	AMBIWA	Address of the DIWA
68 (44)	4	AMBIOBA	Address of the IOB
72 (48)	4	AMBPIXP	Address of the index's AMB
76 (4C)	4	AMBPAMBL	Address of the primary AMBL
80 (50)	4	AMBUPLH	Address of upgrade placeholder
84 (54)	4	AMBCSWD1	
84 (54)	1	AMBAFLG	Flag byte:
	.1..	AMBLSR	Local shared resources
	..1.	AMBGSR	Global shared resources
	...1	AMBICI	Improved control-interval access
	.... 1...	AMBDFR	Defer write operations
	.... .1..	AMBSIS	Sequential insert strategy
	.... ..1.	AMBCFX	Control blocks fixed in real storage
	x... ...x		Reserved
85 (55)	1		Reserved
86 (56)	2	AMBRDCNT	Reserved
88 (58)	4	AMBBM2SH	Reserved
92(5C)	4	AMBCPA	With shared resources: address of the WSHD; without shared resources: address of the first CPA in the chain
96 (60)	4	AMBWSHD	Address of working storage header
100 (64)	8	AMBEXEX	Name of user's exception exit routine
108 (6C)	2	AMBSZRD	Size of the channel program for read
110 (6E)	2	AMBSZWR	Size of the channel program for write
112 (70)	2	AMBSZFW	Size of the channel program for format write
114 (72)	2	AMBSZCP	Size of the CPA base
116 (74)	4	AMBVIOT	Address of the valid-IOMB table

AMBL—Access Method Block List

The AMBL describes a VSAM cluster and points to the cluster's data set and index AMBs. When the cluster is opened, an AMBL is built to describe the cluster. If the cluster's data set (and index) is shared with other users, AMBs already exist for the data set (and index). The existing AMB's addresses are put into the AMBL. If the cluster is not shared, AMBs are built to describe the cluster's data set and, if the cluster is key-sequenced, to describe the data set's index. The AMBL is pointed to by the cluster's ACB (ACBAMBL).

Access Method Block List (AMBL)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	4	AMBLPCHN	Address of the primary AMBL in the AMBL chain
4 (4)	4	AMBLSCHN	Address of the secondary AMBL in the AMBL chain
8 (8)	4	AMBLACB	Address of the ACB associated with the AMBL
12 (C)	1	AMBLEFLG	End of Volume flags:
	1...	AMBLWAIT	End of volume is waiting
	.1..	AMBLESET	End of Volume encountered an error and restored control blocks to their original condition
	..xx xxxx		Reserved
13 (D)	1	AMBLCOMP	End of Volume lock
14 (D)	2		Reserved
16 (10)	8	AMBLDDNM	The ACB's DDNAME field
16 (10)	8	AMBLIDF	Cluster identifier
16 (10)	4	AMBLCACB	Address of the ACB of the catalog
20 (14)	3	AMBLDCI	Control-interval number of the catalog data record
23 (17)	1	AMBLQ	Qualifier:
	1...	AMBLDDC	DD connect only
	.1..	AMBLGSR	Cluster opened for global shared resources
	..1.	AMBLLSR	Cluster opened for local shared resources
	...1	AMBLFSTP	Cluster opened for fast path (improved control-interval access)
	.... 1...	AMBLUBF	Cluster opened for user buffering
	.... .1..	AMBLKSDS	Cluster opened as a key-sequenced data set
	.... ..1.	AMBLESDS	Cluster opened as an entry-sequenced data set
	1	AMBLDFR	Cluster opened for deferred writes
24 (18)		AMBLXPT	In a base AMBL, address of the path AMBL; in a path AMBL, address of the base AMBL
28(1C)	2	AMBLVC	Identifies the entry in the valid-AMBL table that idnetifies this AMBL:
28(1C)	1	AMBLVRT	Number of the valid-AMBL table in the chain of valid-AMBL tables
29(1D)	1	AMBLENO	Offset within the valid-AMBL table

Access Method Block List (AMBL)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
30(1E)	1	AMBLTYPE	Type of control block structure opened:
	1...	AMBLPATH	Path
	.1..	AMBLUPGR	Upgrade set
	..1.	AMBLAIX	Alternate index
	...1	AMBLBASE	Base cluster
	.... 1...	AMBLFIX	Control blocks are fixed in real storage
	xxx		Reserved
31(1F)	1		Reserved
32 (20)	1	AMBLID	Control block identifier, X'50'
33 (21)	1	AMBLSHAR	Sharing indicators:
	1...	AMBLPRIM	Identifies the primary AMBL
	.1..	AMBLCATO	The catalog is open
	..1.	AMBLWRIT	The user intends to write or update records in the data set
	...x xxxx		Reserved
34 (22)	1	AMBLLEN	Length of the AMBL
35 (23)	1	AMBLFLG1	Flags:
	1...	AMBLFULL	The user-supplied master password was verified
	.1..	AMBLCINV	The user-supplied control-interval password was verified
	..1.	AMBLUPD	The user-supplied update password was verified
	...1	AMBLVVIC	The AMBL is for the mass storage volume inventory (MSVI) data set
	.... 1...	AMBLSCRA	The AMBL is for a catalog recovery area in system storage
	.... .1..	AMBLUCRA	The AMBL is for a catalog recovery area in user's storage
	.... ..1.	AMBLCAT	The AMBLACB field points to a catalog's ACB
	1	AMBLDUMY	A DD DUMMY statement was specified
	...x x.x.		The combination of these bits indicates the type of data set: 001 Catalog 101 MSVI 011 SCRA
36 (24)	1	AMBLFLG2	Flags:
	...1....	AMBLSTAG	Cluster is staged
	xxx. xxxx		Reserved
37 (25)	1	AMBLNST	Number of strings
38 (26)	2	AMBLNUM	Number of AMB pointers in the AMBL
40 (28)	1		Reserved
41 (29)	1	AMBLNIDS	Number of identifiers
42 (2A)	10	AMBLMIDS	Five 2-byte fields, each containing a VSAM module's identifier
52 (34)	4	AMBLDTA	Address of the cluster's data set AMB
56 (38)	4	AMBLIX	Address of the cluster's index AMB
60 (3C)	4	AMBLBIB	Address of the base information block
64 (40)	4	AMBLCMB	Address of the cluster management block

AMBXN—Access Method Block Extension

The AMB and the IOMB are kept in a protected subpool with key 0. They cannot be changed by the user or by Record Management. Record Management needs to store information in the AMB and the IOMB. Some of their fields have been moved to the AMBXN. It contains one set of fields for the AMB and one set for each IOMB associated with the AMB.

Access Method Block Extension (AMBXN)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
AMB Portion			
0(0)	8	AMBXEOV	End-of-Volume Interface fields (IDAEOVIF)
0(0)	4	AMBXEVPT	Address of the key or RBA to be used by End of Volume
4(4)	1	AMBXRSC	Resource TS byte
5(5)	1	AMBXEOVR	End-of-Volume request type:
	X‘80’		Page-space data set
	X‘01’		Mount by key
	X‘81’		Mount by RBA
	X‘02’		Allocate by key
	X‘82’		Allocate by RBA
6(6)	2		Reserved
8(8)	8	AMXRMWA	Reserved
16(10)	4	AMBXCSWD	Used by Buffer Management for shared resources serialization
16(10)	2		Unused
18(12)	2	AMBXRDCT	Number of control intervals read
20(14)	4	AMBXBM2S	Address of the PLH being used for a second search of a subpool
IOMB Portion			
0(0)	1	IOMXLOCK	Resource TS byte
1(1)	1	IOMXFLGS	I/O-Management flags:
	1...	IOMXUSE	Error processing is complete
	.xxx xxxx		Reserved
2(2)	2		Reserved
4(4)	14	IOMXPDET	Problem-determination fields
4(4)	2	IOMXBFLG	I/O Flags at I/O initiation
6(6)	2		Reserved
8(8)	4	IOMXR13S	Address of the user's save area
12(C)	4	IOMXRPL	Address of the first RPL in a chain
16(10)	4		Reserved
20(14)	4	IOMXRECB	Record Management I/O ECB
20(14)	1	IOMXECB	I/O ECB:
	1...	IOMXWAIT	I/O wait bit
	.xxx xxxx	IOMXRSLT	I/O completion result
	.1..	IOMXPOST	I/O post bit
	..xx xxxx	IOMXIOCC	I/O completion code
21(15)	3	IOMXRBPT	Pointer to RB (request block) or result

AMDSB—Access Method Data Set Statistics Block

The AMDSB contains statistical information about record processing in the data set. It also contains some of the data set's attributes and specifications. The AMDSB is built, using the data set or index catalog record's AMDSB set of fields, when the cluster is opened. The data or index AMB (AMBDSB) points to its associated AMDSB.

Access Method Data Set Statistics Block (AMDSB)—Description and Format

	Bytes and		
Offset	**Bit Pattern**	**Field Name**	**Description**
0 (0)	1	AMDSBID	Control block identifier, X'60'
1 (1)	1	AMDATTR	Attributes of the data set:
	1...	AMDDST	Key-sequenced data set
	0...		Entry-sequenced data set
	.1..	AMDWCK	Check each record when it is written
	..1.	AMDSDT	Sequence set is stored with the data and replicated
	...1	AMDREPL	All index records are replicated
	.... 1...	AMDORDER	Use the volumes in the same order as in the volume list
	.... .1..	AMDRANGE	The data set is divided into key ranges
	.... ..1.	ANDRRDS	Relative record data set
	1	AMDSPAN	The data set contains spanned records
2 (2)	2	AMDLEN	Length of the AMDSB
4 (4)	2	AMDNEST	Number of index entries in the index section
		AMDAXRKP	Relative key position of the alternate key
6 (6)	2	AMDRKP	Relative key position
8 (8)	2	AMDKEYLN	Key length
10 (A)	1	AMDPCTCA	Percentage of free control intervals in the control area
11 (B)	1	AMDPCTCI	Percentage of free bytes in the control interval
12 (C)	2	AMDCIPCA	Number of control intervals in a control area
14 (E)	2	AMDFSCA	Number of free control intervals in a control area
16 (10)	4	AMDFSCI	Number of free bytes in a control interval
20 (14)	4	AMDCINV	Control interval size
24 (18)	4	AMDLRECL	Maximum record size
28 (1C)	4	AMDHLRBA	Relative byte address (RBA) of the high-level index record
		AMDNSLOT	Number of record slots per control interval
32 (20)	4	AMDSSRBA	Relative byte address (RBA) of the first sequence-set record
		AMDMAXRR	Maximum valid relative record number
36 (24)	4	AMDPARDB	Address of the first ARDB
40 (28)	56	AMDSTAT	Data set statistics:
40(28)	1	AMDATTR3	Attributes of the data set:
	x...	AMDUNQ	The data set has: 0 Unique keys 1 Nonunique keys
	.x..	AMDFAULT	The data set is staged: 0 At open time, if required 1 By cylinder fault
	..x.	AMDBIND	The data set is:

Offset	Bytes and Bit Pattern	Field Name	Description
			0 Not bound 1 Staged and bound
	...x	AMDWAIT	After destaging is begun, control is returned to the program that is closing the data set: 0 Immediately 1 After destaging is finished
	.... x...	AMDLM	0 Load mode, or data set is not loaded 1 Data set is loaded
	xxx		Reserved
41(29)	7		Reserved
48 (30)	8	AMDSTSP	OS/VS system timestamp
56 (38)	2	AMDNIL	Number of index levels
58 (3A)	2	AMDNEXT	Number of extents in the data set
60 (3C)	4	AMDNLR	Number of user-supplied records in the data set
64 (40)	4	AMDDELR	Number of deleted records
68 (44)	4	AMDIREC	Number of inserted records
72 (48)	4	AMDUPR	Number of updated records
76 (4C)	4	AMDRETR	Number of retrieved records
80 (50)	4	AMDASPA	Number of bytes of free space in the data set
84 (54)	4	AMDNCIS	Number of times a control interval was split
88 (58)	4	AMDNCAS	Number of times a control area was split
92 (5C)	4	AMDEXCP	Number of times EXCP was issued by VSAM I/O routines

ARDB—Address Range Definition Block

The ARDB contains information about space allocated to and space actually used by a data set. The block is built by the VSAM Open routine from information in the data set's catalog record. The number of ARDBs depends on whether the data set is divided into key ranges (one ARDB per range, or one for a data set without ranges) and whether the sequence set of the index is placed adjacent to data.

The ARDB is updated by Record-Management routines as additional space is used. The first ARDB in an ARDB chain is pointed to by the AMDSB (AMDPARDB).

Address Range Definition Block (ARDB)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	1	ARDID	Control block identifier, X'61'
1 (1)	1	ARDTYPE	Identifies the type of space defined by the ARDB:
	1...	ARDKR	One key range of a key-range data set
	.1..	ARDHLI	The total index of a key-sequenced data set, or The non-sequence set levels of a key-sequenced data set's index, when the sequence set is stored with the data
	..1.	ARDSS	The sequence set of a key-sequenced data set, when the sequence set is stored with the data
	...1	ARDUOVFL	Overflow may be used for this key range
	.... 1...	ARDEOD	The key range containing the highest data RBA in the data set
	.... .1..	ARDUSED	The ARDHRBA field's initial value has changed
	xx		Reserved
2 (2)	2	ARDLEN	Length of the ARDB
4 (4)	4	ARDNPTR	Address of the next ARDB in the ARDB chain
8 (8)	4	ARDHKRBA	The RBA of the data set control interval containing the key range's high-key value
12 (C)	4	ARDHRBA	The RBA of the next free-space control interval at the end of the data set
16 (10)	4	ARDERBA	The RBA of the highest control interval allocated to the key range
20 (14)	6	ARDVOLSR	The serial number of the volume containing the highest RBA allocated to the key range
26 (1A)	2	ARDRELNO	The sequence number of the Data Space Group set of fields that describes the data space containing the key range—the Data Space Group set of fields is in the volume catalog record identified by ARDVOLSR
28 (1C)	1	ARDPRF	Preformat flags:
	1...	ARDPRSS	The sequence set is stored with the data
	.1..	ARDPRFMT	The key range's extents haven't been preformatted
	..xx xxxx		Reserved
29 (1D)	VL	ARDKEYS	The key range's low and high key values—the length of this field equals twice the key length

BIB—Base Information Block

The BIB contains information for Virtual-Storage Management to control allocation of storage for a particular base cluster in a job step. It is further described in "Virtual-Storage Management" in "Diagnostic Aids."

The BIB is pointed to by the AMBL (AMBLBIB).

Base Information Block (BIB)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	4	BIBHDR	Header:
0 (0)	1	BIBID	Control block identifier, X'10'
1 (1)	1	BIBFLG1	Flag byte 1:
	1...	BIBVIRT	At least one mass-storage UCB is allocated
	.1..	BIBREST	Restart in progress for sphere
	..1.	BIBCKPT	Checkpoint in progress for sphere
	...x xxxx		Reserved
2 (2)	2	BIBLEN	Length of the BIB
4 (4)	1	BIBFLG2	Reserved
5 (5)	3		Reserved
8 (8)	4	BIBUPT	Address of the upgrade table
12 (C)	4	BIBVMT	Address of the volume mount table
16 (10)	4	BIBDACB	Address of an inner ("dummy") ACB
20 (14)	4	BIBAMBL	Address of the primary AMBL in the AMBL chain
24 (18)	4	BIBSPHPT	Address of the sphere block header
28 (1C)	4	BIBPRSPH	Address of the protected sphere block
32 (20)	4	BIBHEBPT	Address of the header element block
36 (24)	4	BIBHEBFQ	Address of the first free header element in the header element block
40 (28)	4	BIBVCRT	Address of the VSAM checkpoint/restart table
44 (2C)	4	BIBWSHD	Address of the working storage header
48 (30)	4	BIBCSL	Address of the first core save list in the chain
52 (34)	4	BIBPSAB	Address of the protected sphere AMBL block
56 (38)	4	BIBVGTT	Address of the VSAM global termination table for the control blocks stored in global storage for a base cluster and its related clusters
60 (3C)	16	BIBRTNS	Addresses of Record-Management routines:
60 (3C)	4	BIBINTRF	VSAM interface (IDA019R1)
64 (40)	4	BIBCEAPP	Channel end appendage
68 (44)	4	BIBASYRT	Asynchronous Routine
72 (48)	4	BIBSIOAP	Start-I/O appendage
76 (4C)	8	BIBJOBNM	Name of the job that issued OPEN for the base cluster
84 (54)	8	BIBSTPNM	Name of the job step that issued OPEN for the cluster
92 (5C)	8	BIBDDNM	Name of the DD statement specified for OPEN

Base Information Block (BIB)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
100 (64)	4	BIBASCB	Address of the address space control block for the address space that issued OPEN

BLPRM—Resource Pool Parameter List

BLPRM is created by the BLDVRP and DLVRP macros. It is used by Record Management for dynamic string addition and by data set management (O/C/EOV and Checkpoint/Restart) for internal processing. BLPRM is mapped by IDABLPRM and pointed to by the parameter list whose address is in register 1 when SVC 19 is issued.

Resource Pool Parameter List (BLPRM)—Description and Format

Offset	Bytes and Bit Patterns	Field Name	Description
0 (0)	1	BLPACBID	ACB ID—X'A0'
1 (1)	1	BLPACBST	ACB subtype—X'11'
2 (2)	2		Reserved
4 (4)	4	BLPBUFLP	Address of the buffer list used by BLDVRP (described below)
	4	BLPUACB	Address of the user ACB (used for dynamic string addition)
	4	BLPIOPLH	Address of the I/O Support PLH (used for CLOSE)
8 (8)	1	BLPKEYLN	Key length
9 (9)	1	BLPSTRNO	String number requests
10 (A)	1	BLPFLAG1	Flag byte 1:
	1...	BLPFDBDC	Shared resources
	.1..	BLPFBLD	BLDVRP request
	..1.	BLPFDEL	DLVRP request
	...1	BLPFLSR	LSR option
	.... 1...	BLPFGSR	GSR option
	.... .1..	BLPFIOBF	Fix IOBs
	.... ..1.	BLPFBFRF	Fix buffers
	1	BLPFSTAD	Add String
11 (B)	1	BLPFLAG2	Flag byte 2 (used for I/O support internal processing):
	1...	BLPFPART	Partial build request
	.1..	BLPFUPGR	Upgrade set Open
	..1.	BLPFPATH	Path (AIX) Open
	...1	BLPFPRIM	Primary Open
	.... 1...	BLPFDATA	Data AMB
	.... .1..	BLPFINDX	Index AMB
	.... ..1.	BLPFIOSR	I/O support request
	1	BLPFRSTR	Restart request
12 (C)	1	BLPOCODE	Special use field
13 (D)	3	BLPOACB	Address of ACB
16 (10)	8	BLPCORE	Record management GETCORE request
16 (10)	1	BLPGFLG	Flag byte:
	1...	BLPGREQ	GETCORE request
	.1..	BLPGPG	GETCORE page boundary request
	..xx xxxx		Reserved

Resource Pool Parameter List (BLPRM)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
17 (11)	3	BLPGSZ	GETCORE length
20 (14)	1	BLPGSP	GETCORE subpool
21 (15)	3	BLPGAD	GETCORE return address
24 (18)	4	BLPIOACB	Address of I/O support ACB
24 (18)	3		Reserved
27 (1B)	1	BLPDSORG	X'08' (required for BLDVRP, DLVRP, and string addition)
28 (1C)	20		Reserved
48 (30)	1	BLPOFLGS	X'02'
49 (31)	2		Reserved
51 (33)	1	BLPERFLG	X'00'

The buffer request list (pointed to by BLPBUFLP) is repeated once for each buffer pool. The format is:

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	4	BLPBUFSZ	Buffer size
4 (4)	1	BLPBRLFG	Buffer list flags:
	1... .... .xxx xxxx	BLPBFLST	Last buffer request Reserved
5 (5)	1		Reserved
6 (6)	2	BLPBFLCT	Buffer count

BSPH—Buffer Subpool Header

The BSPH is built for processing with shared resources. It defines a buffer pool in the VSAM resource pool. The first BSPH for the resource pool is pointed to by the VSRT (VSRTBUFH). Each BSPH is pointed to by an AMB (AMBBUFC) that uses the buffer pool defined by the BSPH.

Buffer Subpool Header (BSPH)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	1	BSPHID	Control block identifier, X'72'
1 (1)	1	BSPHFLG1	Flag byte 1:
	1...	BSPHIOBF	I/O-related control blocks are fixed in real storage
	.1..	BSPHBFRF	I/O buffers are fixed in real storage
2 (2)	2	BSPHLEN	Length of the BSPH
4 (4)	4	BSPHNM	Visual name: 'BSPH'
8 (8)	4	BSPHNBSP	Address of the next BSPH for the resource pool
12 (C)	2	BSPHBFNO	Number of buffers in the buffer pool
14 (E)	2	BSPHERCT	Count of write errors
16 (10)	4	BSPHBUFC	Address of the first BUFC for the buffers in the pool
20 (14)	4	BSPHMDBT	Modification bits—they indicate IDs of transactions that have modified the buffer (RPL TRANSID operand)
24 (18)	4	BSPHBSZ	Length of each buffer in the pool

Buffer Subpool Header (BSPH)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
28 (1C)	4	BSPHCSRC	Compare/Swap resource—used to serialize the use chain:
28 (1C)	1	BSPHFLG2	Flag byte 2:
	1...	BSPHAPRT	Arithmetic protect bit
	.1..	BSPHPCUC	The use chain is being changed
	..xx xxxx		Reserved
29 (1D)	1		Reserved
30 (1E)	2	BSPHPSUC	Number of PLHs searching the use chain
32 (20)	4	BSPHCPLH	Address of the PLH that is modifying the use chain
36 (24)	4	BSPHRDS	Number of I/O operations to bring data into the buffer pool
40 (28)	4	BSPHFND	Number of requests for retrieval that could be satisfied without an I/O operation
44 (2C)	4	BSPHUIW	Number of user-initiated writes from the buffer pool
48 (30)	4	BSPHNUIW	Number of non-user-initiated writes (writes that VSAM was forced to do because no buffers were available)
52 (34)	4	BSPHUTOP	Address of the top of the use chain
56 (38)	4	BSPHUBTM	Address of the bottom of the use chain
60 (3C)	4	BSPH1ST	Address of the first BSPH for the resource pool

BUFC—Buffer Control Block

The BUFC consists of a buffer header that describes the buffer pool and a buffer control entry that describes each buffer requested by the user and each buffer required for preformat processing. The header describes the structure of the buffer pool. Each buffer control entry contains function codes, status indicators, and RBAs to describe the buffer. The buffer control entry also contains the address of its associated placeholder (PLH), the data buffer, the associated channel program (pointed to by the CPA), and the next BUFC in the chain.

Index and data have separate blocks of BUFCs. At the end of each block are BUFCs used for preformat processing—they are pointed to by a field in the header.

The BUFC is the interface between I/O Management and Buffer Management (IDA019R2 and its procedures). The BUFC is pointed to by the PLH (PLHBUFC points to the data BUFC; PLHIBUFC points to the index BUFC).

Both the buffer header and the buffer control entry are created by Open and released by Close. The AMB points to the buffer header. The DIWA points to the insert buffer control entry, and each placeholder points to a chain of one or more data buffer control entries and one index buffer control entry.

Buffer Control Block (BUFC)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
Buffer Header			
0 (0)	1	BUFDRID	Buffer-header identifier, X'70'
1 (1)	1	BUFDRNO	Number of buffer control entries in this buffer pool, excluding preformat buffer control entries
2 (2)	2	BUFDRLEN	Length of the buffer header and all buffer control entries associated with this buffer pool
4 (4)	4	BUFDRPFB	Pointer to the first BUFC in a pool of BUFCs that are reserved for preformatting data control areas or index tracks
8 (8)	1	BUFDRPFN	Number of preformat BUFCs
9 (9)	1	BUFDRCIX	Number of index buffers in an index buffer pool that are not assigned to a placeholder and are not reserved for the highest level index record
		BUFDRMAX	Maximum number of buffers that can be assigned to a placeholder that is in sequential mode
10 (A)	1	BUFDRTSB	Test-and-set byte for the buffer header—this byte is set to X'FF' when a buffer is being taken from the buffer pool and assigned to a placeholder; set to X'00' in all other cases

Buffer Control Block (BUFC)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
11 (B)	1	BUFDRFLG	Buffer status flags:
	1...	BUFDRREL	Buffer-released flag, which is set when a placeholder returns a buffer to the buffer pool
	.1..	BUFDRAVL	Buffer is available, which is set when there are data buffers in the pool that are not reserved for inserts and are not assigned to placeholders
	..xx xxxx		Reserved
12(C)	4	BUFDBUFC	Address of the first BUFC
16(10)	4		Reserved

Buffer Control Entry

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	1	BUFCAVL BUFCUCNT	Test-and-set byte for the buffer Use count
1 (1)	1	BUFFLG1	BUFC status flags:
	1...	BUFCUPG	This BUFC is associated with an upgrade set
	.1..	BUFCSEG	The buffer contains a segment of a spanned record
	..1.	BUFCINS	Identifies this buffer as an insert buffer—this buffer can be assigned to a placeholder for data only for the duration of a single request
	...1	BUFCER1	Error generated by input processing
	.... 1...	BUFCER2	Error generated by output processing
	.... .1..	BUFCVAL	Input RBA is valid
	.... ..1.	BUFCEXC	The control interval represented by this BUFC is in exclusive control—this field is meaningful only when the input RBA is valid
	1	BUFCEPT	I/O-complete flag
2 (2)	1	BUFCIOFL	I/O status flags:
	1...	BUFCMW	Control interval must be written at the indicated output RBA (BUFCORBA)—note that output processing is done before input processing for the same BUFC
	.1..	BUFCFMT	This BUFC is associated with a format-write channel program
	..1.	BUFCRRD	The control interval indicated by BUFCDDDD must be read
	..1	BUFCREAL	BUFCBAD is a real address
	.... 1...	BUFCWC	The channel program associated with this BUFC includes write-validity-checking CCWs
	.... .1..	BUFCXEDB	The RBA that was to be read or written was in an extent of the data set that was unavailable (for example, not mounted)
	.... ..1.	BUFCPFCP	Preformatted channel-program segment is complete
	1	BUFCFIX	Buffer is fixed in real storage

Buffer Control Block (BUFC)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
3 (3)	1	BUFCFLG2	Flags:
	1...	BUFCXDDR	Suppress dynamic device reconfiguration on errors
	.1..	BUFCNLAS	Indicates last BUFC
	..1.	BUFCBSYR	For processing with shared resources, a read operation is in progress—the bit is on during the operation
	...1	BUFCBSYW	For processing with shared resources, a write operation is in progress—the bit is on during the operation
	 xxxx		Reserved
4 (4)	4	BUFCPLH	Address of the placeholder associated with this BUFC
		BUFCAMB	Address of the access method block associated with this BUFC
8 (8)	4	BUFCDDDD	RBA for input processing (valid only if bit in FLG1 is set)
12 (C)	4	BUFCORBA	RBA for output processing (valid only if IOFL indicates that a control interval must be written).
16 (10)	4	BUFCCPA	Channel program area address
20 (14)	4	BUFCBAD	Address to or from which control interval is to be written or read
24 (18)	4	BUFCNXT1	Next BUFC for which I/O can be requested
28 (1C)	4	BUFCINV	Invoker's field for OS/VS Auxiliary Storage Manager
28 (1C)	2	BUFCWLEN	BUFC data length
32 (20)	4	BUFCDSPC	Address of data-set page-control table
36 (24)	1	BUFCIDXL	For processing *without* shared resources, the level of the index record in the buffer—used in the selection of the buffer to be replaced
37 (25)	3	BUFCNXT2	Address of the next logical buffer
40 (28)	4	BUFXIRBA	RBA of the record in the buffer or, for a spanned record, of the record's first segment
44 (2C)	4	BUFXORBA	Same as BUFXIRBA, but used for output
48 (30)	4	BUFCHAIN	Address of the next BUFC in the pool
52 (34)	4	BUFCMDBT	For shared resources, modification bits—they identify IDs of transactions that have modified the buffer (RPL TRANSID operand)
56 (38)	4	BUFCUCUP	Address of the next BUFC up the use chain
60 (3C)	4	BUFCUCDN	Address of the next BUFC down the use chain

CLW—CLOSE Work Area

The CLW contains information used for communication among the CLOSE and temporary CLOSE modules. It is built by IDA0200T (CLOSE) and IDA0231T (CLOSE, TYPE=T), mapped by IDACLWRK, and pointed to by register 4 during VSAM CLOSE processing.

CLOSE Work Area (CLW)—Description and Format

Offset	Bytes and Bit Patterns	Field Name	Description
0 (0)	8	CLWID	Work area ID—IDACLWRK
8 (8)	4	CLWCOMWK	Address of common work area
12 (C)	4	CLWAMBPT	Address of current AMB
16 (10)	12	CLWSFI	Subfunction information area
28 (1C)	2	CLWFLAGS	Flag bytes:
	Byte 1:		
	1...	CLWBNOFL	No buffer flush
	.1..	CLWCNOUP	No catalog update
	..1.	CLWNWRIT	No write buffer
	...1	CLWPATH	Path processing
	.... 1...	CLWSPHCL	Close entire sphere
	.... .1..	CLWDUMMY	Dummy data set
	.... ..1.	CLWOUTPT	Base data set opened for output
	1	CLWPARCL	Partial close
	Byte 2:		
	1...	CLWPRMCL	Primary close
	.1..	CLWSECCL	Secondary close
	..1.	CLWGMAIN	Module work area built
	...1	CLWTERM	Terminating error in IDA0200B
	 xxxx		Reserved

CMB—Cluster Management Block

The CMB contains the addresses of header elements in the header element block that describe storage obtained for the control blocks of a key-sequenced or entry-sequenced data set.

The CMB is pointed to by the AMBL (AMBLCMB). It is further described in “Virtual-Storage Management” in “Diagnostic Aids.”

Cluster Management Block (CMB)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	1	CMBID	Control block identifier, X‘11’
1 (1)	1		Reserved
2 (2)	2	CMBLEN	Length of the CMB
4 (4)	1	CMBFLGS	Flags:
	1...	CMBOUT	The control block structure allows output requests
	.xxx xxxx		Reserved
5 (5)	1	CMBNST	Number of strings set up in the control block structure
6 (6)	2	CMBCNT	Number of addresses that follow:
8 (8)	40	CMBPTRS	Addresses of header elements in the header element block.
8 (8)	4	CMBUSRPT	User block header
12 (C)	4	CMBPRPTR	Protected user block header
16 (10)	4	CMBSTPTR	String block header
20 (14)	4	CMBUSPTR	Upgrade string block header
24 (18)	4	CMBFSTPT	Fixed string block header

Cluster Management Block (CMB)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
28 (1C)	4	CMBUFSPT	Fixed upgrade string block header
32 (20)	4	CMBBFRPT	Buffer block header
36 (24)	4	CMBUBFPT	Upgrade buffer block header
40 (28)	4	CMBDEBPT	DEB (data extent block) block header
44 (2C)	4	CMBEDBPT	EDB (extent definition block) block header
48 (30)	4	CMBPSTPT	Protected string block header
52 (34)	4	CMBPUSPT	Protected upgrade string block header
56 (38)	4	CMBFXDPT	Fixed block header
60 (3C)	4		Reserved

CPA—Channel Program Area

The CPA contains addresses to CCW chains that perform specialized I/O processing. The CPA also contains information needed to convert the addresses of virtual storage data areas to real main storage addresses for the channel. Each BUFC has a CPA associated with it, pointed to by the BUFCCPA.

Note: See I/O-Management module listings for channel program building and execution details. The formats of four channel programs follow this description of the CPA.

Channel Program Area (CPA)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	1	CPAID	Control block identifier, X'71'
1 (1)	1		Reserved
2 (2)	2	CPALEN	Length of the CPA
4 (4)	4	CPAWREAL	Real address of the previous write channel program segment
8 (8)	4	CPAWCPS	Real address of the first CCW in the write channel program segment
12 (C)	4	CPAWCPE	Real address of the last CCW in the write channel program segment
16 (10)	4	CPAWCKS	Real address of the first CCW in the write check channel program segment
20 (14)	4	CPAWCKE	Real address of the last CCW in the write check channel program segment
24 (18)	4	CPARREAL	Real address of the previous read program channel segment
28 (1C)	4	CPARCPS	Real address of the first CCW in the read channel program segment
32 (20)	4	CPARCPE	Real address of the last CCW in the read channel program segment
36 (24)	8	CPAWPHAD	The physical address for records to be written, in the form MBBCCHHR:
36 (24)	1		Reserved (M value)
37 (25)	6	CPAWSEEK	Seek address:
37 (25)	2	CPAWBB	BB value

Channel Program Area (CPA)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
39 (27)	4	CPAWCHR	Cylinder and head address (CCHH value)
43 (2B)	1	CPAWPHR	Reserved (record number R)
44 (2C)	4	CPAWSID	Address of the search argument list for write channel program segments
48 (30)	4	CPAFWCNT	Address of the count fields list for the format write channel program segment
52 (34)	8	CPARPHAD	The physical address for records to be read, in the form MBBCCHHR:
52 (34)	1		Reserved (M value)
53 (35)	6	CPARSEEK	Seek address:
53 (35)	2	CPARBB	BB value
55 (37)	4	CPARSID	Cylinder and head address (CCHH value) (Read search-ID argument)
59 (3B)	1		Reserved (record number R)
60 (3C)	4	CPAIDAL	Address of the real page list (indirect data-address list)
64 (40)	4	CPAVPL	Address of the virtual page list
68 (44)	4	CPAWORK1	Work area
72 (48)	4	CPAWORK2	Work area
76 (4C)	4	CPABLKSZ	The physical blocksize value calculated by the I/O Manager: Convert routine
80 (50)	2	CPABCINV	Number of physical blocks per control interval
82 (52)	1	CPASSECT	Set sector argument
83 (53)	1	CPASTAT1	Flags:
	1...	CPAVPLV	The virtual page list (VPL) is valid
	.xxx xxxx		Reserved
84 (54)	2	CPAFLAGS	Flags:
84 (54)	Byte 1	CPAFLAG1	
	1...	CPAWV	The write channel program segment is valid
	.1..	CPAWCV	The write check channel program segment is valid
	..1.	CPARV	The read channel program segment is valid
	...1	CPAWRPS	The write channel program segment (preceeded by a set sector CCW) is valid
	.... 1...	CPARRPS	The read channel program segment (preceeded by a set sector (CCW) is valid
	.... .1..	CPACHNED	Chaining of the channel program segments is complete
	xx		Reserved

Channel Program Area (CPA)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
85 (55)	Byte 2	CPAFLAG2	
	1...	CPAWREPL	The write channel program segment is used to write replicated index records
	.1..	CPARREPL	The read channel program segment is used to read replicated index records
	..1.	CPAXLRA	There has been a LRA instruction error
	...1	CPAPFENT	The pagefix appendage has been called
	.... 1...	CPATKOFL	Track overflow
	xxx		Reserved
86 (56)	1	CPARSECT	Set sector argument—read
87 (57)	1	CPAWSECT	Set sector argument—write
88 (58)	4	CPANXT1	Next CPA in chain from AMB
92 (5C)	4	CPACPCHN	Next CPA for a particular request (from IOMB)

Channel Programs

Four channel programs (read, format write, update write, and write check) are used for I/O operations.

Read Channel Program

The read channel program is used to retrieve data from direct-access storage.

Read Channel Program—Description and Format

CCW	Command Code			Flags		
Number	Hex	Description	Address	Hex	Description	Count
R1	1B	Seek head	CPARSEEK	40	CC	6
R2	23[1]	Set sector	CPARSECT	60	CC, SLI	1
R3	31	Search ID eq.	CPARSID	60	CC, SLI	5
R4	08	TIC	R3			
R5[2]	06[3]	Read data	IDAL	40	CC	CPABLKSZ
	86[4]	M-T read data	IDAL	40	CC	CPABLKSZ
R*n*	03[5]	No op		20	SLI	2

[1] Unless there is RPS (rotational position sensing), R2 is a no op.

[2] R5 is repeated for each physical record per control interval that is retrieved.

[3] R5 uses a read-data command for the first physical record.

[4] R5 uses a multiple-track read-data command for subsequent physical records.

[5] R*n* can be changed to a TIC (transfer in channel) command to chain to another read channel program.

Format Write Channel Program

The format write channel program is used to preformat or write data on a whole track (as in loading a data set with the SPEED option).

Format Write Channel Program—Description and Format

CCW	Command Code			Flags		
Number	Hex	Description	Address	Hex	Description	Count
FW1	1B	Seek head	CPAWSEEK	40	CC	6
FW2	23[1]	Set sector	CPAWSECT	60	CC, SLI	1
FW3	31	Search ID eq.	CPAWSID	40	CC	5
FW4	08	TIC	FW3			
FW5[2]	D	Write C,K,&D	CPAFWCNT	80	CC	8
FW6[2]	D	Write C,K,&D	IDAL	44	CC, IDAL	CPABLKSZ
FW*n*	03[3]	No op		20	SLI	2

[1] Unless there is RPS (rotational position sensing), FW2 is a no op.

[2] FW5 and FW6 are repeated (write count, key, and data) for each physical record on a track.

[3] FW*n* can be changed to a TIC (transfer in channel) command to chain to another format write channel program or to a write check channel program.

Update Write Channel Program

The update write channel program is used to write data on a part of a track (as in insertion).

Update Write Channel Program—Description and Format

CCW	Command Code			Flags		
Number	Hex	Description	Address	Hex	Description	Count
UW1	1B	Seek head	CPAWSEEK	40	CC	6
UW2[1]	23	Set sector	CPAWSECT	60	CC, SLI	1
UW3[2]	31	Search ID eq.	CPAWSID	40	CC	5
UW4[2]	08	TIC	UW3			
UW5[2]	05	Write data	IDAL	44	CC, IDAL	CPABLKSZ
UW*n*	03[3]	No op		20	SLI	2

[1] Unless there is RPS (rotational position sensing), UW2 is a no op.

[2] UW3, UW4, and UW5 are repeated for each physical record indicated in the CPA. The command code for subsequent UW3s is B1, multiple-track search ID equal.

[3] UW*n* can be changed to a TIC (transfer in channel) command to chain to another update write channel program or to a write check channel program.

Write Check Channel Program

The write check channel program is used to retrieve data to compare it with the data that was previously written.

Write Check Channel Program—Description and Format

CCW	Command Code			Flags		
Number	**Hex**	**Description**	**Address**	**Hex**	**Description**	**Count**
WC1	1B	Seek head	CPAWSEEK	40	CC	6
WC2	23[1]	Set sector	CPAWSECT	60	CC, SLI	1
WC3 5	31	Search ID eq.	CPAFWCTN[2]		40 CC	
			CPAWSID[3]			
WC4	08	TIC	WC3			
WC5	06[4]	Read data	IDAL	50	CC, Skip	CPABLKSZ
	86[5]	M-T read data	IDAL	50	CC, Skip	CPABLKSZ
WC*n*	03[6]	No op		20	SLI	2

[1] Unless there is RPS (rotational position sensing), WC2 is a no op.

[2] CPAFWCNT is used to check a format write.

[3] CPAWSID is used to check an update write.

[4] WC5 uses a read-data command for the first physical record.

[5] WC5 uses a multiple-track read-data command for subsequent physical records.

[6] WC*n* can be changed to a TIC (transfer in channel) command to chain to another write check channel program.

CSL—Core Save List

The CSL contains up to 32 entries that describe virtual-storage areas acquired by GETMAIN in Open. It enables Open to free these areas if it detects an error that prevents them from being freed in normal Open termination. The CSL is used by the Open error-cleanup routine as well as by the recovery routine.

The CSL is pointed to by the BIB. Additional CSLs are chained as required.

Core Save List (CSL)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	4	CSLR0	Used to load register 0 for FREEMAIN
0 (0)	1	CSLSUBPL	Subpool number of the CSL
1 (1)	3	CSLLENTH	Length of the CSL
4 (4)	8	CSLID	Identifier: 'ƀIDACSLƀ'
12 (C)	4	CSLNXPTR	Address of the next CSL (zero for the last CSL in the chain)
16 (10)	2	CSLACTEN	Number of active entries
18 (12)	2		Reserved
20 (14)	12 x 32	CSLNTRYS	Entries for virtual-storage areas:
CSL Entry			
0 (0)	12	CSLENTRY	An entry for a virtual-storage area:
0 (0)	8	CSLFREMN	Information for FREEMAIN
0 (0)	1	CSLPOOLN	Subpool number of the virtual-storage area
1 (1)	3	CSLCORLN	Length of the virtual-storage area
4 (4)	4	CSLCORPT	Address of the virtual-storage area
8 (8)	1	CSLFLAGS	Flags:
	1...	CSLKEY5	The storage is in key 5
	.1..	CSLKEY7	The storage is in key 7
	00..		The storage is in key 0 or the key of the problem program
	..1.	CSLJSTCB	The storage is owned by the job-step TCB
	...x xxxx		Reserved
9 (9)	3	CSLANCPT	Address of the CMB location that contains the address of the header element in the HEB for the virtual-storage area, or zero

DIWA—Data Insert Work Area

The DIWA is a work area used by the control area and control interval splitting modules. The DIWA is pointed to by the data AMB (AMBIWA).

Data Insert Work Area (DIWA)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	1	DIWID	Control block identifier, X'41'
1 (1)	1	DIWATV	Test-and-set (TS) assembler instruction is issued against this field to obtain exclusive use of the DIWA
2 (2)	2	DIWLEN	Length of a DIWA in bytes
4 (4)	1	DIWFLG1	Flag byte 1:

Data Insert Work Area (DIWA)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
	1...	DIWCAS	Control-area split is in progress
	.1..	DIWCISPL	Control-interval split has been performed
	.1.	DIWPFERR	I/O error occurred during preformating
	...1	DIWEOKR	Key of a record to be inserted in a key-range data set is greater than the highest possible key in the current key range—this end-of-key-range condition causes a control-interval split
	.... 1...	DIWGSPC	Spanned record needs a new control area
	.... .1..	DIWSHIFT	There is a shift in the insert point
	.... ..1.	DISNOT1	The buffer had intermediate or last segment of a spanned record
	1	DIW1ST	The buffer had first or intermediate segment of a spanned record
5 (5)	1	DIWFLG2	Flag byte 2:
	1...	DIWFSPF	Preformatting is needed in an entry-sequenced data set
	.xxx xxxx		Reserved
6 (6)	2		Reserved
8 (8)	4	DIWLRBA	Address of the first control interval in a control area that is being split
12 (C)	4	DIWHRBA	Address of the last control interval in a control area that is being split
16 (10)	4	DIWPLH	Address of the PLH which is currently associated with the DIWA
20 (14)	4	DIWBUFC	Address of the BUFC that controls the insert work buffer
24 (18)	4	DIWSPLTP	Address of the RDF associated with the first record to be moved to a new control interval as a result of a control-interval split
28 (1C)	20	DIWSAVE	Register save area:
28 (1C)	4	DIWSAVE1	Register 1
32 (20)	4	DIWSAVE2	Register 2
36 (24)	4	DIWSAVE3	Register 3
40 (28)	4	DIWSAVE4	Register 4
44 (2C)	4	DIWSAVE5	Register 5

DSL—DEB Save List

The DSL contains up to 16 entries that describe DEBs that have been successfully chained and added to the DEB table. It enables Open to free the DEBs if an error prevents them from being freed normally. The DSL is used only by the Open error-cleanup routine, not by the recovery routine.

The DSL is pointed to by OPWA (called the ACB work area). Additional DSLs are chained as required.

DEB Save List (DSL)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	1	DSLSUBPL	Subpool number of the DSL
1 (1)	3	DSLLENTH	Length of the DSL
4(4)	8	DSLID	Identifier: 'ƀIDADSLƀ'
12 (C)	4	DSLNXPTR	Address of the next DSL (zero for the last DSL in the chain)
16 (10)	2	DSLACTEN	Number of active entries
18 (12)	2		Reserved
20 (14)	4 x 16	DSLENTRY	Entries for DEBs:
20 (14)	1	DSLFLG	Flags:
	1	DSLFDDEB	The DEB is a dummy DEB
	xxxx xxx.		Reserved
21 (15)	3	DSLDEBAD	Address of the DEB

EDB—Extent Definition Block

The EDB describes all extents of the space allocated to the cluster's data set. The EDB is built by the VSAM Open routine from information in the data set's catalog record.

The EDB header contains the length of the EDB and the number of EDB entries that follow the header. Each EDB entry describes an extent, and contains the address of the associated LPMB. The EDB header is pointed to by the AMB (AMBEDB).

Extent Definition Block (EDB)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
EDB Header			
0 (0)	1	EDBID	Control block identifier, X'90'
1 (1)	1	EDBNO	Number of EDB entries—one EDB per extent
2 (2)	2	EDBLEN	Length of an EDB entry in bytes
4 (4)	4	EDBLPMBC	Address of first LPMB
EDB Entry			
0 (0)	2		Reserved
2 (2)	1	EDBFLG1	Flags
	1...	EDBLKR	For a catalog, low-key range
	.1..	EDBTOFLW	For page-space data set, track overflow used
	..1.	EDBPSDS	Page-space data set
	...x xxxx		Reserved
3 (3)	1	EDBM	Extent number—specifies the relative location of an extent entry in a DEB

Extent Definition Block (EDB)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
4 (4)	4	EDBLPMBA	Address of LPMB
8 (8)	4	EDBSTTRK	Relative track address of the extent associated with this EDB
12 (C)	4	EDBLORBA	RBA of the start of the extent
16 (10)	4	EDBHIRBA	RBA of the end of the extent
20 (14)	4	EDBTKBAL	For track-overflow processing, the number of bytes left on a track after the last I/O operation

ESL—Enqueue Save List

The ESL contains up to 16 entries that describe ENQ requests that have been issued by Open, Close, or End of Volume for data set sharing. It enables Open to dequeue the indicated resources if an error prevents them from being dequeued normally. The ESL is used only by the Open error-cleanup routine, not by the recovery routine.

The ESL is pointed to by OPWA (called the ACB work area). Additional ESLs are chained as required

Enqueue Save List (ESL)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	1	ESLSUBPL	Subpool number of the ESL
1 (1)	3	ESLLENTH	Length of the ESL
4 (4)	8	ESLID	Identifier: 'ƀIDAESLƀ'
12 (C)	4	ESLNXPTR	Address of the next ESL (zero for the last ESL in the chain)
16 (10)	2	ESLACTEN	Number of active entires
18 (12)	2		Reserved
20 (14)	9 x 16	ESLENTRY	Entries for resources enqueued:
20 (14)	1	ESLENQOP	The ENQ option that was used for this resource: 0 Exclusive use 1 Shared use
21 (15)	8	ESLRNAME	ENQ resource name (minor) that identifies this resource:
21 (15)	3	ESLCINBR	Control-interval number for the resource
24 (18)	4	ESLACBAD	Address of the ACB of the catalog for the resource
28 (1C)	1	ESLIO	Indicator of the purpose of the ENQ: I input O output

EXLST—Exit List

The EXLST contains the addresses of exit routines supplied by the user. It is created by the user with the EXLST or GENCB macro. The EXLST is pointed to by the ACB (ACBEXLST).

Exit List (EXLST)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	1	EXLID	Control block identifier, X'81'
1 (1)	1	EXLSTYP	Subtype identifier: X'10' = VSAM X'20' = VTAM
2 (2)	2	EXLLEN	Length of the control block
4 (4)	1		Reserved
5 (5)	1	EXLEODF	Entry description
6 (6)	4	EXLEODP	Address of the EODAD exit routine
10 (A)	1	EXLSYNF	Entry description
11 (B)	4	EXLSYNP	Address of the SYNAD exit routine
15 (F)	1	EXLLERF	Entry description
16 (10)	4	EXLLERP	Address of the LERAD exit routine
20 (14)	10		Reserved
30 (1E)	1	EXLJRNF	Entry description
31 (1F)	4	EXLJRNP	Address of the JRNAD exit routine
35 (23)	10		Reserved

HEB—Header Element Block

The HEB is used by VSAM Virtual-Storage Management to allocate and free unprotected storage blocks. It contains 16 header elements, each of which describes a storage block. It is further described in "Virtual-Storage Management" in "Diagnostic Aids."

The HEB is pointed to by the BIB (BIBHEBPT). The first free header element is pointed to by BIBHEBFQ.

Header Element Block (HEB)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
HEB Block Definition			
0 (0)	1	HEBID	Control block identifier, X'13'
1 (1)	1		Reserved
2 (2)	2	HEBLEN	Length of the HEB (including header elements)
4 (4)	4	HEBNHEB	Address of the next HEB (or 0)
8 (8)	2		Reserved
10 (A)	2	HEBCNT	Number of header elements
12 (C)	20 x 16	HEBHDELS	Header elements:
HEB Header Element Definition			
0 (0)	8	HEBFREMN	Information for freeing the storage block described by this header element:
0 (0)	1	HEBSP	Subpool in which the storage block is located

Header Element Block (HEB)—Description and Format

	Bytes and		
Offset	Bit Pattern	Field Name	Description
1 (1)	3	HEBLN	Length of the storage block
4 (4)	4	HEBBLKPT	Address of the storage block
8 (8)	1	HEBFLAGS	Flags:
	1...	HEBJSTCB	The storage is owned by the job step TCB
	.1..	HEBKEY5	The storage is in key 5
	..1.	HEBKEY7	The storage is in key 7
	.00.		Storage is obtained in key 0 or in problem-program key
	...1	HEBIOSUP	O/C/EOV special request block
	.... 1...	HEBRTFLG	Recovery termination freed storage
	xxx		Reserved
9 (9)	3	HEBAVSP	Amount of space available in the storage block
12 (C)	4	HEBELCHN	Address of the next header element
16 (10)	4	HEBNBYTE	Address of the next available byte

ICWA—Index Create Work Area

The ICWA contains information needed when a VSAM index record is being built or modified during key-sequenced data set creation. The sequence-set ICWA is pointed to by the index AMB (AMBIWA). ICWAs are built by Open; there is one for each level of the index.

Index Create Work Area (ICWA)—Description and Format

	Bytes and		
Offset	Bit Pattern	Field Name	Description
0 (0)	1	ICWID	Control block identifier, X'43'
1 (1)	1	ICWFLG1	Flag byte:
	1...	ICWWNF	Entry won't fit in the index record
	.1..	ICWWAGM	The Open routine did not supply a workarea
	..1.	ICWRBAOK	Don't get RBA on initial
	...1	ICWVSE	The section entry is valid
	.... 1...	ICWVNE	The previous entry is valid
	.... .1..	ICWKRDS	The data set is divided into key ranges
	.... ..1.	ICWSPLIT	The work area contains a split index record
	1	ICWENDRQ	The Close routine requires a control interval split
2 (2)	2	ICWLEN	Length of the ICWA
4 (4)	4	ICWCHN	Address of the next ICWA
8 (8)	4	ICWBUFC	Address of the current index BUFC
12 (C)	4	ICWCRBA	Current index RBA
16 (10)	4	ICWPRBA	Previous index RBA
20 (14)	2	ICWPSEO	Displacement from the beginning of the index record to the prior section entry
22 (16)	2	ICWSCNT	Number of entries in the current section
24 (18)	4	ICWADD	Address of the current work area
28 (1C)	4	ICWTBASE	Base RBA
32 (20)	4	ICWTPTR	Address of the index save position
36 (24)	4	ICWARDBP	Address of the current ARDB
40 (28)	2	ICWLN	Index level number

Index Create Work Area (ICWA)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
42 (2A)	2	ICWKEY1L	Length of the current key
44 (2C)	2	ICWKEY2L	Length of the previous key
46 (2E)	2	ICWKEY3L	Length of the section key
48 (30)	2	ICWNEST	Number of entries in the index section
50 (32)	2	ICWNOSEG	Number of segments in a spanned record
52 (34)	2	ICWCRSEG	Number of the segment being processed
54 (36)	1	ICWREQ	Request type
55 (37)	1	ICWPTL	Index entry pointer length
56 (38)	1	ICWCER	Rear compression count of the current index entry
57 (39)	1	ICWCEF	Current index entry F—number of front-key compressed bytes
58 (3A)	1	ICWCEL	Current index entry L—length of the compressed key in the entry
59 (38)	1	ICWCERP	Rear compression count of the previous index entry
60 (3C)	(key length)	ICWKEY1	Save area for the current key
VL	(key length)	ICWKEY2	Save area for the previous key
VL	(key length)	ICWKEY3	Save area for the section key

IICB—ISAM Interface Control Block

The IICB is used to address the DCB (ISAM) and the ACB and RPL (VSAM) control blocks and associated areas needed by the ISAM interface. The IICB is pointed to by the DEBWKPT5 field in the ISAM DEB to provide integrity and by the RPLIICB field in the RPL Extension to provide the connection to VSAM control programs.

ISAM Interface Control Block (IICB)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	1	IICBID	Control block identifier, X'80'
1 (1)	1		Reserved
2 (2)	2	IICBLEN	Length of IICB, in bytes
4 (4)	4	IIDCBPTR	Address of DCB
8 (8)	4	IIACBPTR	Address of ACB
12 (C)	4	IIRPLPTR	Address of RPL
16(10)	4	IIW1CBF	Address of dummy scan work area
16 (10)	2	IISAVLRL	Length of current record
18 (12)	2	IIMAXLRL	Maximum record length
20 (14)	4	IIKEYPT	Address of key (dummy ISAM) save area
24 (18)	1	IIFLAG1	ISAM interface status flags:
	1...	IIFSCAN	Scan mode
	.1..	IIFGET	First GET request
	..1.	IIFPASS	First pass in load mode
	...1	IIFCLOSE	Close in process
	.... 1...	IIDATA	Data only retrieval

ISAM Interface Control Block (IICB)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
	.... .1..	IIFTEST	Loop test bit
	.... ..1.	IISEQCHK	Resume load sequence check
	1	IIQBFRS	QISAM does not use buffers—no FREEMAIN is required
25 (19)	3	IIACBL	ACB, EXLST, IICB length for GETMAIN/FREEMAIN
28 (1C)	1	IIFLAG2	ISAM interface status flags used by Open to designate the fields being merged by ISAM Interface. ISAM Interface Close uses the same mask to restore the DCB to its pre-open status.
	1...	MRKP	Relative key position
	.1..	MLRECL	Logical record length
	..1.	MBLKSI	Block size
	...1	MOPTCD	Option code
	.... 1...	MRECFM	Record format
	.... .1..	MBUFL	Buffer length
	.... ..1.	MBUFNO	Buffer number
	1	MKEYLE	Key length
29 (1D)	3	IIRPLL	RPL and RPLE: length for GETMAIN/FREEMAIN
32 (20)	2	IIKEYSL	Length of key save area, in bytes
34 (22)	2	IIBUFL	Length of single ISAM Interface buffer (used in calculations)
36 (24)	1	IIFLAG3	ISAM interface status flags:
	1...	MBFALN	BFALN merge bit
	.xxx xxxx		Reserved
37 (25)	3	IIMSGL	Message area length
40 (28)	4	IIMSGPTR	Message area pointer
44 (2C)	1	IIBUFNO	Number of ISAM Interface buffers built by Open
45 (2D)	3	IITBUFL	Total BCB and buffer length for GETMAIN/FREEMAIN
48 (30)	4	IISVCLST	SVC exit for SYNADAF
52 (34)	8	IISAMSYN	ISAM SYNAD name—used when SYNAD is specified in the AMP parameter
60 (3C)	72	IIREGSAV	Register save area
60 (3C)	4		Reserved
64 (40)	4	IIREGBC	Previous save area pointer
68 (44)	4	IIREGFC	Next save area pointer
72 (48)	60		Remainder of save area
132 (84)	36	IIAUD	Audit information
132 (84)	4	IIAUDHDR	
132 (84)	1	IIAUDFL1	Audit flags
	1...	AUDACBOP	OPEN was issued for ACB
	.1..	AUDACBRO	Control was returned from Open
	..1.	AUDDCBEX	A DCB exit was taken
	...1	AUDDCBRT	Control was returned from the DCB exit
	.... xx..	AUDPRMOD	A processing module was loaded: '01'IDAIIPM1

ISAM Interface Control Block (IICB)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
			'10'IDAIIPM2
			'11'IDAIIPM3
	.... ..1.	AUDIISYN	ISAM-Interface SYNAD routine was loaded
	1	AUDURSYN	User SYNAD routine was loaded
133 (85)	1	IIAUDFL2	Audit flags
	1...	AUDIIFBF	IDAIIFBF was loaded
	.1..	AUDACBCL	CLOSE was issued for ACB
	..1.	AUDACBRC	Control was returned from Close
	...1	AUDBFREX	A flush-buffer exit was taken to IDAIIPM1
	.... 1...	AUDBRFRT	Control was returned from IDAIIPM1
	.... .1..	AUDDEBXF	The DEB extension was freed
	xx		Reserved
134 (86)	2	IIGMCNTR	Offset from IIAUD to the next available entry in the audit-information fields
136 (88)	32	IIGMAUD	Address of virtual-storage areas gotten
136 (88)	4	AUDIICB	Address of this IICB
140 (8C)	4	AUDCSPLI	Subpool number and length
140 (8C)	1	AUDCSPI	Subpool number
141 (8D)	3	AUDCLI	Length
144 (90)	4	AUDCDEB	Address of the DEB
148 (94)	4	AUDCSPLD	Subpool number and length
148 (94)	1	AUDCSPD	Subpool number
149 (95)	3	AUDCLD	Length
152 (98)	4	AUDCBFRS	Address of the area for buffers and RPLs
156 (9C)	4	AUDCSPLB	Subpool number and length
156 (9C)	1	AUDCSPB	Subpool number
157 (9D)	3	AUDCLB	Length
160 (A0)	4	AUDCMSGA	Address of the physical-error message area
164 (A4)	4	AUDCSPLM	Subpool number and length
164 (A4)	1	AUDCSPM	Subpool number
165 (A5)	3	AUDCLM	Length

IMWA—Index Insert Work Area

The IMWA is a control block used in inserting an index entry into the index of a key-sequenced data set. The IMWA is created by the Open routine, and is pointed to by the ICWA (ICWCHN).

Index Modification Work Area (IMWA)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	1	IMWID	Control Block identifier, X'42'
1 (1)	1	IMWFLAGS	Control flags:
	1...	IMWNEWHL	Indicates a new high level should be built in the index structure
	.1..	IMWRIPL	Indicates a new entry must be built in an index record at the next higher level to reflect a new index record created by an index split

Index Modification Work Area (IMWA)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
	..1.	IMWBSE	Indicates the new index entry should be a section entry
	.x xxxx		Reserved
2 (2)	2	IMWLEN	Length of IMWA in bytes
4 (4)	4	IMWIXSP	Address of index search parameter list
8 (8)	32	IMWISWKA	Index search parameter list (see IXSPL)
40 (28)	4	IMWXKEYP	Address of the next (higher-keyed) index entry
44 (2C)	4	IMWIKEYP	Address of the new index entry's key
48 (30)	4	IMWXPTR	Value of the index pointer field in the next (higher-keyed) index entry
52 (34)	4	IMWIPTR	Value to be inserted in new index entry's pointer field
56 (38)	4	IMWLBUFC	Address of a data BUFC for a data buffer containing the lowest key following a control-area split
60 (3C)	4	IMWBUFP	Address of the index record being processed
64 (40)	1	IMWFGAIN	Front key-compression adjustment to be added to IBFLPF field in next (higher-keyed) index entry
65 (41)	1	IMWIEL	Value of IBFLPL field—that is, compressed length of new index entry's key
66 (42)	1	IMWSVIEL	Save area for IBFLPL value
67 (43)	1		Reserved
68 (44)	2	IMWCIMVN	Readjustment to number of control intervals in old control area following a control area split to enable an index record to be built for the new control area
70 (46)	2	IMWNSOFF	Offset to next section entry in index record
72 (48)	4		Reserved
76 (4C)	(key length)	IMWKEY1	Highest possible key for a mass insertion—that is, last key in a sequence of keys to be inserted which is less than an existing key; also, save area for current insert key under a no-fit condition

IOMB—I/O-Management Block

The IOMB is used by I/O Management to control its processing of a request. It contains the addresses of other control blocks, flags used by I/O Management, and a 16-word register save area. The addresses of the first BUFC and CPA are inserted by I/O Management after it verifies the control blocks.

The IOMB is pointed to by the PLH (PLHIOB). It points to the IOSB, which points to the SRB. These three control blocks take the place of the IOB, which is used by some other drivers of the I/O Supervisor.

I/O Management Block (IOMB)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	4	IOMBID	Control block identifier: 'IOMB'
4 (4)	4	IOMBUFC	Address of the first BUFC
8 (8)	4	IOMCPA	Address of the first CPA
12 (C)	4	IOMPLH	Address of the PLH
16 (10)	4	IOMAMB	Address of the AMB
20 (14)	4	IOMIQE	Address of the IQE (interrupt queue element)
24 (18)	4	IOMECBPT	Address of the ECB
28 1C)	4	IOMVSL	Address of the VSL (virtual subarea list, which is the same as the PFL, page fix list)
32 (20)	4	IOMPGAD	Address of the caller to get control when I/O operation is complete (zero for Record Management—used for Auxiliary Storage Management)
36 (24)	4	IOMIOSB	Address of the IOSB
40 (28)	3	IOMFLAGS	Flags:
40 (28)	2	IOMFL	Flags reset after I/O completes:
			Byte 1:
	xx..	IOMAPEND	Appendage flags:
	1...	IOMNE	Normal End Appendage completed
	.1..	IOMAE	Abnormal End Appendage completed
	..1.	IOMPURGE	A purge is in progress
	.... 1...	IOMCBERR	A control block wasn't valid
	.... .1..	IOMADERR	Virtual addresses in the VPL weren't successfully converted to real addresses for the IDAL
	.... ..1.	IOMPGFIX	Pages are fixed in real storage
	1	IOMCSW	The address of the channel status word is incorrect
	...x		Reserved
			Byte 2:
	1...	IOMDDR	Dynamic-device reconfiguration
	.xxx xxxx		Reserved
42 (2A)	1	IOMSTIND	Status indicators:
	1...	IOMAMUSE	The IOMB is in use
	.1..	IOMEOVW	End of Volume is waiting for an IOMB
	..1.	IOMEOVTS	End of Volume has set the IOMLOCK field
	...1	IOMEOVXC	End-of-Volume indicator
	.... 1...	IOMLLOCK	A local lock is held
	.... .1..	IOMSLOC	SALLOC is held
	.... ..1.	IOMSRBM	The user is processing with SRB (event) dispatching
	x		Reserved

I/O Management Block (IOMB)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
43 (2B)	1	IOMCKEY	Key of the caller of I/O Management
44 (2C)	1	IOMPFERR	Return code from the VS2 PGFIX routine
45 (2D)	1	IOMLOCK	End-of-Volume lock
46 (2E)	2	IOMNMOD	Number of modules to be fixed in real storage
48 (30)	2	IOMNBUF	Number of buffers
50 (32)	2	IOMNSEG	Number of channel program segments
52 (34)	64	IOMSAVER	16-word register save area and work area:
52 (34)	64	IOMSAVE0 through IOMSAVEF	16 four-byte registers save areas
116 (74)	4	IOMNXT1	Address of the next IOMB
120 (78)	4	IOMUFLD	Address of the IOMB extension (IOMBXN)

IOMBXN—I/O Management Block Extension

See the AMBXN.

IOSB—I/O-Supervisor Block

The IOSB is used by the VS2 I/O Supervisor to initiate and terminate an I/O operation. It is passed to the I/O Supervisor by VSAM I/O Management (IDA121A2—the Actual Block Processor), along with an SRB.

The IOSB is used to communicate between the I/O Supervisor and the requester of I/O, between the I/O Supervisor and an error-recovery procedure, between an error-recovery procedure and write-to-operator and statistics-update modules, and among the components of the I/O Supervisor. It is also used to control successive entries from the I/O Supervisor to an error-recovery procedure.

The format of the IOSB is given in *OS/VS2 Data Areas.*

IXSPL—Index Search Parameter List

The IXSPL is used to pass index search parameters to the index search routine. It also contains status information about the results of the search. It is used as a work area by the SCIB (Search Compressed Index Block) routine (IDA019RC). The PLH contains the address of the IXSPL (PLHISPLP) or the contents of the IXSPL (PLHIXSPL).

Index Search Parameter List (IXSPL)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	4	IXSSTRBA	RBA of the index record to search first.
4 (4)	4	IXSBUFC	Address of the index BUFC
8 (8)	4	IXSARG	Address of the search argument (a key field)
12 (C)	1	IXSTLN	Index level number at which the search is to terminate
13 (D)	1	IXSILN	Index level number at which the search is to begin

Index Search Parameter List (IXSPL)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
14 (E)	3		Reserved
17 (11)	1	IXSBFLG	Flags:
		IXSSSRH	Used by the Search Compressed Index Search routine:
	1...		Search for a section entry only
	0...		Search for a normal entry
	.1..	IXSLELV	The entry located by the Index Search routine is the last entry in the terminating level (F = 0 and L = 0)
	..xx xxxx		Reserved
18 (12)	1	IXSEKON	Length of the F, L, and pointer fields in each index entry
19 (13)	1	IXSPEC	The number of characters in the index entry preceding the entry located by the Index Search routine that equalled the search argument
20 (14)	4	IXSHEP	Address of the index entry located by the Index Search routine
24 (18)	4	IXSSEP	Address of the section entry that is greater than or equal to the index entry located by the Index Search routine
28 (1C)	4	IXSLEP	Address of the lowest-valued entry in the section identified by IXSSEP

KEYWDTAB—Keyword Processing Table

KEYWDTAB is a branch tabe that controls the execution of IDA019C1 and supports processing for the GENCB, MODCB, SHOWCB, and TESTCB macros. The table is built by and contained within IDA019C1 and is not referred to by any other module. The table contains one 14-byte row for each keyword processed by a control block macro, and each row is identified by a keyword type code (0-255). Each column in the table represents functions for the keywords and contains index points for specific keyword functions. Each column also contains either offsets and lengths for byte-oriented fields or pointers to descriptive information about bit-oriented fields. The index points are used to route specific requests through IDA019C1 on the bases of keyword, block (ACB, EXLST, NIB (VTAM), and RPL), and function (GENCB, MODCB, SHOWCB, and TESTCB).

Offset	Bytes	Description
0 (0)	14	The description for the keyword with type code = 0 (KW00)
0 (0)	3	The index points for the ACB
0 (0)	1	The index point for MODCB of the ACB
1 (1)	1	The index point for SHOWCB of the ACB
2 (2)	1	The index point for TESTCB of the ACB
3 (3)	3	The index points for the EXLST
3 (3)	1	The index point for MODCB of the EXLST
4 (4)	1	The index point for SHOWCB of the EXLST
5 (5)	1	The index point for TESTCB of the EXLST
6 (6)	3	The index points for the RPL
6 (6)	1	The index point for MODCB of the RPL
7 (7)	1	The index point for SHOWCB of the RPL
8 (8)	1	The index point for TESTCB of the RPL
9 (9)	3	The index points for the NIB (VTAM)
10 (A)	1	The index point for SHOWCB of the NIB
11 (B)	1	The index point for TESTCB of the NIB
12 (C)	2	The offset to a bit definition if this is a bit-level keyword
13 (D)	1	The offset of the resultant field in the target field, if this is a byte field
14 (E)	14	The description for the keyword with type code = 1 (KW01)
.		
.		
.		
28 (1C)	14	The description for the keyword with type code = 2 (KW02)
.		
.		
.		
3570 (DF2)	14	The description for the keyword with type code = 255 (KW255), the maximum value

LPMB—Logical-to-Physical Mapping Block

The LPMB contains information about the direct-access device that contains the user's data set. The LPMB is built by the VSAM Open routines, which use information in the data set's catalog record. Each EDB entry (EDBLPMBA) contains the address of an LPMB.

Logical-to-Physical Mapping Block (LPMB)—Description and Format

	Bytes and		
Offset	**Bit Pattern**	**Field Name**	**Description**
0 (0)	1	LPMBID	Control-block identifier, X'91'
1 (1)	1	LPMBFLGS	Flags:
	1...	LPMBRPS	The device has the rotational position sensing (RPS) feature
	.1..	LPMREPL	Records are replicated on the track
	..1.	LPMSS	Sequence set records are stored with the data records
	...1	LPMBTOFL	Track overflow
	.... 1...	LPMBSSTH	The set sector table is included at the end of the LPMB
	xxx		Reserved

Logical-to-Physical Mapping Block (LPMB)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
2 (2)	2	LPMBLEN	Length of the LPMB
4 (4)	4	LPMAUSZ	The minimum number of bytes that can be allocated to an object.Allocation is always an integer multiple of LPMAUSZ. For a data set, this field is the control interval size. For an index, this field is the device's track size.
8 (8)	4	LPMBPTRK	Number of bytes per track
12 (C)	4	LPMBLKSZ	Number of bytes per physical record (control interval)
16 (10)	2	LPMTRKAU	Number of tracks per allocation unit (extent)
18 (12)	2	LPMTPC	Number of tracks per cylinder
20 (14)	2	LPMBLKTR	Number of physical records (control intervals) per track
22 (16)	2		Reserved
24 (18)	4	LPMBEXT	Reserved for address of LPMB extension
28 (1C)	VL	LPMBSST	Set sector table—it is built by Open for use in deriving the sector number from the record number

OPW—OPEN Work Area

OPW is the common work area used by VSAM OPEN routines. It is built by IDA0192A, mapped by IDAOPWRK, and pointed to by register 4 during VSAM processing.

OPEN Work Area (OPW)— Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	1	OPWSUBPL	Subpool of work area
1 (1)	3	OPWLENTH	Work area length
4 (4)	8	OPWID	Block ID—IDAOPWRK
12 (C)	1	OPWFLGS1	Flag byte 1:
	1...	OPWCAT	Catalog open
	.1..	OPWSCRA	System CRA open
	..1.	OPWVVIC	MSVI data set
	...x xxxx		Reserved
13 (D)	1	OPWFLGS2	Flag byte 2:
	1...	OPWUCRA	User CRA open
	.1..	OPWIXDT	Index open as an ESDS
	..1.	OPWAIXDT	Alternate index open for end use
	...1	OPWDUMMY	Open dummy data set
	 xxxx		Reserved
14 (E)	1	OPWFLGS3	Flags for IDA0192F
	1...	OPWDAVAT	Dummy AMBL added to VAT
	.1..	OPWPUPGR	Path also in upgrade set
	..1.	OPWUPGOP	Upgrade set open
	...1	OPWNOWRK	MOD work area does not exist
	.... 1...	OPWRSTRT	Restart in progress
	xxx		Reserved

OPEN Work Area (OPW)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
15 (F)	1	OPWFLGS4	Authorization flags:
	1...	OPWFULL	Full access
	.1..	OPWCINV	Control-interval access
	..1.	OPWUPD	Update access
	...x xxxx		Reserved
16 (10)	4	OPWBIB	Address of the BIB
20 (14)	4	OPWCOMWA	Address of Open common work area
24 (18)	8	OPWIDF	Cluster identifier
24 (18)	4	OPWCACB	Address of catalog ACB
28 (1C)	3	OPWDCI	Control interval number of data component
31 (1F)	1	OPWQ	Open qualifier:
	1...	OPWDDC	Connect by DD name
	.1..	OPWGSR	Opened for GSR
	..1.	OPWLSR	Opened for LSR
	...1	OPWFSTP	Opened for ICI
	.... 1...	OPWUBF	Opened for user buffering
	.... .1..	OPWKSDS	Opened as a KSDS
	.... ..1.	OPWESDS	Opened as an ESDS
	1	OPWDFR	Opened with deferred write option
32 (20)	16	OPWVSMPL	O/C/EOV Virtual Storage Manager parameter list
32 (20)	4	OPWVMANC	Address of anchor block
36 (24)	1	OPWVMSP	Subpool for direct request
37 (25)	3	OPWVMLNG	Amount of storage requested
40 (28)	4	OPWVMADR	Address of storage acquired (zero, if storage not obtained)
44 (2C)	1	OPWVMTYP	Request type
45 (2D)	1	OPWVMFLG	Flag byte:
	1...	OPWVMPGB	Get storage on a page boundary
	.1..	OPWVMKE5	Get storage in Key 5
	..1.	OPWVMKE7	Get storage in Key 7 (if not key 5 or key 7, get storage in key 0 or problem program key)
	...1	OPWVMSRB	Special request block
	.... 1...	OPWVMNSL	Do not build a CSL for this request
	.... .1..	OPWVMTCB	Storage is owned by JOBSTEP TCB
	xx		Reserved
46 (2E)	2		Reserved
48 (30)	76	OPWVSMWA	O/C/EOV Virtual Storage Manager work area
48 (30)	4	OPWVANCP	Pointer to the address of the first HEB header element associated with this request
52 (34)	4	OPWVTBLP	Address of the request table used by GETSPACE routine
56 (38)	4	OPWVCSLP	Used to scan for a CSL entry
60 (3C)	4	OPWVCSLE	Address of save list entry
64 (40)	4	OPWVHDRE	Address of header element
68 (44)	4	OPWVR13	Address of caller's save area
72 (48)	16	OPWVSAVE	IDA0192M save area

OPEN Work Area (OPW)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
88 (58)	36		The following 12-byte field is repeated three times:
0 (0)	12	OPWVGSPL	GET SPACE parameter list
0 (0)	1	OPWVGSSP	Subpool number
1 (1)	3	OPWVGETL	Length of acquired storage
4 (4)	4	OPWVGSPT	Address of acquired storage
8 (8)	1	OPWVGFLG	Flags for GET SPACE (see OPWVMFLG, above, for description of bit settings)
9 (9)	3	OPWVREQL	Length of request
124 (7C)	4	OPWVANCS	Address of BIB anchor for sphere block requests
128 (80)	8	OPWVLSAV	SETLOCK save area
128 (80)	4	OPWVRG12	Save area for register 12
132 (84)	4	OPWVRG13	Save area for register 13
136 (88)	8	OPWVFMPL	FREEMAIN parameter list
136 (88)	1	OPWVFMSP	FREEMAIN subpool number
137 (89)	3	OPWVFMLN	FREEMAIN length
140 (8C)	4	OPWVFMPT	FREEMAIN address
144 (90)	20	OPWSAVE	Addresses of save lists
144 (90)	4	OPWCSL	Address of core save list
148 (94)	4	OPWESL	Address of ENQ save list
152 (98)	4	OPWPSL	Address of page-fix save list
156 (9C)	4	OPWDSL	Address of DEB save list
160 (A0)	4	OPWSSL	Address of swap save list
164 (A4)	4	OPWCURPT	Address of cluster being processed. This field can point to OPWBSECL (528(210)), OPWPTAIX (536(218)), OPWUPAIX (548(224)), and every 8 bytes thereafter, since OPWUPAIX is a repeating field. The format of current cluster information is described below by OPWCURCL.
168 (A8)	4	OPWXAMBL	Address of current AMBL
172 (AC)	4	OPWCAMBL	The address of the existing AMBL for connecting to an existing structure
176 (B0)	4	OPWBCON	Address of base AMBL connecting to
180 (B4)	4	OPWPCON	Address of path AMBL connecting to
184 (B8)	4	OPWBAMBL	Address of AMBL for base
188 (BC)	4	OPWPAMBL	Address of AMBL for path
192 (C0)	6	OPWCRA	CRA volume serial number
198 (C6)	1		Reserved
199 (C7)	1	OPWCATTR	MVS cluster attributes:
	xxxx xxx.		Reserved
	1		Page space data set
200 (C8)	4	OPWUPT	Address of upgrade table
204 (CC)	4	OPWUACB	Address of user ACB

OPEN Work Area (OPW)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
208 (D0)	4	OPWWRKPT	Address of current AMB work area
212 (D4)	4	OPWDTWRK	Address of data AMB work area
216 (D8)	4	OPWIXWRK	Address of index AMB work area
220 (DC)	4	OPWCTCB	Address of current TCB
224 (E0)	4	OPWJSTCB	Address of JOBSTEP TCB
228 (E4)	4	OPWTIOT	Address of TIOT entry
232 (E8)	4	OPWBUFND	Number of data buffers
236 (EC)	4	OPWBUFNI	Number of index buffers
240 (F0)	1	OPWCSTRN	Current string number
241 (F1)	1	OPWSTRNO	Path string number, if path processing; otherwise, base string number
242 (F2)	1	OPWBSTRN	Base string number, if base processing
243 (F3)	1		Reserved
244 (F4)	52	OPWDACB	Dummy ACB for opening base
296 (128)	12	OPWSFI	Subfunction information
308 (134)	256	OPWERMAP	Map of return codes to ACBERFLG, where return code *rc* is defined in *OS/VS Message Library: VS2 System Messages* for messages IEC070I, IEC161I, IEC251I, and IEC252I.
564 (234)	4	OPWSAVEA	Return address save area
568 (238)	8	OPWBSECL	Base cluster information
568 (238)	1		Reserved
569 (239)	3	OPWBDTCI	Base data control interval number
572 (23C)	1		Reserved
573 (23D)	3	OPWBIXCI	Base index control interval number
576 (240)	8	OPWPTAIX	Path alternate index information
576 (240)	1		Reserved
577 (241)	3	OPWPDTCI	Path alternate index data control interval number
580 (244)	1		Reserved
581 (245)	3	OPWPIXCI	Path alternate index control interval number
584 (248)	1	OPWNOUPG	Number of upgrade alternate indexes
585 (249)	3	OPW2YPLH	Address of PLHNXT for IDA0192Y and IDA0192Z
588 (24C)			The following 8-byte field, pointed to by OPWCURPT (164(A4)), is repeated once for each upgrade alternate index associated with the base cluster being processed.
0 (0)	8	OPWUPAIX	Upgrade alternate index information
0 (0)	1		Reserved
1 (1)	3	OPWUDTCI	Upgrade alternate index data control interval number
4 (4)	1		Reserved
5 (5)	3	OPWUIXCI	Upgrade alternate-index index control interval number

OPEN Work Area (OPW)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
The format of information about the cluster being processed (pointed to by OPWCURPT 128 (X'80')) is shown below:			
0 (0)	8	OPWCURCL	Current cluster information
0 (0)	1	OPWCFLG1	Cluster flags (set by sphere Open):
	1...	OPWBASE	Open base cluster
	.1..	OPWPATH	Open path alternate index
	..1.	OPWUPGR	Open upgrade alternate index
	...1	OPWSVWRK	Do not free AMB work areas
	.... 1...	OPWPRTBL	Partial control-block build
	xxx		Reserved
1 (1)	3	OPWCDCTI	Data component control interval number
4 (4)	1	OPWFLG2	Cluster flags (set by cluster Open)
	1...	OPWDOPEN	Open indicator on in catalog for data
	.1..	OPWMODWK	Module work area exists
	..1.	OPWEMPUP	Empty upgrade data set
	...1	OPWERR2B	Terminating error in IDA0192B
	.... 1...	OPWIOPEN	Open indicator on in catalog for index
	xxx		Reserved
5 (5)	3	OPWCIXCI	Index component control interval number

PLH—Placeholder

The PLH contains current information about a string of requests. This information includes positioning information, request options, and buffer location and status. The PLH is built by the Open routine and is pointed to by the AMB (AMBPH). The next PLH in the chain is pointed to by PLHCHAIN. The number of PLH entries is the number given in STRNO in the ACB. (When a data set is shared, the number is the number in the first ACB opened for the data set.) When a Record-Management routine is processing a PLH, the PLH's address is in register 2 (RPLH).

Placeholder (PLH)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
PLH Header			
0 (0)	1	PLHID	Control block identifier, X'30'
1 (1)	1	PLHCNT	Number of PLH entries that follow the header
2 (2)	2	PLHELTH	Length of each PLH entry
4(4)	4	PLHDRREQ	Count of requests that have been deferred
8(8)	2	PLHDRMAX	Maximum number of placeholders (PLH entries) in concurrent use
10(A)	2	PLHDRCUR	Number of active placeholders
12(C)	4	PLHIOSDQ	Data-Set Management (I/O Support) deferral queue header
PLH Entry			
0 (0)	1	PLHAVL	Zero if the PLH entry is available
1 (1)	1	PLHATV	Zero if there are no active requests

Offset	Bytes and Bit Pattern	Field Name	Description
2 (2)	1	PLHFLG1	Process flags, byte 1:
	1...	PLHEOVW	The VSAM End of Volume routine is waiting
	.1..	PLHENDRQ	The caller issued an ENDREQ request
	..1.	PLHASKBF	Less than maximum buffers
	...1	PLHSSR	The sequence set is stored with the data
	.... 1...	PLHRDEXC	Read exclusive mode
	.... .1..	PLHASYRQ	IRB execution needed
	.... ..1.	PLHDRPND	A deferred request is pending
	x		Reserved
3 (3)	1	PLHPFLG2	Process flags, byte 2:
	1...	PLHUPD	The previous request was a GET-for-update
	.1..	PLHSQINS	Sequential insertion mode
	..1.	PLHKEYMD	Keyed mode
	...1	PLHADDTE	Add to the end processing
	.... 1...	PLHKRE	End of key range indicator
	.... .1..	PLHCIINS	Control interval split insertion
	.... ..1.	PLHSVADV	Save the PLHNOADV field during Scan Data
	1	PLHIWAIT	IDAWAIT indicator that an asynchronous test is in progress
4 (4)	1	PLHEFLGS	Exception flags:
			Byte 1:
	1...	PLHNOSPC	No space found—create mode
	.1..	PLH1ST	This is the first request after the data set was opened
	..1.	PLHSKPER	Skip across the error control interval
	...1	PLHSRINV	Spanned record is invalid
	.... 1...	PLHNOADV	Don't advance the PLH
	.... .1..	PLHEODX	The EODAD exit was taken
	.... ..1.	PLHINVAL	Something is invalid
	1	PLHDSCAN	Scan data after read exclusive
			Byte 2:
	1...	PLHRSTRT	Restart
	.xxx xxxx		Reserved
6(6)	1	PLHFLG3	Flags:
	1...	PLHSRBSG	Update numbers in RDFs of spanned-record segments aren't the same
	.1..	PLHRAHD	Do read-ahead buffering
	..1.	PLHSLVLD	Second level of the index is valid
	...1	PLHBWD	Previous request specified backward processing
	.... 1...	PLHRVRS	The I/O chain is reversed
	xxx		Reserved
7(7)	1	PLHAFLGS	Flags:
	1...	PLHDRLM	A direct request was issued during loading of an empty data set
	..1.	PLHVAMB	The AMB that points to the PLH is valid
	...1	PLHDBDC	The PLH is from the VSAM resource pool
	.... 1...	PLHIOSID	I/O-Support ID
	.... .1..	PLHRABWD	IDA019RA was entered for backward processing
	.x.. ..xx		Reserved
			The next two fields comprise the DSID (Data-Set Identification):
8 (8)	4	PLHACB	Address of the caller's ACB

Placeholder (PLH)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
12 (C)	1	PLHDSTYP	Data set type:
	X'01'		Data
	X'02'		Index
13 (D)	1	PLHRMIN	Read threshold
14 (E)	1	PLHFRCNT	Number of free buffers
15 (F)	1	PLHBFRNO	Total number of buffers
16 (10)	4	PLHMRPL	Address of the RPL header
20 (14)	4	PLHCRPL	Address of the current RPL
24(18)	4	PLHDSIDA	Address of the DSID (PLHACB field above)
28 (1C)	4	PLHCRBA PLHJORBA	Current RBA Old RBA—to support the JRNAD exit routine
32 (20)	4	PLHJRNLL	Length of the data—to support the JRNAD exit routine
36 (24)	4	PLHJNRBA	New RBA—to support the JRNAD exit routine
40 (28)	1	PLHJCODE	Entry code—to support the JRNAD exit routine
	0000 0000	PLHJGET	JRNAD entry for GET
	.... .1..	PLHJPUT	JRNAD entry for PUT
	.... 1...	PLHJERS	JRNAD entry for ERASE
	.... 11..	PLHJRBAC	JRNAD entry for RBA change
41 (29)	1	PLHRCODE	Indicates the previous request type
42 (2A)	1	PLHEOVR	End-of-Volume request code—indicates space allocation or volume mount
43 (2B)	1		Reserved
44 (2C)	4	PLHARDB	Address of the current data ARDB
48 (30)	4	PLHLRECL	Length of the record processed during the previous request
52 (34)	4	PLHDBUFC	Address of the current data BUFC
56 (38)	4	PLHNBUFC	Address of the next read BUFC
60 (3C)	4	PLHRECP	Address of the current record
64 (40)	4	PLHFSP	Address of the first byte of free space within the record
68 (44)	4	PLHRDFP	Address of the current RDF
72 (48)	2	PLHRDFC	Replication count for the current RDF
74 (4A)	2	PLHSRSID	Spanned-record segment ID
76 (4C)	4	PLHDIOB	Address of the data IOMB
		PLHIIOB	Address of the index IOMB
80 (50)	4	PLHARET	Return address to the I/O Manager's Asynchronous Routine
84 (54)	24	PLHSAVE1 through PLHSAVE6	Six 4-byte register save areas—not to be used by Buffer Management, I/O Management, IDADRQ, or IDATJXIT
108 (6C)	4	PLHAMB	AMB-address save area for IDADRQ and IDATJXIT
112 (70)	4	PLHCHAIN	Address of the next PLH in the chain

Placeholder (PLH)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
116 (74)	2	PLHRET0	Offset to the current register 14 save area in the push-down list (PLHRET1)
118 (76)	2		Reserved
120 (78)	44	PLHRET1	Save area (push-down list) for 11 return registers (register 14)
164 (A4)	4	PLHASAVE	Beginning of save area for I/O Management's Asynchronous Routine
168 (A8)	4		Save area for thirteenth return register
172 (AC)	4		Save area for fourteenth return register
176 (B0)	4	PLHAR14	Address to which the Asynchronous Routine is to return
180 (B4)	4	PLHEOVPT	Address of the RBA provided by the End-of-Volume routine
		PLHDDDD	RBA of the previous request
184 (B8)	4	PLHNRBA	Next RBA
188 (BC)	4	PLHIBUFC	Address of the index BUFC
192 (C0)	4	PLHRBUFC	RBUFC save area for IDADRQ and IDATJXIT
196 (C4)	4	PLHISPLP	Address of the IXSPL
200 (C8)	32	PLHIXSPL	Space for one IXSPL
200 (C8)	4	PLHSSRBA	RBA of the sequence-set control interval
		PLHHIREC	RBA of the high record
204 (CC)	4	PLHIXBFC	Address of a BUFC for index
208 (D0)	24		Parameter area for index search
232 (E8)	4	PLHWAX	Address of the work area for path processing
		PLHXPLH	Address of the PLH for the alternate index of the base cluster
236 (EC)	4	PLHLLOR	Address of the least length of the data record that contains all of the record's key fields
240 (F0)	2	PLHNOSEG	Number of segments in a spanned record
242 (F2)	2	PLHSRCSG	Number of the segment being processed
244 (F4)	4	PLHSLRBA	RBA of the second level of the index
248 (F8)	4	PLHKEYPT	Address of the current key (PLHKEY at end of PLH entry)
		PLHRRN	Previous relative record number
252 (FC)	4	PLHDRRSC	Address of the deferred-request flag byte
256 (100)	4	PLHPARM1	RPARM1 save area for IDADRQ and IDATJXIT
260 (104)	4	PLHR13	Register 13 save area for IDADRQ and I/O Management
264 (108)	1	PLHDRMSK	Mask to test for resources for a deferred request
265 (109)	3		Reserved
268 (10C)	4	PLHECB	Address of event control block for cross-region post
272 (110)	4	PLHASCB	Address of address space control block for cross-region post

Placeholder (PLH)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
276 (114)	4	PLHERRET	Address to which to return from an error (for cross-region post)
280 (118)	0	PLHEND	Label for the end of the PLH entry before PLHEXTEN and PLHKEY
280 (118)	28	PLHEXTEN	Extension to the PLH for processing with shared resources (optional):
280 (118)	4	PLHRESR1	Address of a serial resource being held
284 (11C)	1		Reserved
285 (11D)	1	PLHBMWRK	Buffer-Management work flags:
	1...	PLHBMRDF	The RBA was found in the buffer pool (for SCHBFR macro)
	.1..	PLHBEUC	End of use chain
	..1.	PLHBMSOV	Start-over flag
	...x xxxx		Reserved
286 (11E)	2	PLHRDCNT	Save area for AMBRDCNT
288 (120)	20	PLHBMSV1 through PLHBMSV5	Five 4-byte save areas for Buffer Management
VL	VL	PLHKEY	The current key, pointed to by PLHKEYPT

PSL—Page Save List

The PSL contains a variable number of entries that describe the pages of virtual storage that have been fixed in real storage by Open. It enables Open to free these pages if an error prevents them from being freed normally.

The PSL is pointed to by OPWA (called the ACB work area). There is no more than one PSL per data set.

Page Save List (PSL)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	1	PSLSUBPL	Subpool number of the PSL
1 (1)	3	PSLLENTH	Length of the PSL
4 (4)	8	PSLID	Identifier: 'ƀIDAPSLƀ'
12 (C)	4	PSLNXPTR	Zero
16 (10)	8 x *n*	PSLENTRY	Entries for pages fixed:
16 (10)	4	PSLSTAD	Address of the beginning of the virtual-storage area that was fixed
20 (14)	1	PSLFLG	Flags:
	1...	PSLFLGLT	This is the last entry
	.xxx xxxx		Reserved
21 (15)	3	PSLENDAD	Address of the first byte beyond the virtual-storage area that was fixed

RPL—Request Parameter List

The RPL contains user-request information and error feedback information. It also contains information required by GET and PUT macros.

The RPL is created by the user with the RPL or the GENCB macro.

Request Parameter List (RPL)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	4	RPLIDWD	Identification word of the RPL:
0 (0)	1	RPLID	Control block identifier, X‘00’
1 (1)	1	RPLSTYP	RPL subtype: X‘10’ = VSAM X‘20’ = VTAM
2 (2)	1	RPLREQ	Request type—when the user issues a VSAM macro, register 0 contains one of the following request-type codes; when VSAM processes the request, the request-type code in register 0 is transferred to the RPLREQ field (unless the request is CHECK or ENDREQ) 0(0) GET request 1(1) PUT request 2(2) CHECK request 3(3) POINT request 4(4) ENDREQ request 5(5) ERASE request 6(6) VERIFY request 8(8) Data preformat request 9(9) Index preformat request 10(A) Force I/O request 11(B) GETIX request 12(C) PUTIX request 13(D) SCHBFR request 14(E) MRKBFR request 15(F) WRTBFR request
3 (3)	1	RPLLEN	Length of the RPL
4 (4)	4	RPLPLHPT	Address of the PLH
8 (8)	1	RPLECB	Address of the external ECB, or an internal ECB:
	1...	RPLWAIT	The event has not yet completed
	.1..	RPLPOST	The event has completed
	..xx xxxx		Reserved
9 (9)	3		Reserved, if RPLECB is an internal ECB, or the address of the external ECB
12 (C)	4	RPLFDBWD	Feedback work:
12 (C)	1	RPLSTAT	RPL status flags:
	.1..	RPLCHKI	CHECK has been issued
	..1.	RPLEDRQI	ENDREQ has been issued
	x..x xxxx		Reserved
13 (D)	3	RPLFDBK	RPL feedback area (See “Diagnostic Aids” for a list of RPL return codes and condition codes.)
13 (D)	1	RPLRTNCD RPLERREG	RPL return code
	X‘00’		Normal return
	X‘04’		Invalid control block

Request Parameter List (RPL)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
	X'08'		Logical error
	X'0C'		Physical error
14 (E)	2	RPLCNDCD	RPL condition code
14 (E)	1	RPLCMPON	Component issuing the code
15 (F)	1	RPLERRCD	Error code
16 (10)	2	RPLKEYLE RPLKEYL	Key length
18 (12)	2	RPLSTRID	RPL string identifier
20 (14)	4	RPLCCHAR	Address of the control character
24 (18)	4	RPLDACB	Address of the caller's ACB
28 (1C)	4	RPLTCBPT	Address of the user's TCB—this field is always zero for a VSAM RPL
32 (20)	4	RPLAREA	Address of the caller's record area
36 (24)	4	RPLARG	Address of the caller's search argument
40 (28)	4	RPLOPTCD	Option flags
40 (28)	1	RPLOPT1	Option flag byte 1:
	1...	RPLLOC	Locate mode
	0...		Move mode
	.1..	RPLDIR	Direct-search access
	..1.	RPLSEQ	Sequential access
	...1	RPLSKP	Skip sequential processing
	.... 1...	RPLASY	Asynchronous request
	.... 0...		Synchronous request
	.... .1..	RPLKGE	Search key greater than or equal
	.... .0..		Search key equal
	.... ..1.	RPLGEN	Generic key
	.... ..0.		Full key
	1	RPLECBSW	The RPLECB field contains the ECB's address
41 (29)	1	RPLOPT2	Option flag byte 2:
	1...	RPLKEY	Locate the record identified by a key
	.1..	RPLADR RPLADD	Locate the record at the caller-specified relative byte address (RBA)
	..1.	RPLCNV	Locate the control interval at the caller-specified RBA
	...1	RPLBWD	Process in backward direction
	.... 1...	RPLLRD	Locate or retrieve the last record in the data set
	.... ..1.	RPLUPD	Update processing
	1	RPLNSP	Note the string position
	.... .x..		Reserved

Request Parameter List (RPL)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
42 (2A)	1	RPLOPT3	Option flag byte 3:
	1...	RPLEODS	End of the user's output data set
	.1..	RPLSFORM	Spool form on remote
	..1.	RPLBLK	Block the records
	..0.		The records are unblocked
	...1	RPLVFY	UCS/FCB verify
	.... 1...	RPLFLD	UCS fold
	.... .xx.	RPLFMT	Format type:
	.... .00.		UCS load
	.... .01.		FCB load
	.... .10.		Reserved
	.... .11.		Reserved
	1	RPLALIGN	Align the buffer and notify the operator
	0		Do not align the FCB buffer loads
43 (2B)	1	RPLOPT4	Reserved
44 (2C)	4	RPLNXTRP RPLCHAIN	Address of the next RPL in the chain
48 (30)	4	RPLRLEN	Length of the record
52 (34)	4	RPLBUFL	Length of the user's buffer
56 (38)	4		Reserved
60 (3C)	8	RPLRBAR	RBA return location
60 (3C)	2	RPLAIXPC	Alternate-index pointer count
62 (3E)	1	RPLAIXID	Alternate-index pointer type:
	x... ...	RPLAXPKP	Pointer is: 0 Prime-key pointer 1 RBA pointer
	.xxx xxxx		Reserved
63 (3F)	1		Reserved
64 (40)	4	RPLDDDD	Relative byte address
68 (44)	1		Reserved
69 (45)	1	RPLACTIV	CHECK not issued
70 (46)	2	RPLEMLEN	Error message length
72 (48)	4	RPLERMSA	Address of the error message area

RPLE—Request Parameter List Extension

An RPLE is built and appended to each ISAM Interface RPL when the user's ISAM program opens a VSAM cluster. The RPLE contains the address of the IICB, a register save area, a linkage to other RPLs in the ISAM Interface RPL pool, and a pointer to the ISAM DECB.

Request Parameter List Extension (RPLE)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	4	RPLIICB	Address of the IICB
4 (4)	4	RPLDECB	Address of the DECB—if the field contains zeros, the RPL has not been assigned to a DECB (BISAM only)

8 (8)	4	RPLIIBFR	Address of the ISAM Interface buffer associated with the RPL (the buffer is required for locate mode processing, data only retrieval, dynamic buffering, and BISAM stand-alone write)
12 (C)	4	RPLRPLPT	Address of the next RPL in the ISAM Interface RPL pool—if the RPL is the last RPL in the pool, this field contains zeros
16 (10)	1	RPLIITSB	Test-and-set (TS) byte—this field is used to indicate the assignment of the RPL to a BISAM DECB
17 (11)	3		Reserved
20 (14)	4	RPLSAVE	Register save area
24 (18)	4	RPLSAVE2	Register save area

SRB—Service Request Block

The SRB is used by the VS2 I/O Supervisor to dispatch I/O processing for a request. It identifies the address space in which processing is to be done.

The format of the SRB is given in *OS/VS2 Data Areas.*

SSL—Swap Save List

The SSL contains up to 16 entries that identify control blocks that are to be chained after Open has otherwise completed successfully. Deferring chaining makes it unnecessary to unchain the control blocks should Open fail.

Open uses the Compare-and-Swap instruction to chain or alter storage that is subject to simultaneous alteration by two or more tasks.

The SSL is pointed to by OPWA (called the ACB work area). Additional SSLs are chained as required.

Swap Save List (SSL)—Description and Format

Offset	**Bytes and Bit Pattern**	**Field Name**	**Description**
0 (0)	1	SSLSUBPL	Subpool number of the SSL
1 (1)	3	SSLLENTH	Length of the SSL
4 (4)	8	SSLID	Identifier: 'ƀIDASSLƀ'
12 (C)	4	SSLNXPTR	Address of the next SSL (zero for the last SSL in the chain)
16 (10)	2	SSLACEN	Number of active entries
18 (12)	2		Reserved
20 (14)	8 x 16	SSLENTRY	Entries for control blocks to be chained:
20 (14)	4	SSLSWPTR	Address of the word in which SSLSWAP is to be placed
24 (18)	4	SSLSWAP	The value that is to be placed at the address given is SSLSWPTR

UPT—Upgrade Table

The UPT describes the upgrade set of a base cluster. It contains an entry for each alternate index in the upgrade set. It is pointed to by the BIB (BIBUPT).

Upgrade Table (UPT)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
UPT Header			
0 (0)	4	UPTHDR	Header:
0 (0)	1	UPTID	Control block identifier, X'45'
1 (1)	1	UPTFLG0	Flags:
	1...	UPTPWS	Continue with scan
	.xxx xxxx		Reserved
2 (2)	2	UPTLEN	Length of the UPT
4 (4)	4	UPTNEW	Address of the new alternate-index record
8 (8)	4	UPTOLD	Address of the old alternate-index record
12 (C)	1	UPTRSC	Resource byte—used to serialize updates
13 (D)	1	UPTNOENT	Number of alternate indexes in the upgrade set (and of entries in the UPT)
14 (E)	2	UPTLLEN	Largest sum of key length plus the key's relative position in a data record
16 (10)	72	UPTSA	Save area:
16 (10)	4	UPTWORK1	Work area
20 (14)	4	UPTLSA	Last save area
24 (18)	1	UPTBEREG	RPLERREG value for the base cluster
25 (19)	1	UPTBERCD	RPLERRCD value for the base cluster
26 (1A)	2		Reserved
28 (1C)	4	UPTR14	Address to which IDA019R4 returns after I/O is issued for upgrading
32 (20)	56	UPTR15	Rest of save area
UPT Entry			
0 (0)	12	UPTAXENT	Entry for an alternate index in the upgrade set:
0 (0)	4	UPTRPL	Address of the upgrade RPL
0 (0)	1	UPTF1LOP	Last operation against the upgrade ACB
4 (4)	2	UPTFLG1	Flags:
			Byte 1:
	1...	UPTF1LST	This is the last entry in the UPT
	.1..	UPTF1ATV	This entry is active for an upgrade operation
	..1.	UPTF1NUK	The alternate index can have nonunique keys
	...1	UPTF1NOP	The alternate index is not open
	.... 1...	UPTF1NRF	A no-record-found error has occurred
	.... .x..	UPTF1KEY	The key being processed is: 0 Old 1 New
	.... ..1.	UPTF1RTY	The last operation is being retried
	1	UPTF1UPG	The alternate index is being upgraded
			Byte 2:
	1...	UPTF1BKO	An upgrade operation is being undone (backed out)
	.1..	UPTF1LOG	A logical error has occurred
	..1.	UPTF1PHY	A physical error has occurred

Upgrade Table (UPT)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
	...1	UPTF1ERA	The operation requiring upgrade was deletion (ERASE)
	.... 1...	UPTF1PNU	The operation requiring upgrade was insertion
	.... .1..	UPTF1PUD	The operation requiring upgrade was update
	xx		Reserved
6 (6)	2	UPTRKP	Relative alternate-key position in a base record
8 (8)	1	UPTPASS	The number of this upgrade operation (pass through the upgrade set)
9 (9)	1	UPTLNCDE	Length of key, minus 1
10 (A)	2	UPTBG	Length of RPLAREA field

VAT—Valid-AMBL Table

The VAT is used to check the validity of each AMBL that is built for processing a base key-sequenced cluster. It contains the address of each AMBL. The first VAT is pointed to by the JSCB (JSCBSHR).

Valid-AMBL Table (VAT)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	4	VATHDR	Header:
0 (0)	1	VATID	Control block identifier, X'11'
1 (1)	1		Reserved
2 (2)	2	VATLEN	Length of the VAT
4 (4)	4	VATNEXT	Address of the next VAT
8 (8)	8	VATVSRT	Used to update the use count and address of the VSAM shared resource table at the same time (with the CDS instruction)
8 (8)	4	VATVUSE	Use count in the VSRT
12 (C)	4	VATVPTR	Address of the VSRT
16 (10)	4	VATPAMBL	Address of the first AMBL in the primary chain
20 (14)	2	VATVC	Used for checking validity of AMBLs
20 (14)	1	VATVRT	The number of this VAT on the chain
21 (15)	1	VATENO	Number of entries in this VAT
22 (16)	2		Reserved
24 (18)	4	VATNAE	Number of active entries in this VAT
28 (1C)	4 x 16	VATAMBL	Addresses of valid AMBLs

VCRT—VSAM Checkpoint/Restart Table

The VCRT is used by VSAM checkpoint/restart. The VCRT, which is mapped by IDAVCRT, is suballocated from VCRCORE in IDA0606C. It contains a count, by entry type, of each entry associated with the VCRT. There are three entry types: open, upgrade, and index. The VCRT is pointed to by the BIB (BIBVCRT).

VSAM Checkpoint/Restart Table (VCRT)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	1	VCRID	Control block ID—X'80'
1 (1)	1	VCRFLAG1	Flag bytes:
	1...	VCRUPGSW	Entry type: 1 = Upgrade; 0 = Open
	.1..	VCRLSR	Local shared resources specified
	..1.	VCROUT	Output ACB is open
	...x xxxx		Reserved
2 (2)	2		Reserved
4 (4)	8	VCRIDNM	VCRT control block name—IDAVCRT
12 (C)	4	VCRCOREH	Address of first VCRCORE header
16 (10)	2	VCROPNCT	Open entry count
18 (12)	2	VCRUPGCT	Upgrade entry count
20 (14)	2	VCRIDXCT	Index entry count
22 (16)	2		Reserved
24 (18)	4	VCRCISIZ	Size of the largest control interval in the sphere
28 (1C)	4	VCRSPHPT	Address of sphere block HEB save area
32 (20)	4	VCRRBUF	Address of the repositioning buffer, or zeros
36 (24)	4	VCROPN	Address of first VCRT open entry
40 (28)	4	VCRUPG	Address of first VCRT upgrade entry
44 (2C)	4	VCRIDX	Address of first VCRT index entry

The VCRT open entry is used by VSAM restart to rebuild the control blocks needed for a valid restart. The format is:

0 (0)	4	VCRHEBS	Address of the HEB save area (zeros if the cluster is part of the upgrade set)
4 (4)	4	VCRAMBL	Address of the user's AMBL

The VCRT upgrade entry points to the upgrade AMBL and the HEB save areas to be processed by VSAM restart. The entry exists only if the upgrade set for this data set was open at checkpoint time. The format is:

0 (0)	4	VCRUHEBS	Address of the HEB save area
4 (4)	4	VCRUAMBL	Address of the upgrade AMBL

The VCRT index entry contains ICWA and buffer addresses for the index level it represents. The entry exists only if the base data set is a key-sequenced data set open for create mode processing. There is one entry for each index level that exists at open time. The format is:

0 (0)	4	VCRICWA	Address of the ICWA
4 (4)	4	VCRBUFPT	Address of the associated buffer

VCRCORE is created by VSAM checkpoint (IDA0C06C) and freed by the VSAM *checkpoint/restart cleanup routine in IDA0196C. The first VCR core header is pointed to by VCRCOREH (12(C)) in the VCRT. The format is:*

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)		VCRCHDR	VCR core header
0 (0)	8	VCRCNM	VCRCORE ID—VCRCORE
8 (8)	4	VCRCNEXT	Address of next VCR core header
12 (C)	4	VCRCDESC	Cleanup information
	1	VCRCSP	Subpool number containing this block
	3	VCRCLEN	Length of this block
16 (10)	4	VCRCPTRA	Address of first available byte in this block
20 (14)	4	VCRCLENA	Length of available storage in this block
24 (18)	VL	VCRCDATA	Storage for data (minimum 4072 bytes)

The HEB save area is pointed to by the open (VCRHEBS) or upgrade (VCRUHEBS) entries of the VCRT. The format of the save area is:

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	8	VCRHHDR	Header for each CMB entry (CMBPTRS) or for BIBSPHPT
0 (0)	2	VCRHNENT	Number of entries in header element chain
2 (2)	1	VCRHFLG	Flag byte:
	1...	VCRHFCON	This is a continuation of a previous CMB entry
	.1..	VCRHFREL	Issue FREEMAIN at restart time
	..xx xxxx		Reserved
3 (3)	1	VCRHCID	Relative CMB entry number, or 0 for BIBSPHPT
4 (4)	4	VCRHNEXT	Address of next HEB save area header for cluster
8 (8)			The following fields are repeated once for each entry in the header element chain
0 (0)	20	VCRHENT	Header element saved at checkpoint as defined by IDAHEB mapping macro
0 (0)	8	VCRHEFMN	FREEMAIN information:
0 (0)	1	VCRHESP	Subpool number
1 (1)	3	VCRHELN	Length of storage
4 (4)	4	VCRHESPT	Address of storage
8 (8)	12		Remaining content of header element

VGTT—VSAM Global Termination Table

The VGTT identifies global virtual storage that may need to be specially freed if an error prevents it from being freed normally. There are four types of VGTTs for:

- Keeping track of an address space's use of a global VSAM resource pool (for processing with global shared resources)
- Keeping track of certain control blocks (sets of IOSB, SRB, and PFL) that are kept in global storage for processing with local shared resources (the normal pointers to these control blocks are in unprotected local storage—the VGTT is in global storage, adjacent to them)
- Keeping track of control blocks during the opening of a catalog, a catalog recovery area, or the mass storage volume inventory data set (all of whose control blocks are kept in global storage
- Keeping track of certain control blocks (sets of IOSB, SRB, and PFL) that are kept in global storage for processing a user data set and all related data sets (such as alternate indexes)

The VGTT is pointed to by the ASCB (address space control block). It is chained to the next VGTT for the address space.

VSAM Global Termination Table (VGTT)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	4	VGTTID	Control block identifier: 'VGTT'
4 (4)	1	VGTTTYPE	VGTT type indicator:
	1..	VGTTRSTR	VGTT is for restart
	.1..	VGTTGSR	VGTT is for processing with global shared resources—VGTTVUSE is used
	..1.	VGTTLSR	VGTT is for processing with local shared resources
	...1	VGTTCTLG	VGTT is for opening a catalog, a catalog recovery area, or the mass storage volume inventory data set
	.... 1...	VGTTOPEN	VGTT is for processing a user's data set without shared resources
	xxx		Reserved
5 (5)	1		Reserved
6 (6)	1		Reserved
7 (7)	1		Subpool number of the VGTT and of the global storage it protects
8 (8)	4	VGTTSIZE	Length of the VGTT
12 (C)	4	VGTTNEXT	Address of the next VGTT (zero for the last VGTT in the chain)
16 (10)	4	VGTTBIB	Address of the base information block for the user's data set and all related data sets (such as alternate indexes)
20 (14)	4	VGTTVUSE	For a VGTT for global shared resources, the use count that was contributed by the processing of the user's data set and all related data sets.
24 (18)	4	VGTTPSB	Address of the protected sphere block (which contains HEBs for use by Virtual-Storage Management)

VSAM Global Termination Table (VGTT)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
28 (1C)	4		Reserved
32 (20)	VL	VGTTCORE	For a VGTT for local shared resources, the virtual-storage area the VGTT protects

VIOT—Valid-IOMB Table

The VIOT contains the address of each valid IOMB within a VSAM resource pool (for processing with shared resources). It is pointed to by the VSRT (VSRTVIOT) and by each AMB associated with the resource pool.

Valid-IOMB Table (VIOT)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
0(0)	4	VIOHDR	Header
0(0)	1	VIOID	Control block identifier, X'16'
1(1)	1		Reserved
2(2)	2	VIOLEN	Length of the valid-IOMB table
4(4)	4 x *n*	VIOPTR	Address of a valid IOMB; this field is repeated *n* times

VMT—Volume Mount Table

The VMT identifies and describes volumes that are mounted for a base cluster and all clusters associated with it for processing. There is a VMT for each device type. The first VMT is pointed to by the BIB (BIBVMT).

Volume Mount Table (VMT)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	4	VMTHDR	Header:
0 (0)	1	VMTID	Control block identifier, X'12'
1 (1)	1		Reserved
2 (2)	2	VMTLEN	Length of the VMT
4 (4)	4	VMTNXT	Address of the next VMT
8 (8)	2	VMTNOVOL	Number of volume entries (*n*) in the VMT
10 (A)	3		Reserved
13 (D)	3	VMTDEV	Device information:
13 (D)	1	VMTDVOPT	Device options
14 (E)	2	VMTDVTYP	Device class and type
16 (10)	16 x *n*	VMTVOL	Volume entry for a volume to be mounted:
16 (10)	4	VMTUSECT	Use count
20 (14)	1	VMTVFLG1	Volume flags:
	1...	VMTOPEN	The volume is being processed by Open
	.xxx xxxx		Reserved
21 (15)	1		Reserved
22 (16)	6	VMTVLSER	The volume's serial number
28 (1C)	4	VMTUCB	Address of the UCB for the volume

VSRT—VSAM Shared Resource Table

The VSRT contains the addresses of buffer pools and PLH pools in the resource pool and addresses of various control blocks built during the processing of a BLDVRP macro. For local shared resources (LSR), the VSRT is pointed to by the VAT (VATVPTR); for global shared resources (GSR), it is pointed to by the AMCBS (CBSVPTR), which is described in *OS/VS2 Catalog Management Logic.*

VSAM Shared Resource Table (VSRT)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	1	VSRTBKID	Control block identifier, X'15'
1 (1)	1		Reserved
2 (2)	2	VSRTLEN	Length of the VSRT
4 (4)	8	VSRTID	Visual identifier, 'IDAVSRTƀ'
12 (C)	2	VSRTFLGS	Flags:
			Byte 1:
	1...	VSRTGSRF	Global resource pool
	.1..	VSRTLSRF	Local resource pool
	..1.	VSRTIOBF	I/O-related control blocks are fixed in real storage
	...1	VSRTBFRF	Buffers are fixed in real storage
	 xxxx		Reserved
			Byte 2:
	xxxx xxxx		Reserved
14 (E)	1	VSRTKL	The maximum key length of the data sets that are sharing the resource pool
15 (F)	1	VSRTSTRN	The total number of placeholders required for all the data sets (specified in BLDVRP)
16 (10)	4	VSRTPLHH	Address of the PLH header
20 (14)	4	VSRTBUFH	Address of the BUFC header
24 (18)	4	VSRTCPAH	Address of the CPA header
28 (1C)	4	VSRTWAH	Address of the working storage header (WSHD)
32 (20)	4	VSRTVIOT	Address of the valid-IOMB table
36 (24)	8 x *n*	VSRTCSL	Entries for gotten storage:
36 (24)	1	VSRTCSLF	Flags:
	1...	VSRTCSFX	The storage is fixed in real storage
	.1..	VSRTCSVS	The storage contains the VSRT
	..1.	VSRTCSBF	The storage contains a buffer
	...1	VSRTCSPF	The storage contains the page fix list
	.... 1...	VSRTCSWS	The storage is for a work area (working storage)
	.... .1..	VSRTCSPL	The storage contains PLHs
	.... ..1.	VSRTCSIO	The storage contains IOMBs
	1	VSRTCSBH	The storage contains a buffer
37 (25)	3	VSRTCSAD	Address of the storage
40 (28)	1	VSRTCSSP	The number of the subpool the storage is located in
41 (29)	3	VSRTCSLN	Length of the storage

WAX—Work Area for Path Processing

The WAX contains addresses and other information required for processing a path. It is pointed to by the PLH (PLHWAX).

Work Area for Path Processing (WAX)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	1	WAXID	Control block identifier, X'73'
1 (1)	1	WAXFLG1	Flags:
	1...	WAXSRAB	Catalog recovery area built in system storage
	.1..	WAXPUG	The alternate index in the path is in the upgrade set
	..1.	WAXPS	The last operation against the path was a sequential PUT
	...x xxxx		Reserved
2 (2)	2	WAXLEN	Length of the WAX
4 (4)	2	WAXPL	Length of the alternate-index record's pointers to base records
6 (6)	2	WAXXXXX2	Reserved
8 (8)	4	WAXIRPL	Address of the inner ("dummy") RPL that is used to gain access to the alternate index
12 (C)	4	WAXURPL	Address of the user's RPL
16 (10)	4	WAXRCDA	Address of the alternate-index record
20 (14)	4	WAXXPTR	Address of the current alternate-index pointer to a base record
24 (18)	4	WAXEPTR	Address of the byte beyond the last alternate-index pointer
28 (1C)	4	WAXBPLH	Address of the PLH for the base cluster
32 (20)	4	WAXSRAA	Address of the saved-record area
36 (24)	4	WAXSRAL	Length of the saved-record area
40 (28)	4	WAXXXXX3	Reserved

WSHD—Working Storage Header

The WSHD describes up to four blocks of storage used for work areas (working storage). It is pointed to by the AMB (AMBWSHD).

Working Storage Header (WSHD)—Description and Format

Offset	Bytes and Bit Pattern	Field Name	Description
0 (0)	1	WSHDID	Control block identifier, X'44'
1 (1)	1	WSHDPOOL	The number of the subpool in which the WSHD is located
2 (2)	2	WSHDLEN	Length of the WSHD
4 (4)	4	WSHDNEXT	Address of the next WSHD
4 (4)	1	WSHDGMTB	GETMAIN resource byte
8 (8)	10	WSHDGMWA	GETMAIN work area
18 (12)	2	WSHDNUS	Number of used slots (entries) in the WSHD
20 (14)	4	WSHDGMRA	GETMAIN result (return code)
24 (18)	4	WSHDOCHN	Address of ordered slot chain
28 (1C)	16 x 4	WSHDSLT	Slot (entry) for each block of working storage:
28 (1C)	4	WSHDSAD	Address of the storage block
32 (20)	12	WSHDSGMW	Work area for the GETMAIN for the storage block:
32 (20)	4	WSHDSFM	FREEMAIN field for the DLVRP macro:
32 (20)	1	WSHDSFSP	The number of the subpool in which the storage block is located
33 (21)	3	WSHDSFLN	Length of the storage block
36 (24)	4	WSHDSONX	Address of the next slot on ordered slot chain
40 (28)	2	WSHDSBV	Number of bytes represented by each bit in WSHDSBM
42 (2A)	1	WSHDSFLG	Slot flags:
	1...	WSHDSFNO	The storage block has no bytes available
	.xxx xxxx		Reserved
43 (2B)	1	WSHDSBM	Bit mask (each bit indicates whether the bytes it represents are used—1, or not—0)

DIAGNOSTIC AIDS

This chapter provides several aids that can be useful when you are trying to diagnose difficulties with VSAM modules. These aids include:

- A description of *OS/VS2 VSAM Cross Reference,* which is published on microfiche cards.
- A list of messages issued by VSAM, with a list of the module(s) causing the message to be issued and a list of function codes for Open, Close, and End of Volume.
- A list of macros that VSAM uses and their functions.
- A description of GTF, how VSAM requests it, and what it provides in the way of VSAM APAR information.
- A list of return codes.
- A description of Virtual-Storage Management and its control blocks.
- A description of Open, Close, and End-of-Volume diagnostics.
- A list of ABENDs issued by VSAM.

Additional aids can be found in other parts of the book and in the program listings. These include:

- Register contents on entry to a module, which are under "INPUT" in the module prologues.
- Use of registers and equated names for registers, which can be found under "NOTES" in the module prologues.
- Error codes, which are under "EXIT-ERROR" in the module prologues.
- A list of modules, their external procedure names, their component, and their associated method of operation diagrams, which is in the "Module Directory."
- A list of external procedure names and their modules, which is in the "External Procedure Directory."
- Definitions of terms and abbreviations used in the book, which are in the "Glossary."
- Page references for the subjects covered in the book, which are in the "Index."

Microfiche Cross-Reference Aids

OS/VS2 VSAM Cross Reference (microfiche) contains valuable cross-reference information. Three reports are available:

- Symbol Where Used Report
- Macro Where Used Report
- Control Flow Report

How to Read the Symbol Where Used Report

The "Symbol Where Used Report" contains three kinds of information.

- A list of symbolic names. This includes field names, symbolic address names, return-code names, constant names, and flag-bit names, in alphabetic order from top to bottom.

  The lower-right corner of each page contains the first and last names listed on the page.

- A list of modules that refer to each symbolic name, in alphabetic order from left to right.
- A code indicating how each module refers to the symbolic name:

W	Write	The data field or bit value was modified by at least one line of code in this module. If the module contains a statement, "A = B" (that is not part of an IF statement), then the module's use of "A" is to *modify* it.
R	Read	The data field or value was referred to by at least one line of code in this module. If the module contains a statement "A = B," then the module's use of "B" is to *refer* to it.
C	Compare	The data field or value was compared with another field. If the module contains a statement, "If A=B, THEN...," then the module's use of "A" is to *compare* it with "B." (The module's use of "B" is to refer to it, not to compare it.)

Other codes are explained in the "Access Codes" at the bottom of each page in the table.

How to Read the Macro Where Used Report

The "Macro Where Used Report" contains three kinds of information.

- A list of macro names in alphabetic order from top to bottom.

  The lower-right corner of each page contains the first and last macros listed on the page.

- A list of the modules and macros that issue each macro in alphabetic order from left to right.
- The total number of times a macro is issued and the number of times each module in the list issues it.

How to Read the Control Flow Report

The middle of each page of the "Control Flow Report" contains an alphabetic list of all modules and procedures. To the left of each item are listed (under "FROM") all modules and procedures *from which* it gets control; to the right are listed (under "TO") all modules and procedures *to which* it gives control.

Next to each item on the left and on the right is a code (under "VIA") that indicates how control is passed. A key to the meaning of each code is given at the bottom of each page.

Unfortunately, control passed by way of a branch instruction is not indicated by the report.

Messages

Messages IDA001 through IDA025 refer to an incorrectly coded macro.

Message Number	**Message Text**
IDA001	INVALID POSITIONAL PARAMETER, *xxx* - IGNORED
IDA002	*xxx* KEYWORD REQUIRED - NOT SPECIFIED
IDA003	INVALID VALUE, *yyy*, SPECIFIED FOR *xxx* KEYWORD
IDA004	*xxx* KEYWORD NOT VALID FOR EXECUTE FORM - IGNORED
IDA005	INVALID OR DUPLICATE SUBLIST ITEM FOR *xxx* KEYWORD, *yyy*
IDA006	*xxx* VALUE, *yyy*, NOT VALID FOR LIST FORM
IDA007	LOGIC ERROR IN MACRO *xxx*
IDA008	INCOMPATIBLE SUBLIST ITEMS, *yyy* AND *zzz,* FOR *xxx* KEYWORD
IDA009	*xxx* CONTROL BLOCK KEYWORDS SPECIFIED - ONLY ONE ALLOWED
IDA010	EXIT ADDRESS REQUIRED FOR *xxx* KEYWORD - NOT SPECIFIED
IDA011	*xxx* IS NOT A VALID *yyy* KEYWORD - IGNORED
IDA018	VTAM KEYWORD, *xxx,* SPECIFIED WITHOUT SPECIFYING AM=VTAM
IDA019	KEYWORDS *xxx* AND *yyy* ARE INCOMPATIBLE
IDA020	VTAM SUBLIST ITEM, *xxx,* SPECIFIED FOR *yyy* KEYWORD WITHOUT SPECIFYING AM=VTAM
IDA02I	*xxx* and *yyy* KEYWORDS MUST BE SPECIFIED TOGETHER BUT ONE IS MISSING
IDA022	CONFLICTING SUBLIST ITEMS WERE SPECIFIED FOR *xxx* KEYWORD
IDA024	*xxx,* A VSAM KEYWORD SPECIFIED FOR A NON-VSAM CONTROL BLOCK
IDA025	*www, yyy, zzz* CONFLICTING SUBPARAMETERS IN *xxx* KEYWORD, *www* ASSUMED

		Detected By	**Issued By**
IEC001A	M *ddd,ser,jjj,sss,dsn*	IDA0192V	IDA0192V
IEC003E	R *ddd,ser,jjj,sss,* [,SPACE=PRM] *,dsn*	IDA0192V	IDA0192V
IEC014E	D *dddd*	IDA0192V	IDA0192V
IEC070I	*rc*[(*sfi*)]-*ccc,jjj,sss,ddn, ddd,vol,cln,dsn,cat* *(IEC070I is an End-of-Volume message.)*	IDA0192D IDA0192S IDA0192V IDA0557A IFG0551F	IDA0192P
IEC101A	M *ddd,ser,jjj,sss,dsn*	IGG0CLBL	IDA0192V
IEC111E	D, *ddd,ser*	IGG0CLBL	IDA0192V
IEC159I	E13-*rc,mod,jjj,sss,ddn* [-#], *ddd*		
IEC161I	*rc*[(*sfi*)]-*ccc,jjj,sss,ddn, ddd,vol,cln,dsn,cat* *(IEC161I is an Open message.)*	IDA0192A IDA0192C IDA0192D IDA0192S IDA0192V IDA0192Z IDA0192A IFG0192A	IDA0192P

Message Number	Message Text	Detected By	Issued By
IEC251I	*rc*[(*sfi*)]-*ccc,jjj,sss,ddn ddd,vol,cln,dsn,cat* *(IEC251I is a Close message.)*	IDAOCEA2 IDA0192C IDA0192D IDA0192S IDA0192V IDA0200T IFG0200V	IDA0192P
IEC252I	*rc*[(*sfi*)]-*ccc,jjj,sss,ddn, ddd,vol,cln,dsn,cat* *(IEC252I is a Close (TYPE=T) message.)*	IDA0192C IDA0192D IDA0192S IDA0192V IDA0231T IGC0002C	IDA0192P
IEC331I	*rc-crs,jjj,sss,func,mmm*		
IEC332I	*func*[*func...*]		
IEC333I	*terr,xx,cat,yyy*		
	IEF175IAMP KEYWORD *nnnnnnnn* DUPLICATE OR CONFLICTING PARM STEP NOT EXECUTED	IEFVAMP	
IEF447I	AMP KEYWORD *nnnnnnnn* IS INVALID STEP WAS NOT EXECUTED	IEFVAMP	
IEF448I	AMP KEYWORD *nnnnnnnn* VALUE *xxxxxx* IS TOO LARGE STEP NOT EXECUTED	IEFVAMP	
IEF449I	AMP KEYWORD *nnnnnnnn* REQUIRES A DECIMAL VALUE STEP NOT EXECUTED		
IHJ000I	CHKPT *jjj* (*ddn*) NOT TAKEN (*xx*)		
IHJ001I	*jjj* (*ddn,ddd,vol*) INVLD *checkid* (*xxx*)		
IHJ002I	*jjj* (*ddn,ddd,vol*) ERROR *checkid* (*xxx*)		
IHJ004I	*jjj* (*ddn,ddd,vol*) CHKPT *checkid*		
IHJ005I	*jjj* (*ddn,ddd,vol*) ENQS *checkid*		
IHJ006I	*jjj* RESTARTING at *xxxxxx yyyyyy*		
IHJ007I	RESTART NOT SUCCESSFUL FOR *jjj* (*xxx* [,*cuu*])		
IHJ008I	*jjj* RESTARTED		
IHJ009I	ERROR ON *ddn*	IDA0A05B	
IHJ010I	CHECKPOINT RESTART OF JOB *jjj* ABENDED		

Function Codes for VSAM Open, Close, and End-of-Volume Messages

When an error occurs during Open, Close, or End-of-Volume processing for a VSAM data set, the message that is issued will contain (besides error identification, job, step, and DD names, device address, volume serial number, and names of cluster, data set, and catalog) a field, ccc, that contains a function code. The following lists these function codes and ties each to the module that detected the error and the operation being performed when the error was detected.

Function Code	Module that Detected Error	Operation Being Performed When Error Was Detected
Open		
1	IDA0192C	Initialize for catalog interface processing.
2	IDA0192C	Determine which data sets are associated with data-set name on DD statement, determine catalog, and check password.
3	IDA0192C	Determine data-set attributes.
4	IDA0192C	Get volume information.
5	IDA0192C	Update "open" indicator in catalog.
6	IDA0192C	Update catalog when data set is being closed.
7	IDA0192C	Retrieve volume timestamp.
8	IDA0192C	Record-Management catalog update.
9	IDA0192C	Update preformat indicator in catalog.
10	IDA0192C	Retrieve 44-byte cluster name.
11	IDA0192C	Retrieve 44-byte component name.
20	IDA0192V	Initialize for mounting and verify volume.
21	IDA0192V	Check volume timestamp.
22	IDA0192V	Handle messages.
23	IDA0192V	Mount volume.
30	IDA0192S	Initialize for SMF processing.
31	IDA0192S	Build SMF record.
40	IDA0192D	Initialize for staging.
41	IDA0192D	Build UCB list.
42	IDA0192D	Build list for ACQUIRE/RELINQUISH (stage/destage).
43	IDA0192D	Issue ACQUIRE or RELINQUISH.
50	IDA0192Z	Initialize for building control blocks.
51	IDA0192Z	Determine number of buffers needed.
52	IDA0192Z	Build buffers.
53	IDA0192Z	Build control blocks.
54	IDA0192Y	Build string blocks.
60	IDA0192B	Module initialization.
61	IDA0192B	Locate data-set attributes and check them for validity.
62	IDA0192B	Volume processing.
63	IDA0192B	Preformat extent.
70	IDA0192W	Initialize for building channel program.
71	IDA0192W	Build channel program area.
80	IFG0193A	Check return codes from IFG0191X or IFG0191Y.
81	IDA0192A	Initialize for VSAM Open processing.
82	IDA0192A	Verify ACB.
83	IDA0192F	Fix control blocks in real storage.
84	IDA0192B	Allow subtasks to share data set.
85	IDA0192F	Mount and verify volumes.
87	IDA0192A	Determine whether to connect base cluster to an existing structure or generate a new structure.

Function Code	Module that Detected Error	Operation Being Performed When Error Was Detected
88	IDA0192F	Open base cluster.
89	IDA0192F	Open alternate index in upgrade set.
90	IDA0192F	Open alternate index in path.
93	IDA0192A	Build a dummy DEB.
95	IDA0192A	Terminate VSAM Open processing.
96	IDA0192A	Clean up after an error in Open processing.
99	IFG0192B	Error processing for ACB being processed on a system not generated for VSAM.
Close		
100	IFG0200V	Read JFCB.
101	IDA0200T	Initialize for VSAM Close processing.
103	IDA0200T	Complete deferred write requests.
104	IDA0200T	Close path.
105	IDA0200T	Close base cluster.
106	IDA0200T	Close sphere (close upgrade alternate indexes and free storage).
107	IDA0200T	Close upgrade set.
108	IDA0200T	Process volume mount table.
110	IDA0200B	Module initialization.
111	IDA0200B	Check validity of AMBLs and DEBs.
112	IDA0200B	SMF processing.
113	IDA0200B	Update statistics and RBA information in the catalog.
114	IDA0200B	Free storage for control blocks.
115	IDA0200B	Write a buffer.
148	IDA0200T	Force deletion of VSAM global resource pool.
149	IDAOCEA2	Force deletion of VSAM global resource pool.
150	IGC0002C	Read JFCB.
151	IDA0231T	Initialize for VSAM Close (TYPE=T) processing.
153	IDA0231T	Complete deferred write requests.
154	IDA0231T	Close (TYPE=T) path.
155	IDA0231T	Close (TYPE=T) base cluster.
156	IDA0231T	Close (TYPE=T) upgrade set.
157	IDA0231B	Module initialization.
158	IDA0231B	Check validity of AMBLs and DEBs.
159	IDA0231B	Update statistics and RBA information.
160	IDA0231B	SMF processing.
161	IDA0231B	Write a buffer.
End of Volume		
200	IFG0551F	Read JFCB.
201	IDA0557A	Initialize for VSAM End-of-Volume processing.
202	IDA0557A	Locate and mount volume.
203	IDA0557A	Allocate space.

Function Code	Module that Detected Error	Operation Being Performed When Error Was Detected
204	IDA0557A	Switch volumes.
205	IDA0557A	Build control blocks.
206	IDA0557A	Update SMF record.
207	IDA0557A	Preformat extent.
208	IDA0557A	Record Management, catalog update.
209	IDA0557A	Reset control blocks.

Macros

The following tables list VSAM and OS/VS macros and explain what they do. The macros are divided into those that define control blocks and data areas (mapping macros) and those that issue executable code (action macros).

OS/VS2 VSAM Cross Reference (microfiche) has a table ("Macro Where Used Report") of all the macros issued by VSAM with a listing of the modules (and other macros) that issue them.

Mapping Macros

The following table lists macros that define the format of control blocks and data areas used by VSAM modules.

Macros That Define Data Areas

Macro	Description
ACB	Builds an access-method control block (ACB) at assembly time
CVT	Maps the communication vector table (CVT)
ECB	Maps the event control block
IDAAIR	Maps the alternate-index record
IDAAMB	Maps the access method block (AMB)
IDAAMBL	Maps the access method block list (AMBL)
IDAAMBXN	Maps the access method block extension (AMBXN)
IDAAMDSB	Maps the access method data set statistics control block (AMDSB)
IDAARDB	Maps the address range definition block (ARDB)
IDAARWA	Maps a recovery work area for the Restart modules
IDABIB	Maps the base information block (BIB)
IDABFR	Maps the buffer control set
IDABLPRM	Maps the resource pool parameter list (BLPRM)
IDABSPH	Maps the buffer subpool header (BSPH) for shared resources
IDABUFC	Maps the buffer control block (BUFC)
IDACBTAB	Maps the tables used by the Control Block Manipulation routine
IDACIDF	Maps the control-interval descriptor field (CIDF)
IDACLWRK	Maps the close work area
IDACMB	Maps the cluster management block (CMB)
IDACPA	Maps the channel program area (CPA)
IDACSL	Maps the core save list (CSL)

Macros That Define Data Areas (continued)

Macro	**Description**
IDACTREC	Maps the work area built when the VS2 Catalog-Management routines use Open, Close, or End of Volume
IDADIWA	Maps the data insert work area (DIWA)
IDADSECT	Maps miscellaneous data areas for Checkpoint/Restart
IDADSL	Maps the DEB save list (DSL)
IDAEDB	Maps the extent definition block (EDB)
IDAELEM	Maps the Control Block Manipulation routine's element argument control entry
IDAEQUS	Defines the equates for the ISAM Interface: SYNAD—Message-Build routine
IDAERMSG	Maps the ISAM Interface: SYNAD Message format
IDAERRCD	Lists the VSAM Open and Close ACB Error Codes
IDAESL	Maps the enqueue save list (ESL)
IDAFOREC	Maps the work area for VSAM Open, Close, and End of Volume (the work area is called "FORCORE" in program comments)
	IDAFOREC issues IDAPDPRM, IEFJFCBN, and IEFJFCBX.
IDAGENC	Maps the GENCB header argument control entry
IDAHEB	Maps the header element block (HEB)
IDAICWA	Maps the index create work area (ICWA)
IDAIDXCB	Lists the VSAM control-block-identifier codes
IDAIICB	Maps the ISAM-Interface control block (IICB)
IDAIIREG	Defines the ISAM-Interface register usage
	IDAIIREG issues IDAIICB, IDARPLE, IFGRPL, IHADCB, and IHADCBDF.
IDAIMWA	Maps the index modification work area (IMWA)
IDAIOB	Maps the VSAM IOB extension
IDAIOMB	Maps the I/O-Management control block (IOMB)
IDAIOSCN	Maps the VSAM Open, Close, and End-of-Volume commonly-used declarations
IDAIRD	Defines the index record
IDAIXSPL	Maps the index search parameter list (IXSPL)
IDALPMB	Maps the logical-to-physical mapping block (LPMB)
IDAMODC	Maps the MODCB header argument control entry
IDAOPWRK	Maps the ACB work area for Open (OPW or OPWRK)
IDAPDPRM	Maps the VSAM Open, Close, and End-of-Volume problem determination parameter list
IDAPLH	Maps the placeholder (PLH)
IDAPSL	Maps the page save list (PSL)
IDARDF	Maps the record definition field (RDF)
IDAREGS	Defines register usage for all Record-Management modules
IDARMRCD	Lists the Record-Management return codes
IDARPLE	Maps the ISAM-Interface request parameter list extension (RPLE)
IDARTMAC	Maps data structures for recovery routines
IDASHOW	Maps the SHOWCB header argument control entry

Macros That Define Data Areas (continued)

Macro	Description
IDASSL	Maps the swap save list (SSL)
IDATEST	Maps the TESTCB header argument control entry
IDAUPT	Maps the upgrade table (UPT) for upgrading alternate indexes
IDAVAT	Maps the valid-AMBL table (VAT)
IDAVCRT	Maps the VSAM checkpoint/restart table (VCRT), the VSAM checkpoint/restart storage blocks (VCRCORE), and the HEB save area (VCRHEBSA).
IDAVGTT	Maps the VSAM global termination table (VGTT)
IDAVIOT	Maps the valid-IOMB table (VIOT)
IDAVMT	Maps the volume mount table (VMT)
IDAVSRT	Maps the VSAM shared resource table (VSRT)
IDAVUCBL	Maps the VSAM Open and End of Volume: Volume Mount and Verify UCB list
IDAVVOLL	Maps the VSAM Open and End of Volume: Volume Mount and Verify volume serial number list
IDAWAX	Maps the work area for path processing (WAX)
IDAWSHD	Maps the working storage header (WSHD)
IECDIOCM	Maps the communication area of the VS2 I/O Supervisor
IECDIOSB	Maps the I/O-Supervisor control block (IOSB)
IECDIPIB	Maps the I/O Supervisor—Purge interface block (IPIB)
IECDSECS	Maps the DSECS
IECDSECT	Maps the common open/close work area
IECRRPL	Maps the common O/C/EOV Recovery Routine parameter list
IECSDSL1	Maps the SDSL1
IEESMCA	Maps the SMCA
IEEVCHWA	Maps a work area that VS2 Checkpoint passes to VSAM Checkpoint
IEEVRSWA	Maps a work area that VS2 Restart passes to VSAM Restart
IEFJFCBN	Maps the job file control block (JFCB)
IEFJFCBX	Maps the job file control block (JFCB)
IEFJMR	Maps the JMR
IEFTCT	Maps the TCT
IEFTIOT1	Maps the task input/output table (TIOT)
IEFUCBOB	Maps the VS2 unit control block (UCB)
IEZABP	Maps the ABP—I/O-Management communication vector table (module IDA121CV)
IEZCTGFL	Maps the VS2 catalog field parameter list (CTGFL)
IEZCTGPL	Maps the VS2 catalog parameter list (CTGPL)
IEZDEB	Maps the VS2 data extent block (DEB)
IEZIOB	Maps the VS2 input/output block (IOB)
IEZJSCB	Maps the VS2 job step control block (JSCB)
IFGACB	Maps the access-method control block (ACB)
IFGEXLST	Maps the exit list (EXLST)
IFGRPL	Maps the request parameter list (RPL)
IGGCAXWA	Maps the VS2 catalog auxiliary work area (CAXWA)

Macros That Define Data Areas (continued)

Macro	Description
IHAASCB	Maps the VS2 address space control block (ASCB)
IHAASXB	Maps the VS2 address space extension control block (ASXB)
IHADCB	Maps the VS2 data set control block (DCB)
IHADCBDF	Maps the VS2 data set control block (DCB)
IHADECB	Maps the VS2 data extent control block (DECB)
IHADSAB	Maps the VS2 data set association block (DSAB)
IHAFRRS	Maps VS2 Recovery Termination Manager Dsects for function recovery routines
IHAIQE	Maps the VS2 interrupt queue element (IQE)
IHAPSA	Maps the VS2 prefixed save area (PSA)
IHAPVT	Maps the VS2 page vector table (PVT)
IHARB	Maps the VS2 request block (RB)
IHARMPL	Maps the VS2 Resource Manager's parameter list for interfacing with VSAM Task Close Executor (IDA0CEA2)
IHASDWA	Maps the STAE diagnostic work area (SDWA, also called the recovery termination communication area—RTCA)
IHASRB	Maps the VS2 service request block (SRB)
IHJSSCR	Maps the subsystem control record of areas saved at checkpoint time
IKJRB	Maps request blocks
IKJTCB	Maps the task control block (TCB)
XCTLTABL	Maps the VS2 XCTL table

Action Macros

This table lists the macros issued by VSAM that generate executable code.

Macros That Generate Executable Code

Macro	Description
ABEND	Abnormal termination (VS2 macro)
BLDVRP	Builds a VSAM resource pool for shared resources
CATLG	Loads the address of the catalog parameter list (CTGPL) into register 1 and issues SVC 26
CLOSE	VSAM CLOSE: Disconnects a user from a VSAM data set
DEBCHK	Checks the validity of the DEB
DELETE	(Same as VS2 DELETE macro)
DEQ	(Same as VS2 DEQ macro)
DLVRP	Deletes a VSAM resource pool for shared resources
DOM	Deletes operator message (VS2 DOM macro)
ENDREQ	Terminates a VSAM record processing request (such as GET or PUT)
ENQ	(Same as VS2 ENQ macro)
ERASE	Deletes a VSAM record
ESTAE	Specifies task asynchronous exit (VS2 macro)
EXCP	(Same as VS2 EXCP macro)
FREEMAIN	Releases virtual storage obtained by a GETMAIN
GENCB	Generates a VSAM control block (ACB, EXLST, or RPL)

Macros That Generate Executable Code (continued)

Macro	Description
GET	Retrieves a record from a data set on a direct-access device
GETIX	Retrieves a control interval from the index of a key-sequenced data set
GETMAIN	Obtains virtual storage for a temporary work area
GTRACE	Calls the Generalized Trace Facility (GTF) to copy VSAM control blocks
IDACALL	Transfers control from procedure A to procedure B and allows procedure B to return control to procedure A at the instruction following the IDACALL instruction-expansion
IDACB1	Transforms operands for Control Block Manipulation macros (GENCB, MODCB, SHOWCB, and TESTCB)
IDACB2	Scans keywords and generates code for Control Block Manipulation macros
IDAERMAC	Prints MNOTEs for Control Block Manipulation macro user-programmer errors
IDAEXITR	Transfers control from VSAM modules to a user's exit routine and allows the user exit routine to return control to the VSAM module at the instruction following the IDAEXITR instruction-expansion IDAEXITR issues DELETE, IDARST14, IDASVR14, and LOAD.
IDAGMAIN	Gets virtual storage for VSAM Open, Close, and End of Volume
IDAPATCH	Generates maintenance space
IDAPFMT	Gives control to End of Volume to preformat a control area
IDARST14	Puts the return address in register 14
IDASVR14	Saves register 14 in the placeholder (PLH) push-down list
IECRES	Transfers control to the VS2Resident routine
LOAD	(Same as VS2 LOAD macro)
MODCB	Modifies a VSAM control block (ACB, EXLST, or RPL)
MODESET	(Same as VS2 MODESET macro)
MRKBFR	Marks a buffer in a VSAM resource pool
OBTAIN	(Same as VS2 OBTAIN macro)
OPEN	Connects a user's program to a VSAM data set
PGFIX	"Fixes" a page of virtual storage so that it remains in real storage for a duration
PGFREE	"Frees" a "fixed" page of virtual storage.
POINT	Identifies a starting point in a VSAM data set
POST	(Same as VS2 POST macro)
PUT	Writes a record into a VSAM data set
PUTIX	Writes a control interval in the index of a key-sequenced data set
RESERVE	(Same as VS2 RESERVE macro)
RETURN	(Same as VS 2 RETURN macro)
SCHBFR	Searches for a control interval in a VSAM resource pool
SDUMP	Schedules SVC dump routine (VS2 macro)
SETFRR	Sets up functional recovery routine (VS2 macro)
SETLOCK	Obtains or releases a lock (VS2 macro)
SETRP	Records recovery information (VS2 macro)
SHOWCAT	Displays information from a VSAM catalog

Macros That Generate Executable Code (continued)

Macro	Description
SHOWCB	Displays information from a VSAM control block (ACB, EXLST, or RPL)
SMFWTM	Writes the SMF message into the SMF data set
STARTIO	Gives control to the VS2 I/O Supervisor to start an I/O operation
SYNCH	(Same as ISAM SYNCH macro)
TESTAUTH	Checks authorization of a calling module to perform certain functions
TESTCB	Tests information in a VSAM control block (ACB, EXLST, or RPL)
TIME	Obtains the correct time from the VS2 system time-of-day clock
VERIFY	Gives control to Record Management to check the end-of-data indicators for Checkpoint/Restart or for Access Method Services VERIFY command
WAIT	(Same as VS2 WAIT macro)
WRTBFR	Writes buffers from a VSAM resource pool
WTO	Writes a message to the operator (no reply)
XCTL	Transfers control (VS2 XCTL macro)

Note: The use of these user macros is described in *OS/VS Virtual Storage Access Method (VSAM) Programmer's Guide:*

CLOSE
ENDREQ
ERASE
GENCB
GET
MODCB
OPEN
POINT
PUT
SHOWCB
TESTCB

The use of these user macros is described in *OS/VS Virtual Storage Access Method (VSAM) Options for Advanced Applications:*

BLDVRP
DLVRP
GETIX
MRKBFR
PUTIX
SCHBFR
SHOWCAT
WRTBFR

Generalized Trace Facility

The Generalized Trace Facility (GTF) can be used to record information about VSAM processing at the time of an error. If GTF is active in the VS2 system, GTF is used to trace VSAM control blocks when there is an error.

GTF is used to record the contents of the ACB, AMBL, AMBs, AMDSBs, and TIOT entry for the data set being processed when the error occurred.

To format and print GTF records, use the IMDPRDMP service aid with 'USR=(FFF,FF5)' specified in the EDIT statement.

Two types of traces are available to help in debugging VSAM Open/Close/End-of-Volume problems:

- The error trace routine traces VSAM control blocks when an error is detected. The optional work area trace traces the Open/Close/End-of-Volume work area WTG table prefix and the current entry in the WTG table at entry to and exit from the VSAM Open/Close/End-of-Volume modules.
- The work area trace is requested by specifying AMP='TRACE'. This is the same trace that is obtained for nonVSAM Open/Close/End-of-Volume processing when DCB=DIAGNS=TRACE is specified in the JCL. (For details on the AMP and DCB JCL parameters and options, see *OS/VS2 JCL.*

Both traces require that GTF be operating in external mode while the job to be traced is running. In addition, the operator must respond with "TRACE=USR" when the GTF trace message"SPECIFY TRACE OPTIONS" appears at the operator's console.

Additional information on GTF and IMDPRDMP is contained in the *OS/VS2 System Programming Library: Service Aids.*

Return Codes

VSAM sets return codes in the RPL and the ACB. These codes are paired with codes in register 15. Codes set in the RPL are listed and explained under "Return Codes from the Record-Management (Request) Macros." Those set in the ACB, which indicate open or close errors, are listed and explained under "Open and Close Return Codes."

VSAM sets a pair of codes in registers 15 and 0 for the control block manipulation macros. These are listed and explained under "Control Block Manipulation Return Codes."

Return Codes from the Record-Management (Request) Macros

After a request macro or a CHECK or ENDREQ macro is issued, register 15 contains a return code.

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

Paired with the 0, 8, and 12 indicators in register 15 are return codes in the feedback field of the request parameter list.

The feedback return codes for the 0 indicator in register 15, which doesn't cause VSAM to exit to an exit routine, are:

RPLFDBK Code	Condition
0(0)	Request completed successfully.
4(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(8)	Duplicate alternate key follows.
12(C)	(Shared resources only.) A buffer needs to be written.

See the discussions below for the logical-error return codes and for the physical-error return codes.

Function Codes for Logical and Physical Errors

When a logical or physical error occurs during processing that involves alternate indexes, VSAM provides a code in the RPLCMPON field that indicates whether the base cluster, its alternate index, or its upgrade set was being processed and whether upgrading was okay or might have been incorrect because of the error:

Code	What Was Being Processed	Status of Upgrading
0(0)	Base cluster	Okay
1(1)	Base cluster	Might be incorrect
2(2)	Alternate index	Okay
3(3)	Alternate index	Might be incorrect
4(4)	Upgrade set	Okay
5(5)	Upgrade set	Might be incorrect

Logical-Error Return Codes

When a logical-error-analysis exit routine (LERAD) is provided, it gets control for logical errors, and register 15 doesn't contain 8, but contains the entry address of the LERAD routine.

Figure 57 gives the contents of the registers when VSAM exits to the LERAD routine.

If a logical error occurs and a LERAD exit routine isn't provided (or the LERAD exit is inactive), VSAM returns control to the processing 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.

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 the exit routine returns to VSAM.
2-13	Same as when the request macro was issued. Register 13, by convention, contains the address of the 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 57. Contents of Registers When a LERAD Routine Gets Control

Figure 58 gives the logical-error return codes in the feedback field and explains what each one means.

RPLFDBK Code	Condition
4(4)	End of data set encountered (during sequential retrieval). Either no EODAD routine is provided, or one is provided and it returned to VSAM and the processing program issued another GET. Detected by: IDA019RA, IDA019RD, IDA019RR, IDA019RY IDA019R2, IDA019R4, IDA019R8
8(8)	Attempt was made to store a record with a duplicate key. Detected by: IDA019RA, IDA019RQ, IDA019RX, IDA019R4
12(C)	Attempt was made to store a record out of ascending key sequence; record may also have a duplicate key. Detected by: IDA019RA, IDA019RR, IDA019RX, IDA019R4
16(10)	Record not found. Detected by: IDA019RA, IDA019RR, IDA019RY
20(14)	Record already held in exclusive control by another requester. Detected by: IDA019RF, IDA019RY, IDA019R2, IDA019R8
24(18)	Record resides on a volume that can't be mounted. Detected by: IDA019RW, IDA019RY, IDA019R2, IDA019R5
28(1C)	Data set cannot be extended because VSAM can't allocate additional direct-access storage space. Either there isn't enough space left in the data space for the secondary-allocation request or an attempt was made to increase the size of a data set during processing with SHROPT=4 and DISP=SHR. Detected by: IDA019R5
32(20)	An RBA was specified that doesn't give the address of any data record in the data set. Detected by: IDA019RA, IDA019R8
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. Detected by: IDA019RM
40(28)	Insufficient virtual storage in the address space to complete the request. Detected by: IDA019RG, IDA019RU, IDA019RX
44(2C)	Work area not large enough for the data record (GET with OPTCD=MVE). Detected by: IDA019RR, IDA019RT, IDA019RY, IDA019R4, IDA019R8
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. Detected by: IDA019RU, IDA019RX, IDA019R1
68(44)	Attempt was made to use a type of processing (output or control-interval processing) that was not specified when the data set was opened. Detected by: IDA019RQ, IDA019R4, IDA019R8
72(48)	A keyed request for access was made to an entry-sequenced data set or a GETIX or PUTIX was issued to an entry-sequenced or relative record data set. Detected by: IDA019R1, IDA019R8

Figure 58 (Part 1 of 3). Logical-Error Return Codes in the RPL Feedback Field from a Request Macro

RPLFDBK
Code **Condition**

76(4C) An addressed or control-interval PUT was issued to add a record to a key-sequenced data set, or a control-interval PUT was issued to a relative record data set.

Detected by: IDA019R1, IDA019R8

80(50) An ERASE request was issued for access to an entry-sequenced data set.

Detected by: IDA019RL, IDA019RX, IDA019R8

84(54) OPTCD=LOC was specified for a PUT request or in a request parameter list in a chain of request parameter lists.

Detected by: IDA019RQ, IDA019R1, IDA019R4, IDA019R8

88(58) A sequential GET or PUT request was issued without VSAM having been positioned for it, or a change was made from addressed access to keyed access without VSAM having been positioned for keyed sequential retrieval, or an illegal switch between forward and backward processing was attempted.

Detected by: IDA019RQ, IDA019RR, IDA019R4, IDA019R8

92(5C) A PUT for update or an ERASE was issued without a previous GET for update, or a PUTIX was issued without a previous GETIX.

Detected by: IDA019RQ, IDA019RX, IDA019R4, IDA019R8

96(60) Attempt was made to change a key during an update.

Detected by: IDA019RL, IDA019RX

100(64) Attempt was made to change the length of a record during an addressed update.

Detected by: IDA019RL, IDA019RQ

104(68) The RPL options are either invalid or conflicting in one of the following ways:

- SKP was specified and either KEY wasn't specified or BWD was specified
- BWD was specified for CNV processing
- FWD and LRD were specified
- Neither ADR, CNV, nor KEY was specified in the RPL
- WRTBFR, MRKBFR, or SCHBFR was issued, but either TRANSID was greater than 31 or a shared-resources option wasn't specified
- ICI processing was specified, but a request other than a GET or a PUT was issued

Detected by: IDA019RA, IDA019RR, IDA019RY, IDA019RX,
IDA019R1, IDA019R4, IDA0198

108(6C) RECLEN specified was larger than the maximum allowed, equal to 0, smaller than the sum of the length and the displacement of the key field, or not equal to record (slot) length specified for a relative record data set.

Detected by: IDA019RL, IDA019RQ, IDA019RU,
IDA019R4, IDA019R8

Figure 58 (Part 2 of 3). Logical-Error Return Codes in the RPL Feedback Field from a Request Macro

RPLFDBK Code	Condition
112(70)	KEYLEN specified was too large or equal to 0. Detected by: IDA019R1
116(74)	A GET, POINT, ERASE, direct PUT, skip sequential PUT, or PUT with OPTCD=UPD not permitted during initial data-set loading (that is, for storing records in the data set the first time it's opened). Detected by: IDA019RR, IDA019R4, IDA019R8
132(84)	An attempt was made in locate mode to retrieve a spanned record. Detected by: IDA019RT
136(88)	An addressed GET was issued for a spanned record in a key-sequenced data set. Detected by: IDA019RT
140(8C)	Inconsistent spanned-record segments. Detected by: IDA019R4
144(90)	Invalid pointer in an alternate index (no associated base record). Detected by: IDA019RX
148(94)	The maximum number of pointers in the alternate index has been exceeded. Detected by: IDA019RU
152(98)	(Shared resources only.) Not enough buffers are available to process the request. Detected by: IDA019RY
192(C0)	Invalid relative record number. Detected by: IDA019RQ, IDA019RR
196(C4)	An addressed request was issued to a relative record data set. Detected by: IDA019R1
200(C8)	Addressed or control-interval access was attempted by way of a path. Detected by: IDA019RX
204(CC)	PUT-insert requests are not allowed in backward mode. Detected by: IDA019RQ, IDA019R4

Figure 58 (Part 3 of 3). Logical-Error Return Codes in the RPL Feedback Field from a Request Macro

Physical-Error Return Codes

When a physical-error-analysis exit routine (SYNAD) is provided, it gets control for physical errors, and register 15 doesn't contain 12, but contains the entry address of the SYNAD routine.

Figure 59 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 a request macro was issued, the RPL is the one pointed to by the request macro; if a CLOSE macro was issued,the RPL was built by VSAM to process the close request. Register 1 must contain this address if the exit 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 the 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 59. Contents of Registers When a SYNAD Routine Gets Control

If a physical error occurs and a SYNAD exit routine isn't provided (or the SYNAD exit is inactive), VSAM returns control to the processing program following the last executable instruction. Register 15 indicates a physical error (12), and the feedback field in the request parameter list contains a code identifying the error. Register 1 points to the request parameter list.

Figure 60 gives the physical-error return codes in the feedback field and explains what each one indicates. If the user provided a message area, it contains a physical-error message with more details about the error.

RPLFDBK Code	Condition
4(4)	Read error occurred for a data component.
8(8)	Read error occurred for the index set of an index component.
12(C)	Read error occurred for the sequence set of an index component.
16(10)	Write error occurred for a data component.
20(14)	Write error occurred for the index set of an index component.
24(18)	Write error occurred for the sequence set of an index component.
	All physical errors are detected by IDA019R5 from I/O Management abnormal-end appendage, IDA121A4.

Figure 60. Physical-Error Return Codes in the RPL Feedback Field from a Request Macro

Figure 61 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 message)	4	5	2	Binary value of 124 (provided for compatibility with SYNADAF
	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 (,)
Data-Set 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 (,)
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 occasioned the error in the first two bytes, followed by "-OP"
	90		1	Comma (,)

Figure 61 (Part 1 of 2). Format of Physical-Error Messages

Field	Bytes		Length	Discussion
Message	91	105	15	Messages are divided according to ECB condition codes: X‘41’— “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, this 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 (,)
Physical Direct-Access Address	107	120	14	BBCCHHR (bin, cylinder, head, and record)
	121		1	Comma (,)
Access Method	122	127	6	“VSAM”

Figure 61 (Part 2 of 2). Format of Physical-Error Messages

Open and Close Return Codes

When a processing program receives control after it has issued an OPEN or CLOSE macro, register 15 indicates whether all of the data sets were opened or closed successfully:

Reg 15	Condition
0(0)	All data sets were opened or closed successfully.
4(4)	Open: all data sets were opened successfully, but one or more warning messages were issued (ACBERFLG codes less than X‘80’). Close: at least one data set (VSAM or nonVSAM) was not closed successfully.
8(8)	Open: at least one data set (VSAM or nonVSAM) was not opened successfully; if there was an error for an ACB, it was restored to the contents it had before OPEN was issued.
12(C)	Open: at least one data set (VSAM or nonVSAM) was not opened successfully; if there was an error for an ACB, it was not restored to the contents it had before

OPEN was issued (and the data set cannot be opened without the ACB's being restored).

ACBERFLG Code	Condition
0(0)	When register 15 contains 0: All data sets were opened or closed successfully When register 15 contains 8: Either VSAM is processing the ACB for some other request, Or DDNAME was not specified in the ACB
4 (4)	Warning message: the ACB is already opened (and the user issued OPEN), or the ACB is already closed (and the user issued CLOSE or temporary CLOSE).
96 (60)	Warning message: an unusable data set was opened for input. Detected by: IDA0192B
100 (64)	Warning message: Open encountered an empty alternate index that is part of an upgrade set. Detected by: IDA0192B
104 (68)	Warning message: the timestamp for the volume does not match the timestamp in the catalog record for the data set. (This may mean the cluster existing on the volume(s) is not accurately described by its catalog record.) Detected by: IDA0192A
108 (6C)	Warning message: the timestamp for the index is less than the timestamp for the data set. (This could occur if the data set was updated without the index being open.) Detected by: IDA0192B
116 (74)	Warning message: the last request to close this data set was not completed successfully. Detected by: IDA0192B
128 (80)	DDNAME not found in TIOT.
132 (84)	An I/O error was detected while the system was reading the JFCB. Detected by: IDA0192F, IDA0200V
136 (88)	Not enough storage was available for work areas, buffers, or control blocks. Detected by: IDA0192A, IDA0192B, IDA0192C, IDA0192F, IDA0192W, IDA0192Y, IDA0192Z, IDA0200B, IDA0200T, IDA0231B, IDA0231T
144 (90)	An I/O error occurred while a catalog record was being read or written. A return code was set by a VS2 Catalog-Management routine. Detected by: IDA0192C
148 (94)	The catalog entry for the data set being opened or closed was not found or an unidentified error occurred while VSAM was searching the catalog. Detected by: IDA0192C
152 (98)	The data set being opened is protected by a password, and the VSAM Open routine was unable to validate the password. Detected by: IDA0192C
160 (A0)	The buffer space specified was not consistent with the buffer requirements of the data set; or the ACB indicated keyed access, but the data set is not a key-sequenced data set; or the device type specified in the DD statement is not consistent with the device type indicated in the catalog entry for the data set; or user buffering is

ACBERFLG Code	Condition
	specified in the ACB's MACRF field and control-interval processing should be specified, but is not. Detected by: IDA0192A, IDA0192B, IDA0192C, IDA0192Z
164 (A4)	The system detected an I/O error while reading the volume label and format-4 DSCB. Detected by: IDA0192F
168 (A8)	The Open routine was unable to get the resource the system requested for the data set being opened. The resource was being used by another task in the system. Detected by: IDA0192B
176 (B0)	The Open routine was unable to fix in real storage the access-method control blocks for the data set being opened. Detected by: IDA0192F
180 (B4)	The requested master or user catalog does not exist or is not open. Detected by: IDA0192C
184 (B8)	An I/O error occurred during I/O processing. Detected by: IDA0192B, IDA0200B, IDA0200T, IDA0231B, IDA0231T
188(BC)	The data set indicated by the ACB is not the type that may be specified by an ACB. Detected by: IDA0192Z, IDA0200B, IDA0231B
192 (C0)	An unusable data set was opened for output. Detected by: IDA0192B
196 (C4)	Access to data was requested by way of an empty path. Detected by: IDA0192B
200 (C8)	The Format-4 DSCB indicates the volume is unusable, so the data set cannot be opened. Detected by: IDA0192F
204 (CC)	The ACB MACRF specified GSR, but the program that issued OPEN isn't in supervisor state with protection key 0 or 7. Detected by: IDA0192A
212 (D4)	The ACB MACRF specified GSR or LSR, but the data set requires create processing. Detected by: IDA0192B
216 (D8)	The ACB MACRF specified GSR or LSR, but the key length of the data set exceeds the maximum key length specified in BLDVRP. Detected by: IDA0192B
220 (DC)	The ACB MACRF specified GSR or LSR, but the data set's control interval size exceeds the size of the largest buffer specified in BLDVRP. Detected by: IDA0192Z
224 (E0)	The ACB MACRF specified ICI, but the data set requires create processing. Detected by: IDA0192B
228 (E4)	The ACB MACRF specified GSR or LSR, but the VSAM shared resource table doesn't exist. Detected by: IDA0192A

ACBERFLG Code	Condition
232 (E8)	Reset was specified for a nonreusable data set, but the data set is empty. Detected by: IDA0192C
236 (EC)	A permanent staging error (ACQUIRE) or destaging error (RELINQUISH) occurred in the Mass Storage System. Detected by: IDA0192D
240 (F0)	Format-4 DSCB and catalog time-stamp verification failed during volume mounting for output processing. Detected by: IDA0192F
244 (F4)	The volume that contains the catalog recovery area wasn't mounted and verified for output processing. Detected by: IDA0192F

End-of-Volume Return Codes

These codes are returned by End of Volume to modules that call End of Volume. For Open and Close, those that indicate an error result in an ACBERFLG code's being returned to the user.

Reg 15	Condition
0 (0)	Successful.
4 (4)	The requested volume could not be mounted.
8 (8)	The requested amount of space could not be allocated.
12 (C)	I/O operations were in progress when End of Volume was requested.
16 (10)	The VS2 catalog could not be updated.

All End-of-Volume errors are detected by IDA0557A.

Control Block Manipulation Return Codes

When the Control Block Manipulation routine returns to the caller after successful completion, register 15 contains 0. If the request is GENCB, register 0 contains the total length of the area that contains the control block(s). Register 1 contains the address of the area.

When the Control Block Manipulation routine returns to the caller with a nonzero value in register 15, an error occurred. If the request is TESTCB and the caller supplied a ERET keyword, return is to the location specified by the ERET keyword. Otherwise, the Control Block Manipulation routine returns control to the point of invocation, via the return address in register 14.

Register 15 contains a return code:

Reg 15	Condition
0(0)	Successful completion.
4(4)	An error has been detected. The error code in register 0 indicates the type of error.
8(8)	Invalid use of the execute form of this macro. Since the return code is set by the macro expansion and not by the Control Block Manipulation routine, the register 0 contents do not indicate an error code.

Register 0 contains an error code:

Code	Applicable Macros*	Condition
1 (1)	G,M,S,T	The function type is invalid.
2 (2)	G,M,S,T	The control-block type is invalid.
3 (3)	G,M,S,T	The keyword type is invalid.
4 (4)	M,S,T	The control block to be processed isn't of the type specified.
5 (5)	S,T	The ACB to be processed is closed—it must be open.
6 (6)	S,T	The cluster whose index component was to be processed isn't key-sequenced (doesn't include an index).
7 (7)	M,S	The EXLST entry to be processed isn't present.
8 (8)	G	Not enough virtual storage is available, or (with AM=VTAM specified) list and execute forms are inconsistent.
9 (9)	G,S	User area is too small.
10 (A)	G,M	Exit address isn't specified in the input.
11 (B)	M	The RPL to be processed is active, or it is already being processed.
12 (C)	M	The ACB to be processed is open—it must be closed.
13 (D)	M	No exit address is specified in the input for the exit to be activated.
14 (E)	G,M,T	An invalid combination of option codes (for example, for MACRF or OPTCD) is specified.
15 (F)	G,S	The user area isn't on a fullword boundary.
16 (10)	G,M,S,T	A VTAM keyword is specified with AM=VTAM not specified.
19 (13)	M,S,T	A specified keyword refers to a field beyond the end of the control block to be processed.
20 (14)	S	A specified keyword requires processing with shared resources to be specified, but it isn't.
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

All errors in control block manipulation are detected by IDA019C1.

Virtual-Storage Management

The getting and freeing of storage for VSAM control blocks is managed centrally by IDA0192M. To allocate storage efficiently, IDA0192M (in most cases) gets storage in blocks large enough to satisfy not only a current request for storage for a control block, but also subsequent requests for storage for the same or a related control block. Figure 62 indicates:

- What control block(s) are stored in each type of storage block
- What block gives the address of each storage block
- What subpool each storage block is located in (subpools 234, 241, 245, and 252 are protected with key 0; subpool 250 is unprotected—attributes of system subpools are described in *OS/VS2 Scheduler and Supervisor Logic*)
- The size of each storage block (unused space in a block is freed after all required control blocks have been allocated)
- Whether each storage block is fixed in real storage by Open

To allocate and free storage in a storage block, IDA0192M uses these control blocks (which are described in detail in "Data Areas"):

- BIB—the base information block is built in subpool 252 upon a request to build it from the VSAM Open module IDA0192A. One BIB is built for all processing related to a particular base cluster in the job step.
- CMB—the cluster management block is built in subpool 252 upon the first request to open a particular cluster from the VSAM Open module IDA0192F. It enables IDA0192M to control the allocation and freeing of control blocks for the cluster. It contains the addresses of the header elements in header element blocks (described next) that identify the storage blocks that contain control blocks for the cluster.

  After a CMB has been built for a cluster, subsequent requests for storage for control blocks for the cluster (related to the same open) are satisfied, if possible, by using storage blocks already obtained. As storage blocks fill up, IDA0192M gets additional ones.
- HEB—the header element block is built (in the protected sphere block) by IDA0192M to manage the allocation and freeing of unprotected storage blocks. A HEB contains 16 header elements, each of which, when used, identifies and describes a storage block. The CMB indicates by the position of an entry that points to a header element what type of storage block the header element describes. The header element gives the block's address, length, subpool number, and available space. It doesn't give the address within the block of individual control blocks. These addresses are given by the control blocks within the VSAM control block structure, which is described in "Data Areas."

Storage Block	Contains	Pointed to by	Obtained in Subpool	Size	Fixed in Real Storage by Open?
Blocks Related to the Job Step as a Whole					
Sphere Block[1]	ACBs for the base cluster (path processing) and the alternate indexes in the upgrade set, RPLs for the alternate indexes in the upgrade set, UPT	BIB	250[2]	2K or larger	No
Protected Sphere AMBL Block[1]	AMBLs for the base cluster (path processing) and the alternate indexes in the upgrade set	BIB	252[2]	1K or larger	No
Protected Sphere Block	HEBs	BIB	241	1K or larger	No
Blocks Related to a Particular Cluster					
Buffer Block[3]	I/O buffers	CMB	250[2]	Length of the buffers requested	No[4]
Upgrade Buffer Block [1,3]	I/O buffers	CMB	250	Length of the buffers requested	No
DEB Block	DEB	CMB	230[5]	Length of the DEB	No[4]
EDB Block	EDB	CMB	252[2]	Length of the EDB	No[4]
String Block[3,6]	BUFCs, PLHs, RPLs for path PLHs, WAXs for path PLHs	CMB	250[2]	2K or larger	No[4]
Fixed String Block[3,6]	PFLs, IOSBs, SRBs, IQEs	CMB	245	Length of fixed string, plus VGTT header	No
Protected String Block[3,6]	IOMBs, CPA	CMB	252[2]	4K or larger	No[4]
Upgrade String Block[1,3]	BUFCs, PLHs	CMB	250	2K or larger	No
Fixed Upgrade String Block[1,3]	PFLs, IOSBs, SRBs, IQEs	CMB	245	Length of fixed string, plus VGTT header	No

Figure 62 (Part 1 of 2). Storage Blocks Used for Virtual-Storage Management

Storage Block	Contains	Pointed to by	Obtained in Subpool	Size	Fixed in Real Storage by Open?
Blocks Related to a Particular Cluster (continued)					
Protected Upgrade String Block[1,3]	IOMBs, CPA	CMB	252	2K or larger	No
User Block	AMBXN, AMDSBs, ARDBs, BUFC headers, preformat BUFCs, preformat CPAs, IWAs	CMB	250[2]	2K or larger	No[4]
Protected User Block	LPMBs, AMBs	CMB	252[2]	3 LPMBs, 2 AMBs, 64 bytes for set sector table	No[4]
Fixed Block[3]	IRB	CMB	254[7]	1 IRB	No

[1] This block doesn't exist for a catalog or a catalog recovery area built in system storage.

[2] Subpool is 231 for a catalog or a catalog recovery area built in system storage; subpool is 241 for processing with global shared resources (GSR).

[3] This block isn't built by Open for processing with shared resources—it's built by BLDVRP and resides in the resource pool.

[4] This block is fixed in real storage if requested by the user for improved control-interval processing (fast path).

[5] Subpool is 241 for a catalog, for a catalog recovery area built in system storage, or for processing with global shared resources (GSR).

[6] For certain processing, Close acquires this block and frees it after the processing is finished.

[7] Subpool is 245 for a catalog, for a catalog recovery area built in system storage, or for processing with global shared resources(GSR).

Figure 62 (Part 2 of 2). Storage Blocks Used for Virtual-Storage Management

Figure 63 gives the interrelationship of these control blocks. It shows two storage blocks obtained for DEBs. Storage blocks are obtained for other control blocks in the same way. A DEB block is just large enough to contain the DEB for which storage is requested. Some other storage blocks are large enough to contain several control blocks of the same or a related type, for which storage might be requested subsequently.

As a by-product, these control blocks map the location, by storage block, of VSAM control blocks for clusters (and associated paths and upgrade sets). BIBs, CMBs, and HEBs are in protected storage; they can be used to find a control block when a pointer in the VSAM control block structure has been destroyed or can't be found.

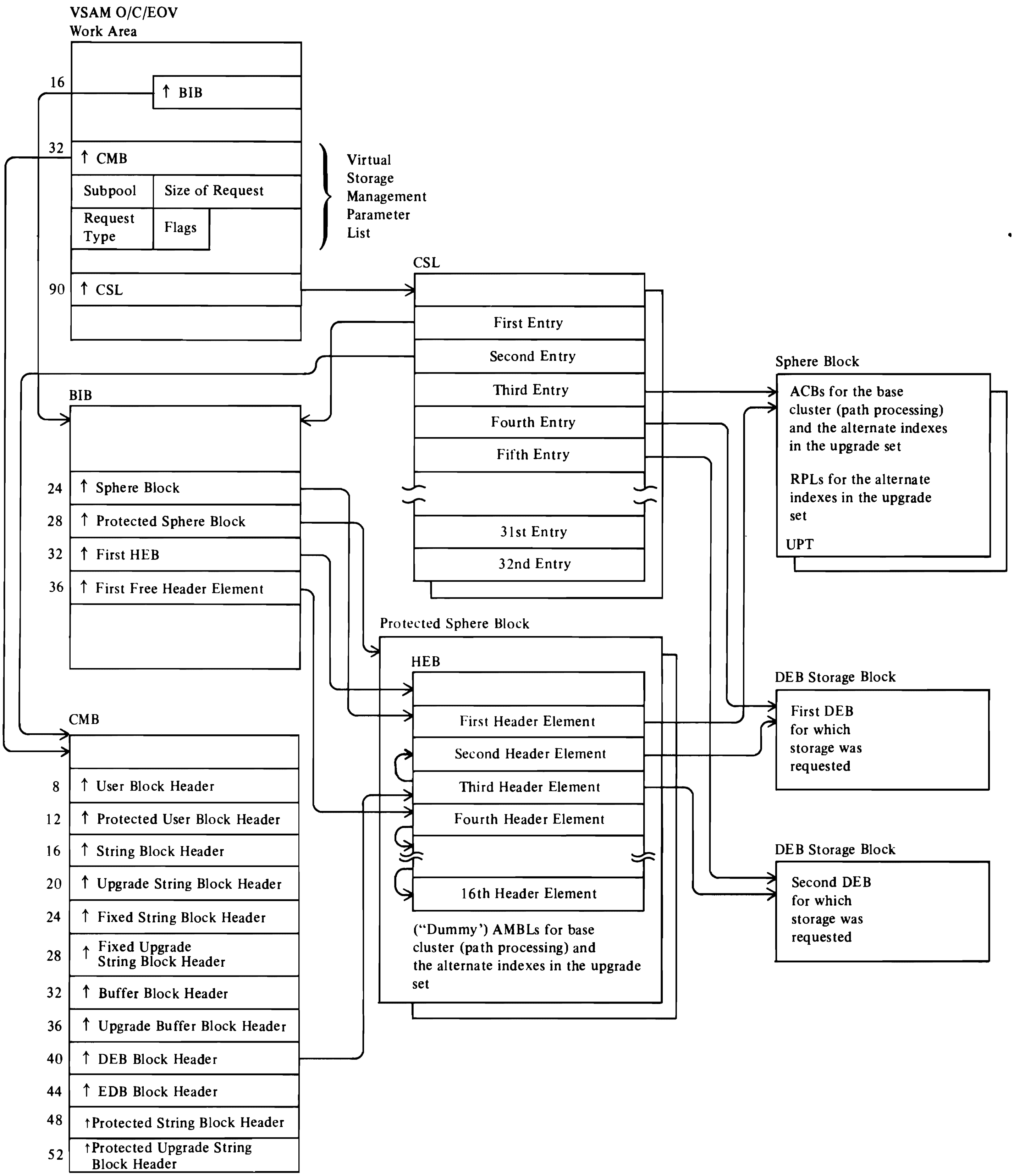

VSAM O/C/EOV
Work Area
16
↑ BIB
32
↑ CMB
Subpool
Size of Request
Request Type
Flags
Virtual Storage Management Parameter List
90
↑ CSL
CSL
First Entry
Second Entry
Third Entry
Fourth Entry
Fifth Entry
31st Entry
32nd Entry
Sphere Block
ACBs for the base cluster (path processing) and the alternate indexes in the upgrade set
RPLs for the alternate indexes in the upgrade set
UPT
BIB
24
↑ Sphere Block
28
↑ Protected Sphere Block
32
↑ First HEB
36
↑ First Free Header Element
Protected Sphere Block
HEB
First Header Element
Second Header Element
Third Header Element
Fourth Header Element
16th Header Element
("Dummy') AMBLs for base cluster (path processing) and the alternate indexes in the upgrade set
DEB Storage Block
First DEB for which storage was requested
DEB Storage Block
Second DEB for which storage was requested
CMB
8
↑ User Block Header
12
↑ Protected User Block Header
16
↑ String Block Header
20
↑ Upgrade String Block Header
24
↑ Fixed String Block Header
28
↑ Fixed Upgrade String Block Header
32
↑ Buffer Block Header
36
↑ Upgrade Buffer Block Header
40
↑ DEB Block Header
44
↑ EDB Block Header
48
↑Protected String Block Header
52
↑Protected Upgrade String Block Header

Figure 63. Virtual-Storage Management Control Block Structuree

Open, Close, and End-of-Volume Diagnostics

This section describes information in dumps from Open, Close, and End of Volume and how to obtain dumps additional to standard dumps.

Data-Set Management Recovery Routine (IDAOCEA1)

IDAOCEA1 gets control from the VS2 ESTAE routine for I/O Support Recovery when an error occurs while Open, Close, or End of Volume is processing.

IDAOCEA1 records in the field SDWARECP of the STAE diagnostic work area (SDWA, also known as the Recovery termination communication area—RTCA) the 8-character name of the failing module, its 8-character csect name, and the 8-character name of the recovery routine. SDWA is dumped into SYS1.LOGREC. The areas and modules identified here are dumped into SYS1.DUMP.

IDAOCEA1 uses the two words DXATEXC1 and DXATEXC2 in the Open/Close/End-of-Volume work area (referred to in program comments as "FORCORE").

The first byte in DXATEXC1 indicates the function in progress when the error occurred:

X'80'	IFG0192A, the interface between VS2 Open/Close/End of Volume and VSAM (this high-order bit is on in all cases)
X'40'	ISAM-Interface Close
X'20'	ISAM-Interface Open
X'10'	Temporary Close
X'08'	End of Volume
X'04'	Close
X'02'	Open
X'01'	BLDVRP or DLVRP (build or delete VSAM resource pool)

The second byte in DXATEXC1 is an indicator for checkpoint/restart processing:

1..	Checkpoint in progress
.1..	Restart in progress
..1.	Checkpoint/restart cleanup processing
...1	Recovery routine recursion indicator
.... xxxx	Reserved

The third byte is an additional option byte:

1...	Open/Close/End of Volume obtaining storage from CSA
.xxx xxxx	Reserved

The fourth byte is reserved.

DXATEXC2 contains the last four characters of the module in control (the first four characters are assumed to be IDA0).

ISAM-Interface Data-Set Management Recovery Routine (IDAICIA1)

IDAICIA1 gets control from IDAOCEA1 when an error occurs in ISAM-Interface Open or Close processing. To determine what to do, it uses audit flags set by IDA0192I or IDA0200S in the IIAUD fields in the IICB control block. The IICB is pointed to by DXATEXC2 in the Open/Close/End-of-Volume work area. (The format of the IIAUD information is given in "Data Areas.")

An error may have been caused by the ISAM Interface or by the user. For an ISAM-Interface error, IDAICIA1:

- Issues SDUMP to record information in SYS1.DUMP
- Issues SETRP to record the STAE diagnostic work area (SDWA) in SYS1.LOGREC
- Closes the associated DCB, which includes:
  Deleting routines
  Freeing storage
  Restoring the DCB
  Unchaining the DEB

The areas dumped by way of the SDUMP macro are indicated in the address list in SDUMPLST, which is associated with the macro. The addresses listed are those of:
The list
The user's DCB
The protected DCB (copied into the VS2 Open work area)
The Open/Close/End-of-Volume work area
The SDWA
The DEB (or 0)
The IICB (or 0)
The I/O buffers (or 0)
The physical-error message area (or 0)

For a user error, IDAICIA1 issues SETRP to record the SDWA in the SYSABEND data set. SDWA contains addresses of the areas to be dumped:
The user's DCB
The IICB
The AMDSB

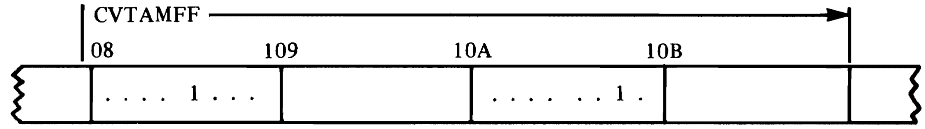
It also contains flags that indicate that this program data should be dumped:
Save areas
Registers
Program Status Word
User subpools

User errors include ABEND 03B from IDA0192I, ABEND 031 from IDAIIPM1 or IDAIISM1, and errors in a DCB exit routine. The next section, "ABENDs Issued by VSAM," discusses errors that result in ABENDs.

Getting a Dump of Open, Close, and End-of-Volume Work Areas

The messages that Problem Determination (IDA0192P) issues for Open, Close, and End of Volume may not be sufficient to determine what's wrong.

In such a case, you can get an ABEND dump by turning on a bit in the CVT (communication vector table) and rerunning the job in error. Use the CPU manual procedure AM (alter main storage) to set bit 4 of the first byte of CVTAMFF to '1' for VSAM to ABEND. Set bit 6 of the third byte of CVTAMFF to '1' to prevent the freeing of work areas.



After the error occurs, IDA0192P issues its message and also issues an ABEND with a user code of 888.

The contents of the registers (0-15) of the module that called IDA0192P (the same module identified by the function code in the problem-determination message) can be found at the address calculated by adding X'140' to the contents of register 4 at entry to ABEND.

The caller's register 13 contains the address of its standard register save area. The save areas of modules that had control before an abnormal termination are chained together, as shown in Figure 64.

Open, Close, and End-of-Volume modules (with the exception of IDA0192D, IDA0192G, IDA0192I, IDA0192M, and IDA0557A) use a main work area, OPWA (often called FORCORE), to communicate with one another. The field DXCCW7 contains the address of IFG0192A's save area, which contains the address of the next save area, and so on. Each save area after IFG0192A's contains the address of the previous save area. By following the back chain from the module that called IDA0192P, you can locate the save area of each module in Open, Close, or End of Volume that had control before the abnormal termination.

The save area is the first part of each module's work area (MWA, module work area, also called an ADA, automatic data area). It is used to store the contents of registers at entry to the *next* module that gets control. It is followed by a visual ID for easy recognition in the EBCDIC part of a dump listing. The visual ID contains the name of the module and the Julian date of its compilation.

For an Open or Close request (but not for End of Volume), the first work area obtained to process the request points to an OPWA for each ACB or DCB to be opened or closed. OPWA points to an area that contains a copy of the ACB or DCB.

During Open processing, register 4 contains the address of the open work area (OPW, mapped by IDAOPWRK and also called the ACB work area). OPW has the visual ID 'IDAOPWRK'.

During Close processing, register 4 contains the address of the close work area (CLW, mapped by IDACLWRK).

The open and close work areas are described in the "Data Areas" Section.

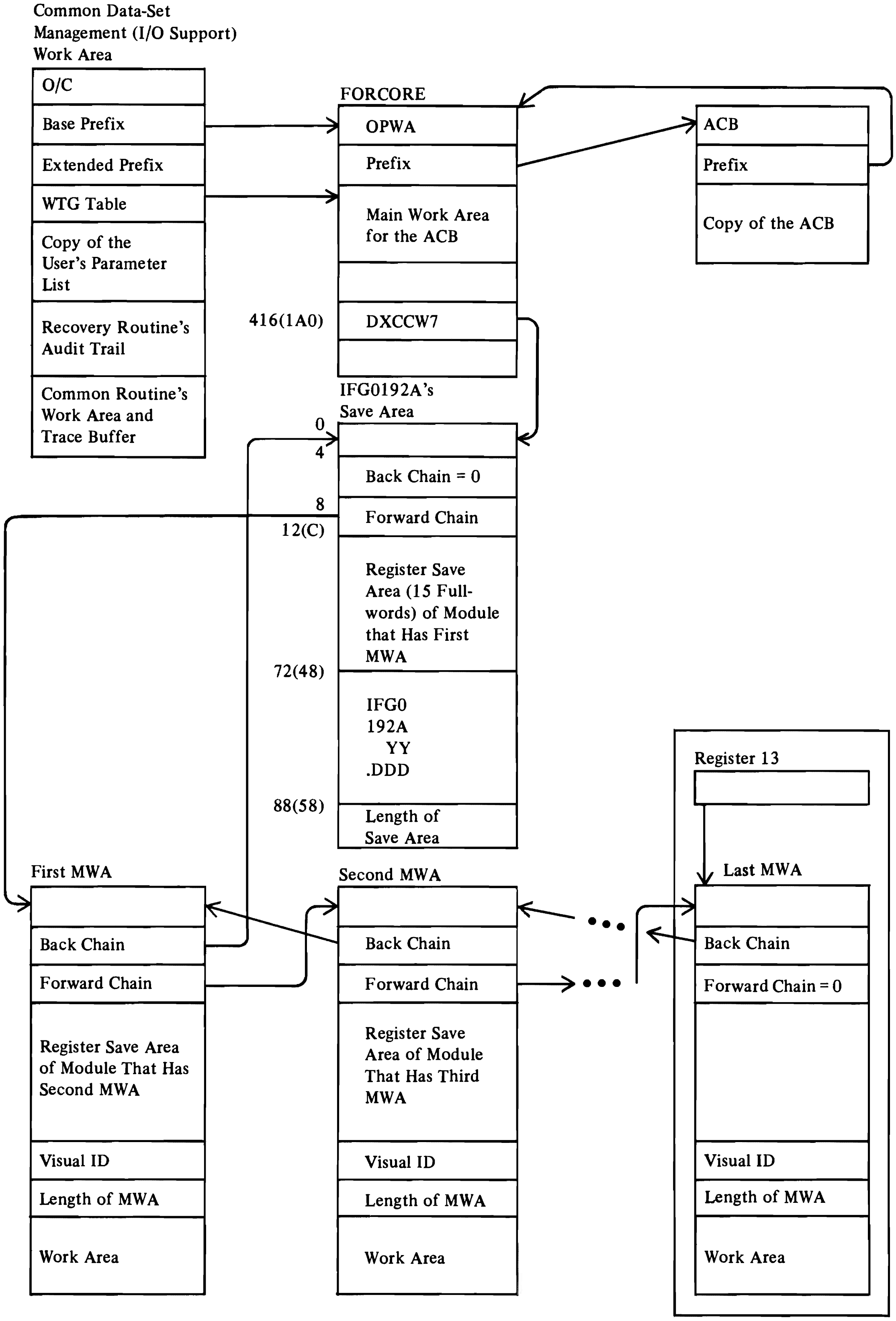


Figure 64. Chaining of Save Areas of O/C/EOV Modules

End of Volume uses its own module work area (MWA)—it doesn't have a special work area that corresponds to OPW or CLW. End of Volume's MWA has no visual ID. (Register 13 contains its address during End-of-Volume processing.)

Getting a Dump of VSAM Control Blocks in CSA

Control blocks and storage areas for data processed with the global shared resources (GSR) option are built in CSA, and the VSAM SNAP dump facility provides hexadecimal printouts of those control blocks and areas.

To get the dump, SDATA=CB must be specified in the SNAP macro (described in *OS/VS2 Sypervisor Services and Macro Instructions*) or must be specified to ABDUMP via the CHNGDUMP operator command or the IEAABD00 member of PARMLIB. Since the dump is actually made during SNAP processing, CSA is available anytime during job execution, not just during abnormal termination.

The VSAM SNAP formatting routine, IDA0195A, receives control from SNAP module IEAVAD08 (described in *OS/VS2 System Logic Library*). It locates, formats, and passes to the SNAP output routine five types of VSAM data in CSA:

- The JSCBSHR for the TCB being snapped. This field points to the valid-AMBL table (VAT) at the head of the VAT chain.
- The control blocks for any open VSAM GSR data set for the TCB being dumped.
- The control blocks that make up the GSR pool, if there were open GSR data sets or if the TCB being dumped is the jobstep TCB which issued the GSR BLDVRP macro.
- The VGTT chain for the ASCB associated with the TCB being dumped as well as any PSBss associated with these VGTTs.

The control blocks and storage areas made available by the dump facility are shown in Figure 65. On the actual printouts, each block of data is preceded by an identifying line that names the data (VSRT, for example) and gives its address and length. Output samples are shown in *OS/VS2 System Programming Library: Debugging Handbook*.

Entry and Exit

IDA0195A receives control in key 0 supervisor state with no locks held; ESTAE has been issued. Register contents upon entry are:

R1 - Address of IHAABDPL

R13 - Save area address

R14 - Return address

R15 - IDA0195A base address

Return in most cases will be to the caller, with registers 0 through 14 restored. Register 15 contains a return code: zero is normal return, nonzero is an error return. A nonzero return code causes VSAM formatting to stop and the message "VSAM CONTROL BLOCKS UNAVAILABLE" to appear in the dump.

Should an error occur that precludes the dumping of data, the message "IDA0195A DATA SUPPRESSED DUE TO ERROR" will appear in place of the data. Data will be suppressed because of machine checks and program interrupts (for example, address, page, segment, and protection exceptions).

Range Variables

To prevent endless looping through invalid chained data, IDA0195A has established range variables for loop detection and control. Control block chains will be followed until the range value is reached, and then the routine will force a logical end of chain and place the message "EXCESSIVE *XXXX* DETECTED BY IDA0195A" in the SNAP data set. *XXXX* describes the data being formatted when the suspension occurred. For example, "EXCESSIVE *GSR PSAB CHAIN* DETECTED BY IDA0195A".

The range variables (identified as DEBCNTMX, HEBCNTMX, VSAMCBMX, and MAXVCSLN) are grouped in the IDA0195A CSECT. Although the variables have been set high to allow for very large VSAM structures, they can be changed by using the Service Aid IMASPZAP ('SUPERZAP'). Modification requires a listing and possibly a dump of IDA0195A on the affected system. The variables are located in the following IDA0195A structure:

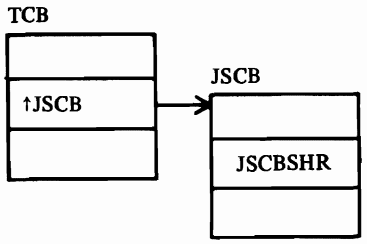
Offset	Variable	Value
0	Visual ID	EBCDIC: THRESHOLD VALUES
16	DEBCNTMX	Decimal 200
20	HEBCNTMX	Decimal 17
24	VSAMCBMX	Decimal 16
28	MAXVCSLN	Decimal 20

The following list identifies the range variable that influences the display of specific data or chains of data:

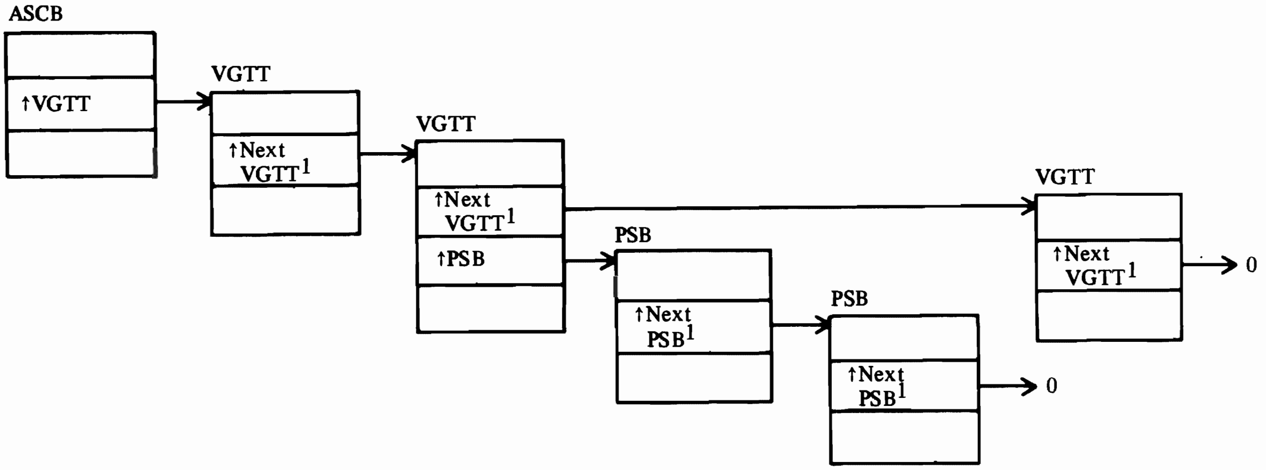
Data/Chain Name	Range Variable
TCB DEB Chain	DEBCNTMX
PSB chain of VGTT	VSAMCBMX
VGTT chain	VSAMCBMX
WSHD slot count	DIM (WSHDSLT)
CPA WSHD chain	VSAMCBMX
WSHD chain	VSAMCBMX
VSRT internal CSL list	MAXVCSLN
GSR VMT chain	VSAMCBMX
GSR HEB entries	HEBCNTMX
GSR PSB chain	VSAMCBMX
GSR PSAB chain	VSAMCBMX

Formatting of VSAM information will be suspended without an error message when one SNAP exceeds 256 CMBs or BIBs or when a HEB spans the PSB in which it resides. When the CMBs or BIBs exceed 256, that portion of formatting produces no output while other logic remains operative. Unlike the range variables, values for CMBs and BIBs cannot be modified. Increasing the BIB and CMB values will rdquire a recompilation, but this should not be necessary since these limits exceed the DEB chain variable in size.

JSCBSHR:



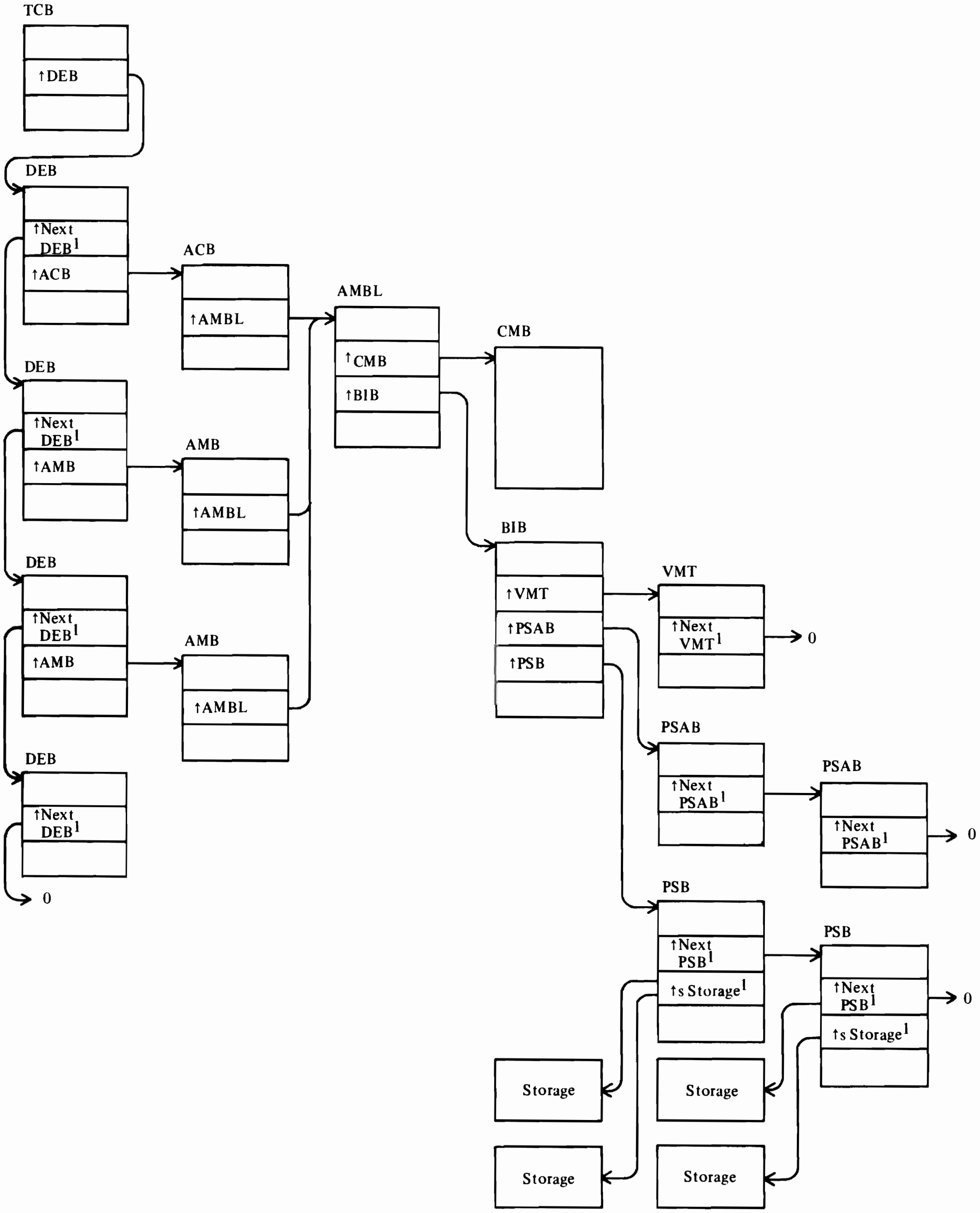
VGTT Chain and Associated PSBs:



[1] IDA0195A monitors the following of this chain, and processing may end before all control blocks in the chain are formatted. See "Range Variables."

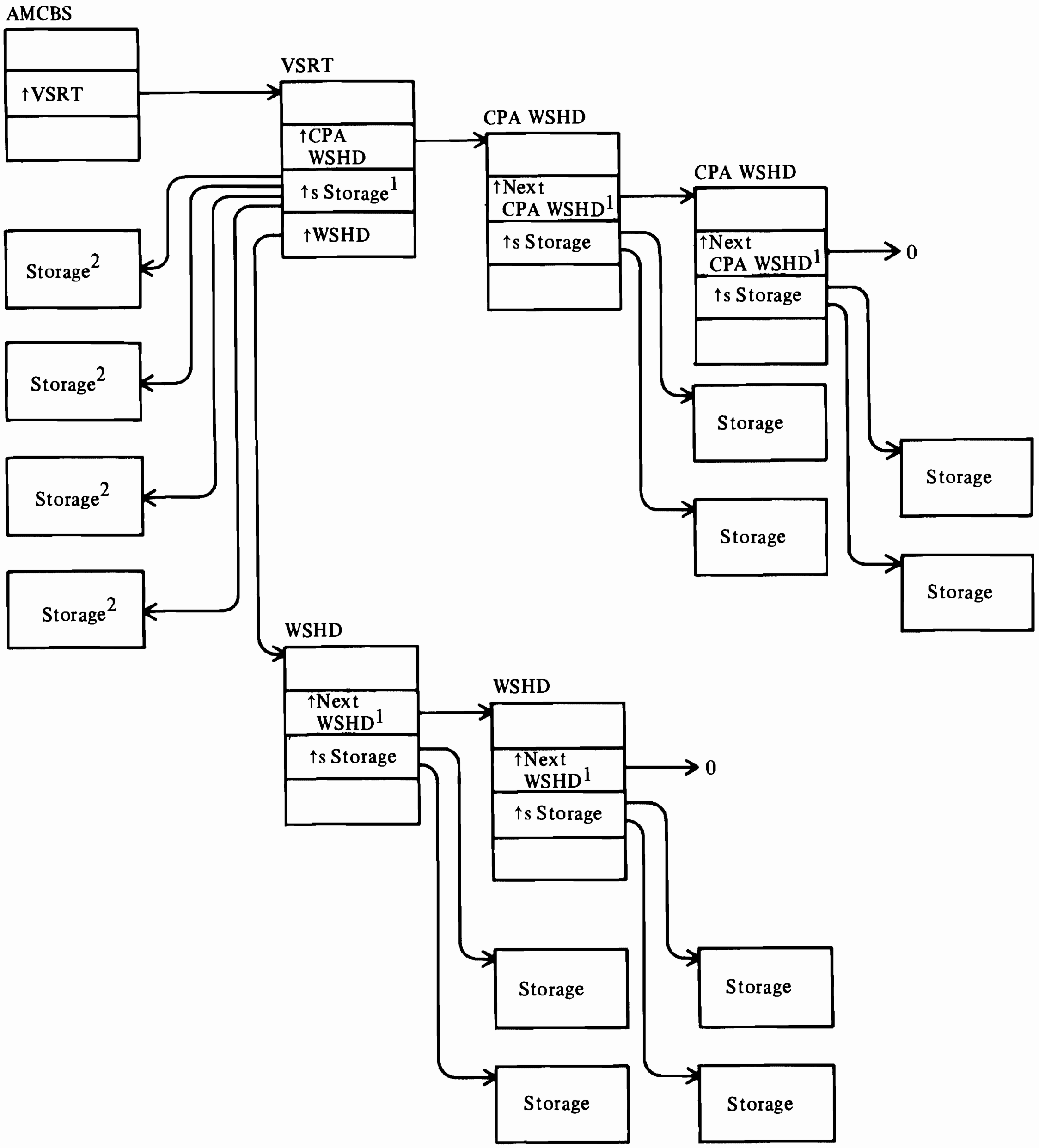
Figure 65 (Part 1 of 3). Control Blocks Made Available by the VSAM SNAP Dump Facility

Open VSAM Data Set with GSR Option:



[1] IDA0195A monitors the following of this chain, and processing may end before all control blocks in the chain are formatted. See "Range Variables."

Figure 65 (Part 2 of 3). Control Blocks Made Available by the VSAM SNAP Dump Facility



[1] IDA0195A monitors the following of this chain, and processing may end before all control blocks in the chain are formatted. See "Range Variables."

[2] This storage is located via the imbedded VSRT CSLs. The other storage is located via chains out of the VSRT or by slot entries in WSHDs (WSHDSLT).

Figure 65 (Part 3 of 3) Control Blocks Made Available by the VSAM SNAP Dump Facility

Recovery

Although covered by mainline SNAP's recovery routine, IDA0195A establishes its own ESTAE environment after gaining control. The ESTAE routine, RCVRRTN, returns control to mainline IDA0195A, which displays the message ‘IDA0195A DATA SUPPRESSED DUE TO ERROR” and concludes VSAM formatting with the message ‘END OF VSAM DATA”. No retry is done and percolation from RCVRRTN occurs when:

- No SDWA is available
- SDWACLUP=ON (clean-up entry)
- SDWANRBE=ON (error was not from this RB)
- Previous retry under this SNAP caused RCVRRTN to gain control (recursion)

RCVRRTN will attempt to record SDWA errors in SYS1.LOGREC, if retry is not successful.

Recovery with Global Shared Resources

When the user processes data with the GSR (global shared resources) option, a task in one address space issues the BLDVRP macro to build the VSAM resource pool in global storage. This address space is responsible for issuing the DLVRP macro to delete the resource pool. If the address space or the region control task terminates without issuing DLVRP, VS2 assumes responsibility for deleting the resource pool.

When the use count (count of data sets open) for the resource pool (in the AMCBS, which is described in the *OS/VS2 Catalog Management Logic*) drops to 0, VS2 issues the DLVRP macro.

When VS2 forces resource-pool deletion, it sends the standard problem-determination message, IEC251I, to the operator and to the output data set of the task that had been responsible for issuing DLVRP. The following return codes (rc) and function codes (ccc) indicate what happened:

rc 176 (B0)	Control blocks were dumped into the SYS1.DUMP data set.
rc 180 (B4)	Only some control blocks could be dumped into the SYS1.DUMP data set.
rc 184 (B8)	No control blocks could be dumped into the SYS1.DUMP data set.
ccc 148 (94)	VSAM Close module IDA0200T issued DLVRP.
ccc 149 (95)	VSAM Task Close Executor, IDAOCEA2, issued DLVRP.

In the SYS1.DUMP data set, the dumped control blocks are preceded by “IEC251I, VSAM GSR FORCE DLVRP DUMP DATA.” These control blocks are dumped:

- AMCBS
- VSRT
- WSHD and storage pointed to by it
- CPA WSHD and storage pointed to by it
- Control blocks and storage pointed to by the VSRT core save lists (CSLs)

For a forced deletion of the global resource pool, VSAM does *not:*

- Set the ACBERFLG return code
- Provide GTF tracing
- Provide a message in the ACB message area

ABENDs Issued by VSAM

The I/O Manager and the ISAM Interface issue ABENDs. The I/O manager stores the reason code for an ABEND in register 2 and stores all registers in the save area in the IOMB. Figures 65 and 66 list and explain each occurrence of an ABEND.

ABEND	Reason Code	Module	Explanation
377(179)	4(04)	IDAM19R3	Error return from issuance of SETLOCK macro
	8(08)	IGC121	Invalid AMB or IOMB
	12(0C)	IGC121	Invalid CPA
	16(10)	IGC121	Error in the VS2 PGFIX Routine (it returned a code other than 0 or 8)
	20(14)	IGC121 (PAGEOUT routine)	Invalid buffer address
	24(18)	IDA121A2	Error in converting to real address with LRA instruction
	28(1C)	IDA121A2	Block size not 4K (4096) for track overflow
633(279)	4(04)	IDA121A4 (IDA121F4 routine)	Invalid BUFC—the virtual storage originally assigned to the BUFC no longer belongs to the user
	20(14)	IDA121A4	Protection check indicated in the IOSB from the VS2 I/O Supervisor (invalid buffer address assumed to be the reason)

Figure 66. I/O-Management ABENDs

ABEND		Error	Error detected by	ABEND issued by	Error indication set in DCB or DECB by
1(001)		The user did not specify a SYNAD exit routine.			
	(a)	I/O error	VSAM initially and BISAM during CHECK	BISAM (IDAIIPM3)	SYNAD (DECB) (IDAIISM1)
	(b)	Invalid request	BISAM	BISAM	BISAM (DECB)
49(031)		The user did not specify a SYNAD exit routine.			
	(a)	VSAM physical or logical error	VSAM	SYNAD	SYNAD
	(b)	Invalid request	VSAM	SYNAD	GET and SETL routines of SCAN (IDAIIPM2)
	(c)	Sequence check	LOAD (IDAIIPM1)	LOAD	RESUME routine of LOAD
	(d)	Length error (RDW greater than LRECL)	LOAD	LOAD	LOAD
57(039)		End of data without EODAD routine	VSAM	SCAN (IDAIIPM2)	EODAD routine of SCAN
59(03B)		Validity check	OPEN (IDA0192I)	OPEN	Validity-check routine of OPEN

Catalog values and DCB values for LRECL, KEYLE, or RKP don't correspond, or, with QISAM, DISP is specified OLD when the data set is being opened for output, and there are already records in the data set (implying RELOAD).

Figure 67. ISAM-Interface ABENDs

Exception codes may be set in the DCB (for QISAM processing) or the DECB (for BISAM processing) in connection with ISAM-Interface ABENDs. Figures 68 and 69 give the exception codes. Except where indicated, register 15 contains 8, for logical errors.

DECB exception code	Explanation	Corresponding RPL feedback code(s)	Explanation	Error detected By
DECBEXC1				
1...	Record not found	16(10)	Record not found	VSAM
		24(18)	Record on unmountable volume	VSAM
.1..	Record-length check	108(6C)	Record-length check	VSAM
..1.	Space not found	28(1C)	Data set not extendable	VSAM
...1	Invalid request	none	No RPL available	ISAM Interface
		20(14)	Exclusive-control conflict	VSAM
		36(24)	No key range defined for insertion	VSAM
		64(40)	Placeholder not available	VSAM
		96(60)	Key-change attempted	VSAM
.... 1...	Uncorrectable I/O error	4-24 (04-18)	A physical error (Register 15 contains 12(0C))	VSAM
.... .1..	Unreachable block	—	A logical error not covered by another exception code	VSAM
.... ..1.	Overflow record (indicated for all READ requests)	none		ISAM Interface
....1	Duplicate record	8(08)	Duplicate record	VSAM
DECBEXC2				
xxxx xx..	Reserved (always 0)	none		
.... ..1.	Channel program initiated by an asynchronous routine (never indicated, always 0)	none		
....1	Previous macro was READ KU	none		ISAM Interface

Figure 68. BISAM Exception Codes in Relation to VSAM Return Codes

DECB exception code	Explanation	Corresponding RPL feedback code(s)	Explanation	Error detected by
DCBEXCD1				
1...	Record not found	none	Record not found (SETL K for deleted record)	ISAM Interface
		16(10)	Record not found	VSAM
		24(18)	Record on unmountable volume	VSAM
.1..	Invalid device address (never indicated, always 0)	none		
..1.	Space not found	28(1C)	Data set not extendable	VSAM
		40(28)	Virtual storage not available	VSAM
...1	Invalid request	none	Two consecutive SETL requests; invalid SETL (I or ID); or invalid generic key (KEY=0)	ISAM Interface
		4(04)	Request issued after reaching end of data	VSAM
		20(14)	Exclusive-control conflict	VSAM
		36(24)	No key range defined for insertion	VSAM
		64(40)	Placeholder not available	VSAM
		96(60)	Key-change attempted	VSAM
.... 1...	Uncorrectable input error	4(04)	Read error in data set	VSAM
		8(08)	Read error in index set	VSAM
		12(0C)	Read error in sequence set (Register 15 contains 12(0C))	VSAM
.... .1..	Uncorrectable output error	16(10)	Write error in data set	VSAM
		20(14)	Write error in index set	VSAM
		24(18)	Write error in sequence set (Register 15 contains 12(0C))	VSAM
.... ..1.	Unreachable block (input)	—	A logical error covered by another exception code	VSAM
....1	Unreachable block (output)	—	A logical error not covered by another exception code	VSAM

Figure 69 (Part 1 of 2). QISAM Exception Codes in Relation to VSAM Return Codes

DECB exception code	Explanation	Corresponding RPL feedback code(s)	Explanation	Error detected by
DCBEXCD2				
1...	Sequence check	none	Sequence check (during resume load only)	ISAM Interface
		12(0C)	Sequence check	VSAM
.1..	Duplicate record	8(08)	Duplicate record	VSAM
..1.	DCB closed when error routine entered	none	Error in Close	VSAM
...1	Overflow record (always indicated)	none		ISAM Interface
.... 1...	Length of logical record is greater than DCBLRECL (VLR only)	none	Length of logical record is greater than DCBLRECL (VLR only)	ISAM Interface
		108(6C)	Invalid record length	VSAM
.... .xxx	Reserved (always 0)			

Figure 69 (Part 2 of 2). QISAM Exception Codes in Relation to VSAM Return Codes

GLOSSARY

Acronyms and Abbreviations

Following is an alphabetized list of the acronyms and abbreviations used in this book and in the VSAM code listings. If you do not find the term you are looking for, refer to the index or to the *IBM Data Processing Glossary,* GC20-1699.

ABEND	abnormal end
ABP	Actual Block Processor (either the IOM module IDA121A2 or the IOM communication vector table)
ACB	access method control block
ADDR	addressed processing or addressed
ADR	same as ADDR
AIX	alternate index
AMB	access method block
AMBL	access method block list
AMBXN	access method block extension
AMDSB	access method data statistics block
AMS	Access Method Services
ARDB	address range definition block
ASCB	address-space control block
BIB	base information block
BISAM	Basic ISAM
BLPRM	resource pool parameter list
BSPH	buffer subpool header
BUFC	buffer control block
BWD	backward (processing)
C	Close
C/R	Checkpoint/Restart
CA	control area
CCB	command control block
CHKPT	checkpoint
CI	control interval
CIDF	control interval definition field
CLW	close work area (mapped by IDACLWRK)
CMB	cluster management block
CNV	control interval or control-interval processing
core	virtual storage
CPA	channel program area
CRA	catalog recovery area
CSA	common service area
CSL	core save list
CVT	communication vector table
DCB	data control block
DDNAME	data definition name
DEB	data extent block
DIR	direct processing
DIWA	data insert work area
DSCB	data set control block
DSL	DEB save list
DSNAME	data set name
DSORG	data set organization
ECB	event control block
EDB	extent definition block
ENDREQ	end the request
EOD	end of data
EOF	end of file
EOV	End of Volume
EP	external procedure entry point
ERFLG	error flags
ESL	enqueue save list
EXCD	exceptional conditions
EXCP	execute channel program
EXLST	exit list
Ext Proc	external procedure
FKS	full key search
FS	free space
FWD	forward (processing)
GC	type code (group code)
GEN	generic key search
GSR	global shared resources
HEB	header element block
ICIP	improved control-interval processing
ICWA	index create work area
ID	identifier
IDAL	indirect data-address list (real page list)
II	ISAM Interface
IICB	ISAM interface control block
IMWA	index modification work area
Int Proc	internal procedure
I/O	input/output
IOB	input/output block
IOM	I/O Management
IOMB	I/O-Management block
IOMBXN	I/O-Management block extension
IOSB	I/O-Supervisor block
ISAM	Indexed Sequential Access Method
JFCB	job file control block
JSCB	job step control block
JSTCB	job step task control block
KEQ	search on key equal
KEY	keyed accessing
KGE	search on key greater or equal
L	link
LLOR	least length of record (that contains all key fields)
LPMB	logical-to-physical mapping block
LSR	local shared resources
MACR	macro reference
MOD	module
MSS	Mass Storage System
MSVI	mass storage volume inventory
MWA	module work area
n	integer number
NSI	next sequential instruction
NSP	next string position
NUP	no update
O	Open
O/C/EOV	Open/Close/End of Volume
OFLG	open flags
OPTCD	option code
OPW	open work area (mapped by IDAOPWRK)
OPWA	common O/C/EOV base work area

OPWRK	VSAM O/C/EOV ACB work area (mapped by IDAOPWRK)
OS/VS	Operating System/Virtual Storage
PFL	page fix list
PFPL	PGFIX parameter list (same as PFL)
PIOD	Problem-State I/O Driver
PL/I	programming language/one
PLH	placeholder list
PROC	procedure
PSB	protected sphere block
PSL	page save list
PSR	Programming Systems Representative
PSW	program status word
QISAM	Queued ISAM
RAB	record area block
RBA	relative byte address
RDF	record definition field
RM	Record Management
Rn	general-purpose register n
RPL	request parameter list
RPLE	request parameter list extension
RPS	rotational position sensing
RRDS	relative record data set
RTM	recovery/termination manager
RTN	routine
SCIB	search compressed index block
SCRA	catalog recovery area in system storage
SDWA	system diagnostic work area
SEQ	sequential or sequential processing
SIOD	Supervisor-State I/O Driver
SKP	skip sequential or skip sequential processing
SMF	system management facilities
SRB	service request block
SSCR	Subsystem checkpoint record
SSL	swap save list
SST	set sector table
STRNO	number of RPL strings
SVC	supervisor call
TCB	task control block
TIOT	task I/O table
TSO	time sharing option
UCB	unit control block
UCRA	catalog recovery area in user's storage
UPD	update mode (or data modify)
UPT	upgrade table
USAR	user security-authorization record
USVR	user security-verification routine
VAT	valid-AMBL table
VCRCORE	VSAM checkpoint/restart core
VCRT	VSAM checkpoint/restart table
VGTT	VSAM global termination table
VIOT	valid-IOMB table
VMT	volume mount table
VPL	virtual page list
VRP	VSAM resource pool
VS	(Operating System/) Virtual Storage
VSAM	Virtual Storage Access Method
VSL	virtual subarea list (same as PFL or PFPL)
VSRT	VSAM shared resource table
VTOC	volume table of contents
VVIC	(replaced by MSVI)
WAX	work area for path processing
WSHD	working storage header
WTG	where-to-go table
XCTL	transfer control (macro)
XPT	checkpoint
XREF	cross reference

Definitions of Terms Used In This Book

Access Method Services: A multifunction service program that defines VSAM data sets and allocates space for them, converts indexed sequential data sets to key-sequenced data sets with indexes, modifies data-set attributes in the catalog, reorganizes data sets, facilitates data portability between operating systems, creates backup copies of data sets and indexes, helps make inaccessible data sets accessible, and lists data-set records and catalog entries.

addressed direct access: The retrieval or storage of a data record identified by its relative byte address, 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 access: 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.

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 cluster over which one or more alternate indexes are built.

candidate volume: A direct-access storage volume that has been defined in a VSAM catalog as a VSAM volume; VSAM can automatically allocate space on this volume, as needed.

catalog: (*See* master catalog and user catalog.)

catalog recovery area: (*See* CRA.)

CIDF: Control interval definition field. The 4-byte control-information field at the end of a control interval that gives the displacement from the beginning of the control interval to free space and the length of the free space. If the length is 0, the displacement is to the beginning of the control information.

cluster: A combination of related VSAM data sets, identified by one name in a VSAM catalog and requiring a single DD statement. A key-sequenced data set and its index form a cluster; an entry-sequenced data set alone forms a 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.

compendium: A compendium gathers together and presents in concise form all the essential facts and details about a VSAM functional unit.

component: As used in this book, a group of modules that perform a function, such as I/O Management.

compression: (*See* key compression.)

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 and distributes free space. It is the unit of information transmitted to or from auxiliary storage by VSAM, some integer multiple of blocksize.

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 volume itself. The CRA contains self-describing records as well as 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 and not from the standpoint of the manner in which it is stored (see also stored record).

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 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 relative byte addresses cannot change. Records are retrieved and stored by addressed access, and new records are added at the end of the data set.

extent: A continuous space allocated on a direct-access storage volume, reserved for a particular data space or data set.

external procedure: A procedure that can be called by any other VSAM procedure; a procedure whose name is in the module's (assembler listing) "external symbol dictionary".

field: In a record or a control block, a specified area used for a particular category of data or control information.

free space: (*See* distributed free space.)

generic key: A high-order portion of a key, containing characters that identify those records that are significant for a certain application. For example, it might be desirable to retrieve all records whose keys begin with the generic key AB, regardless of the full key values.

global storage: Virtual storage that is not part of a user's private address space.

GSR: global shared resources. (*See* shared resources.)

horizontal extension: An extension record pointed to by a catalog record's extension field. (*See also* vertical extension.)

horizontal pointer: A pointer in an index record that gives the location of another index record in the same level that contains the next key in collating sequence; used for keyed sequential access.

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; organized in levels of index records. (*See also* index level, index set, and sequence set.)

index entry: A key and a pointer paired together, where the key is the highest key (in compressed form) entered in an index record or contained in a data record in a control interval, and the pointer gives the location of that index record or control interval.

index level: A set of index records that order and give the location of records in the next lower level or (sequence set record) that give the location of control intervals in the control area that it is associated with.

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.

index set: The set of index levels above the sequence set. The index set and the sequence set together comprise the index.

index upgrade: The process of reflecting changes made to a base cluster in its associated alternate indexes.

integrity: (*See* data integrity.)

internal procedure: A procedure that can be called only by other procedures within the module. (*See also* external procedure.)

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 with an index.

key: One or more characters within an item of data that are used to identify it or control its use. As used in this publication, one or more consecutive characters taken from a data record, used to identify the record and establish its order with respect to other records. (*See also* key field and generic key.)

key compression: The elimination of characters from the front and the back of a key that VSAM does not need to distinguish the key from the preceding or following key in an index record; reduces storage space for an index.

key field: A field located in the same position in each record of a data set, whose contents are used for the key of a record.

key sequence: The collating sequence of data records, determined by the value of the key field in each of the data records. May be the same as, or different from, the entry sequence of the records.

key-sequenced data set: A data set whose records are loaded in key sequence and controlled by an index. Records are retrieved and stored by keyed access or by addressed access, and new records are inserted in the data set in key sequence by means of distributed free space. Relative byte addresses of records can change.

keyed direct access: The retrieval or storage of a data record by use of an index that relates the record's key to its relative location in the data set, 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 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.)

local storage: Virtual storage in a user's private address space.

LSR: local shared resources. (*See* shared resources.)

mass sequential insertion: A technique VSAM uses for keyed sequential insertion of two or more records in sequence into a collating position in a data set: more efficient than inserting each record directly.

mass storage volume: Two data cartridges in the IBM 3850 Mass Storage System that contain information equivalent to what could be stored on a direct-access storage volume.

master catalog: A key-sequenced data set with an index containing 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.

memory: As used in this book, a synonym for the private address space in virtual storage.

module: The unit of code that is link-edited. A program module has at least one procedure, and may have many.

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: On a track of a direct-access storage device, the space between interrecord gaps.

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. (*See also* horizontal pointer and vertical pointer.)

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 having one or more alternate indexes. (*See also* index and alternate index.)

prime key: The key of reference for a key-sequenced data set when it was loaded. (*See also* key.)

procedure: A functional unit of VSAM code that is entered only at one entry point and exits at the end of the procedure (the last line of the procedure's code). The procedure can call (transfer control, with a return to the procedure expected) other procedures within the module (internal calls) and can call other procedures in other VSAM modules (external calls). (*See also* internal procedure and external procedure.)

random access: (*See* direct access.)

RBA: Relative byte address. The displacement of a data record or a control interval from the beginning of the data set to which it belongs; independent of the manner in which the data set is stored.

RDF: Record definition field. A 3-byte control-information field to the left of the CIDF in a control interval that gives the length of a record in the control interval or the number of consecutive records having the same length.

record: (*See* index record, data record, stored record.)

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 in a relative record data set but also the record occupying the slot.

replication: (*See* index replication.)

reusable data set: A VSAM data set that can be reused as a work file, regardless of its old contents.

security: (*See* data security.)

segment: The portion of a spanned record contained within a control interval. (*See also* spanned record.)

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 or its key sequence, relative to the previously retrieved or stored record. (*See also* addressed sequential access and keyed sequential access.)

shared resources: 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. Resources are shared either locally (LSR) or globally (GSR).

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.

spanned record: A 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.

sphere: The collection of base cluster, alternate indexes, and upgrade alternate indexes opened to process one or more paths related to the same Base Information Block (BIB).

stored record: A data record, together with its control information, as stored in auxiliary storage.

string: The part of a control block structure built around a placeholder (PLH) that enables VSAM to keep track of one position in the data set that the control block structure describes.

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: A catalog used in the same way as the master catalog, but optional and pointed to by the master catalog, and also used to lessen the contention for the master catalog and to facilitate volume portability.

vertical extension: An extension record pointed to by a set-of-fields pointer in the object's base catalog record or its horizontal extension. (*See also* base catalog record and horizontal extension.)

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 *OS/VS2 Master Index of Logic,* GY28-0694.

SY26-3825-1

OS/VS2 Virtual Storage Access Method (VSAM) Logic (File No. S370-30) Printed in U.S.A. SY26-3825-1

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

OS/VS2 Virtual Storage Access Method (VSAM) Logic
SY26-3825-1

Reader's Comment Form

Your comments about this publication will help us to improve it for you. Comment in the space below, giving specific page and paragraph references whenever possible. All comments become the property of IBM.

Please do not use this form to ask technical questions about IBM systems and programs or to request copies of publications. Rather, direct such questions or requests to your local IBM representative.

If you would like a reply, please provide your name, job title, and business address (including ZIP code).

Fold on two lines, staple, 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.

Fold and Staple

First Class Permit
Number 439
Palo Alto, California

Business Reply Mail

No postage necessary if mailed in the U.S.A.

Postage will be paid by:

IBM Corporation
General Products Division
Programming Publishing—Department J57
1501 California Avenue
Palo Alto, California 94304

Fold and Staple

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

OS/VS2 Virtual Storage Access Method (VSAM) Logic (File No. S370-30) Printed in U.S.A. SY26-3825-1