SY26-3857-0
File No. S370-30

Systems

# OS/VS2 SVS Independent Component: Virtual Storage Access Method (VSAM) Logic

**Release 1.7**

**Feature Numbers 5083**
**5084**
**5472**
**5473**

IBM

**First Edition (January 1977)**

This edition applies to Release 1.7 of OS/VS2 and to any subsequent releases of that system unless otherwise indicated in new editions or technical newsletters.

The Feature Numbers that apply for OS/VS2 SVS (Program Number 5742-017) are:

| Number | Meaning |
|---|---|
| 5083 | Basic material; 1600 bpi, 9-track tape |
| 5084 | Basic material; 6250 bpi, 9-track tape |
| 5472 | Optional material; 1600 bpi, 9-track tape |
| 5473 | Optional material; 6250 bpi, 9-track tape |

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, P.O. Box 50020, Programming Publishing, San Jose, California 95150. 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, index, and catalog 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 used in this book and in the VSAM program listings.
- "Index," which is a subject index to the book.

## Required Reading

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

- *OS/VS2 SVS Independent Component: Virtual Storage Access Method (VSAM) Programmer's Guide,* GC26-3868, which introduces VSAM concepts and contains definitive explanations of VSAM macro instructions.
- *OS/VS2 SVS Independent Component: Access Method Services,* GC26-3867, which describes the catalog record processing commands: DEFINE, ALTER, DELETE, LISTCAT, and CONVERTV.

## Related IBM Publications

- *Introduction to the IBM 3850 Storage System (MSS),* GA32-0028
- *OS/VS Mass Storage System (MSS) Planning Guide,* GC35-0011
- *OS/VS Mass Storage System (MSS) Services: General Information,* GC35-0016
- *OS/VS Mass Storage System (MSS) Services: General Reference,* GC35-0017
- *OS/VS DADSAM Logic,* SQ66-3787

- *OS/VS Data Management Macro Instructions,* GT00-0132
- *OS/VS Catalog Management Logic,* ST00-0181
- *OS/VS JCL Reference,* GT28-0618
- *OS/VS JCL Services,* GT00-0141
- *OS/VS Message Library: VS2 System Messages,* GT38-1002
- *OS/VS Open/Close/EOV Logic,* ST00-0138
- *OS/VS Service Aids,* GT28-0633
- *OS/VS Supervisor Services and Macro Instructions,* GT27-6979
- *OS/VS System Management Facilities (SMF),* GT00-0134
- *OS/VS1 VSAM Cross Reference,* SYB6-3844
- *OS/VS2 Checkpoint/Restart Logic,* SQ66-3820
- *OS/VS2 Data Areas,* ST68-0606
- *OS/VS2 Debugging Guide,* GT28-0632
- *OS/VS2 I/O Supervisor Logic,* SQ66-3823
- *OS/VS2 Supervisor Logic,* SY27-7244
- *OS/VS2 SVS Independent Component: Access method Services,* GC26-3867
- *OS/VS2 SVS Independent Component: (VSAM) Options for Advanced Applications,* GC26-3870
- *OS/VS2 SVS Independent Component: Virtual Storage Access Method (VSAM) Programmer's Guide,* GC26-3868

## Using This Book

This book is designed to be used with the VSAM program listings in the microfiche for VSAM and with *OS/VS1 VSAM Cross Reference,* SYB6-3844, also on microfiche cards. Cross-reference reports are described in "Microfiche 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, "Program Organization."

# CONTENTS

# ILLUSTRATIONS

## Figures

## Diagrams

# SUMMARY OF ENHANCEMENTS

## *New Data Areas*

The following work areas have been added:

- Resource pool parameter list (BLPRM) is used by Record Management for dynamic string addition and by data set management for internal processing
- Close work area (CLW) is used for communication among the Close and temporary Close modules
- Open work area is used by the VSAM Open modules
- Keyword table (KEYWDTAB) is a branch table that controls execution of module IDA0191C and supports processing for the control block manipulation macros

## *Technical Changes*

Diagrams CA and CB have been changed to support improved control block manipulation macro processing.

A new control block manipulation return code is issued when a block to be displayed or tested does not exist because the data set is a dummy data set.

## *Editorial Changes*

Each CAXWA in the CAXWA chain contains a pointer to the CRA ACB. That pointer and the associated control block structure were added to Figure 72, Open Catalog Control Blocks.

Flowcharts that describe Catalog Management I/O functions have been given figure numbers 51.1 through 51.19. The new figure numbers are in the List of Figures following the Table of Contents, and they are referred to from the module directory.

VSAM has several new functions and data structures for this release of SVS; alternate indexes, spanned records, reusable data sets, relative record data set, processing the index of a key-sequenced data set, shared resources among data sets, improved control-interval processing, backward sequential processing, catalog recovery, data staging for the IBM 3850 Mass Storage System, 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.

HIPOs and compendiums have been added for Record Management to document Buffer Management and I/O Management.

### *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 Open and Close. Access by way of a path and alternate-index upgrading change and add HIPOs to Record Management. Defining and deleting alternate indexes, paths, and upgrade sets add HIPOs for the DEFINE and DELETE functions of Catalog Management, add types of catalog records, and change many catalog 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.

### *Reusable Data Sets*

The catalog record processing required when a VSAM data set or alternate index is reused adds a HIPO to Catalog Management.

### *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 Open/Close/End of Volume for the BLDVRP and DLVRP macros. 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 many HIPOs and adds a HIPO to Catalog Management. Catalog recovery changes the format of the catalog record header, adds types of catalog records, and adds field to various catalog control blocks.

### *Data Staging for the IBM 3850 Mass Storage System*

Staging and destaging of data between mass storage and direct-access storage for the IBM 3850 Mass Storage System changes HIPOs for Open/Close/End of Volume and adds a HIPO to Catalog Management.

### *Virtual-Storage Management*

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

# INTRODUCTION

Virtual Storage Access Method (VSAM) is an access method for use with OS/VS2 SVS. VSAM is used with direct-access storage devices 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. A data set's records can be ordered according to when the records are stored, where the records are stored, or what values are in each record's key field. With key-sequenced data sets, the user can access a record by specifying its key or its relative byte address (RBA). With entry-sequenced data sets, the user can access a record only by specifying its RBA. With relative record data sets, the user can access a record by specifying its relative record number. For additional information on VSAM records and how they are stored, see "Data Areas."

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

VSAM resides in the pageable link pack area along with the user's processing program. Figure 1 illustrates VSAM's relationship to OS/VS, the processing program, and the data stored on a direct-access storage device and in mass storage.

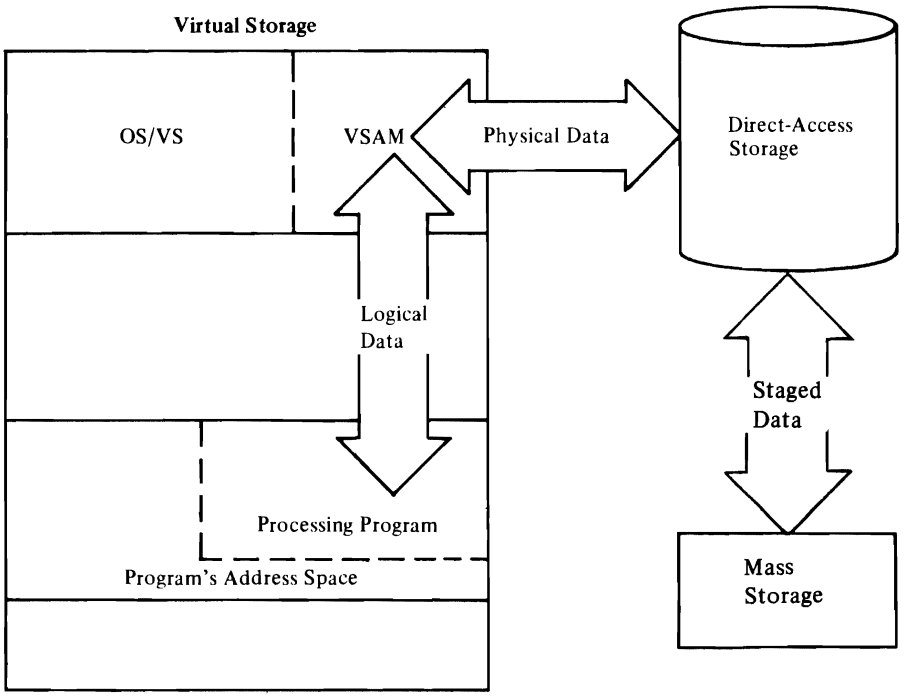



Figure 1. Relationship of VSAM, OS/VS, User's Processing Program, and Staged Data

VSAM is controlled by user macro instructions. For additional information on user macro instructions, see *OS/VS2 SVS Independent Component: Virtual Storage Access Method (VSAM) Programmer's Guide* and *OS/VS2 SVS Virtual Storage Access Method (VSAM) Options for Advanced Applications.*

VSAM catalogs can contain entries for data sets that are stored on a mass storage volume with the IBM 3850 Mass Storage System, which is described in *Introduction to the IBM 3850 Mass Storage System (MSS).*

VSAM communicates with other parts of the operating system through the SVC processor and through OS/VS control blocks used by VSAM. In addition to the OS/VS 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 direct-access device space management, that is, VSAM makes only minor use of OS/VS Open and Close and relies on OS/VS DADSM for only part of its direct-access device space management. To do much of this work, VSAM uses its own catalogs. VSAM catalogs contain a description of VSAM direct-access device space: where available space is, how space is used, and the location of data sets. For additional information on the VSAM catalog, see "Catalog" in "Data Areas" and *OS/VS Virtual Storage Access Method (VSAM) Programmer's Guide.*

VSAM modules are logically grouped into the following components:

- Open, which 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.
- Close, which 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 VSAM catalog.
- End of Volume, which mounts volumes and allocates space. End of volume modifies the existing control blocks to reflect the newly mounted volumes and newly allocated space.
- Record management, which reads and writes records in response to user-issued VSAM and ISAM macro instructions. This component also reads and writes records for the catalog management component and causes volumes to be mounted and demounted when it detects end-of-volume.
- ISAM Interface, which allows user programs that contain ISAM macro instructions to process VSAM data sets. The ISAM Interface routines translate a user's ISAM macro instructions into appropriate VSAM macro instructions and control blocks. The ISAM Interface routines next issue the VSAM macro instruction to read or write the user's VSAM record. When the VSAM read/write operation completes, the ISAM Interface routine interprets the VSAM record management return codes and translates them into appropriate ISAM return-code information for the user's program.

- Catalog management, which writes and updates catalog records. Catalog management processes the catalog to obtain and store information for Open, Close, End-of-Volume, and Access Method Services.
- Control block manipulation, which allows the user program to create, modify, display, and test the contents of some VSAM control blocks (the ACB, EXLST, and RPL, which are described under "Data Areas" in this publication).

# METHOD OF OPERATION

A method of operation diagram describes one of the VSAM functions by listing the process steps required to complete the function, and by showing the data required for each process step and the data produced by each process step.

## *Reading Method of Operation Diagrams*

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

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.

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

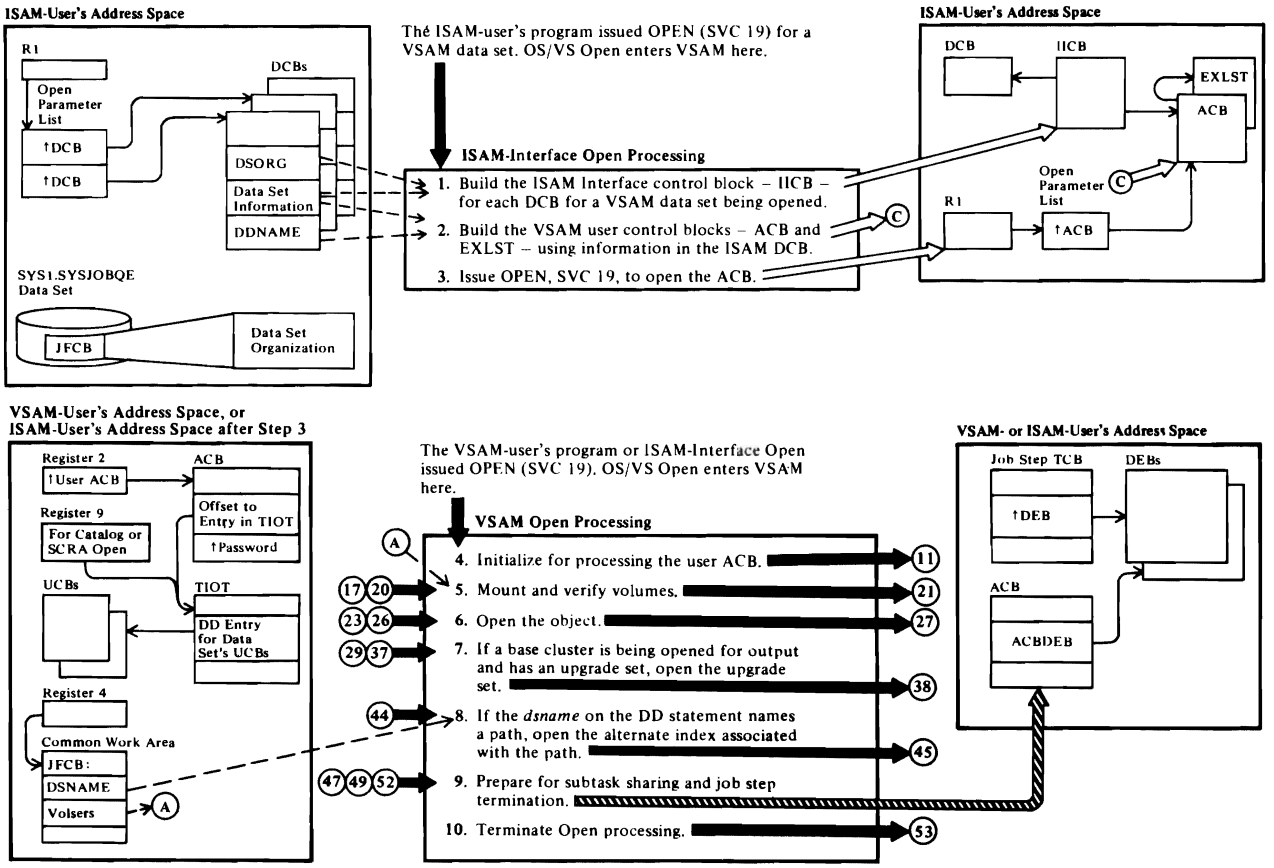



Figure 2. Method of Operation Diagram

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 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.

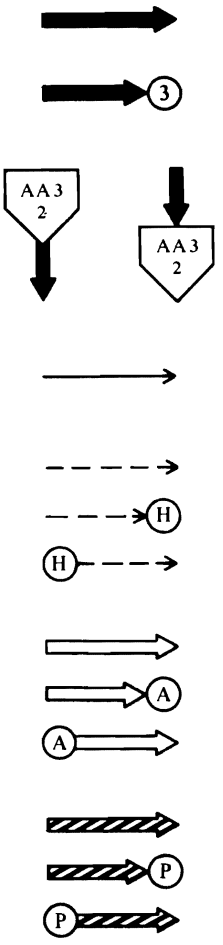



Figure 3. Graphic Symbols Used in Method of Operations Diagrams

Figure 4 shows part of the notes to Figure 2.

**Notes for Diagram AC1**

**OS/VS Open: Initial Processing:**

When the caller issues the OPEN macro instruction, SVC 19, IGC00011 (OS/VS Open) is entered by the OS/VS SVC Interruption handler.

OS/VS Open reads the JFCB from the SYS1.SYSJOBQE data set.

**ISAM Interface Open Processing:**

If the JFCB data-set organization (JFCDSORG) field indicates a VSAM data organization and the DCB data-set organization (DCBDSORG) indicates indexed sequential organization, IFG0196V (OS/VS Open) sets the identifier for each DCB-for-VSAM-data-organization entry in the WTG table to '21', the identifier of the ISAM-Interface Open routine.

**1 IDA01921: 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 VSAM records.

See "Data Areas" for details about the IICB.

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

**2 IDA01921: 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 IDA01921: OPENACB**

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

When VSAM Open processing completes (the ACB built in step 2 is open), ISAM Interface Open processing continues at step 58 (see Diagram AC7).

**VSAM Open Processing:**

If the open-parameter-list entry addresses a VSAM ACB, OS/VS 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 OS/VS Open processing is bypassed for each ACB entry until the VSAM Open routine returns control to OS/VS Open at step 57

**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

**Note:** 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, IOB, 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

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.

# Diagram AA1. Method of Operation Contents

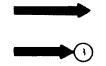
**Macro Instructions:**

| | |
|---|---|
| OPEN, CLOSE, CLOSE (TYPE=T), BLDVRP, DLVRP, CHKPT | VSAM Open, Close, and End-of Volumn, and VSAM Checkpoint/Restart — Diagrams AC1-AI1 |
| GET, PUT, POINT, CHECK, ENDREQ, ERASE, GETIX, PUTIX, MRKBRF, SCHBFR, WRTBFR | Record Management — Diagrams BA1-BT1 |
| GENCB, MODCB, SHOWCB, TESTCB | VSAM Control Block Manipulation — Diagrams CA1-CB2 |
| CATLG (From VSAM) | VSAM Catalog Management — Diagrams DA1-DM3 |
| CATLG (From AMS) | VSAM Catalog Management Services — Diagrams EA1-EM1 |

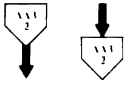
**ISAM Macro Instructions:**

| | |
|---|---|
| OPEN | ISAM Interface Open — Diagrams AC1, AC7 |
| CLOSE | ISAM Interface Close — Diagrams AD1, AD6 |
| GET, PUT, PUTX, SETL, ESETL, RELSE, READ, WRITE, CHECK, FREEDBUF | ISAM Interface Processing — Diagrams BU1-BU2 |

**LEGEND**



Flow of control on the same page: '3' indicates a number of a process step on the same page.



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



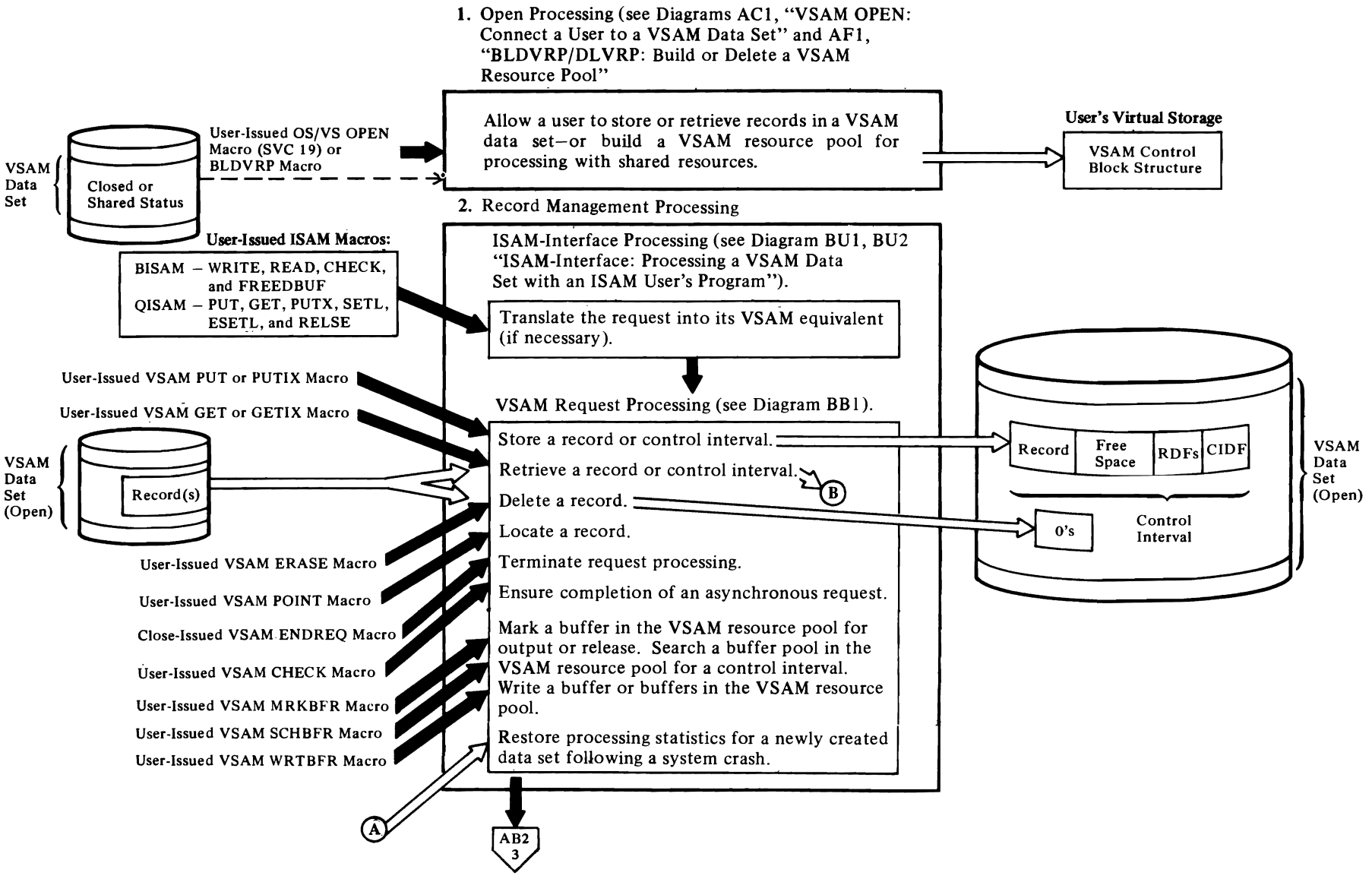
Pointers

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

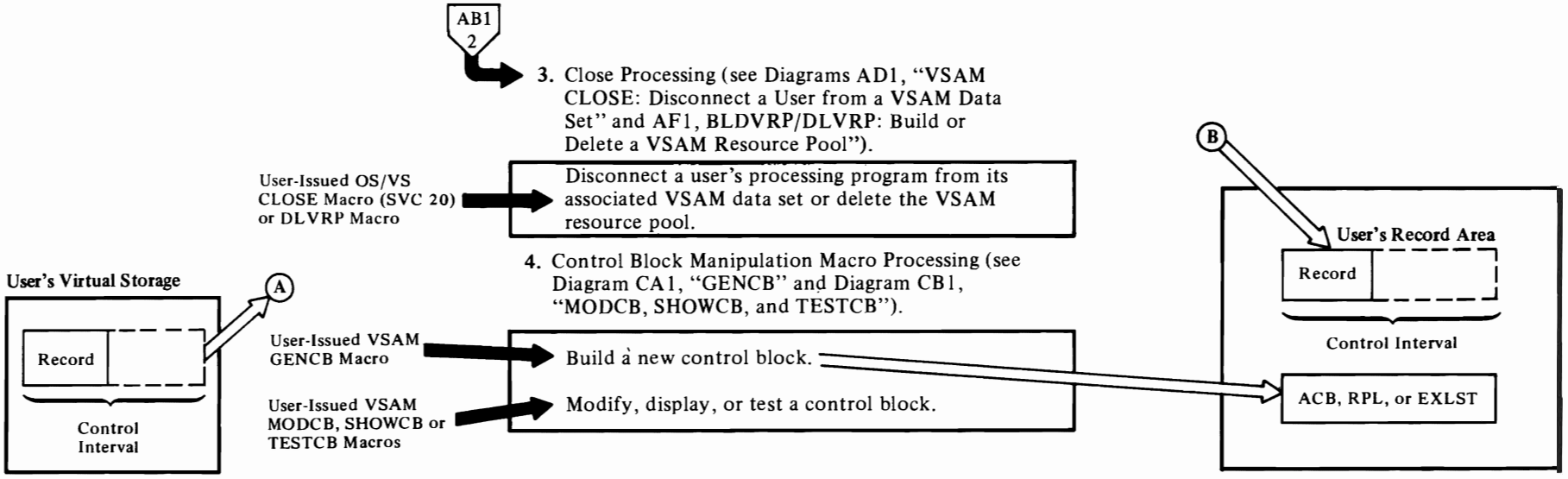
Input to process steps and output from process steps; 'A' is an arbitrary designation.

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

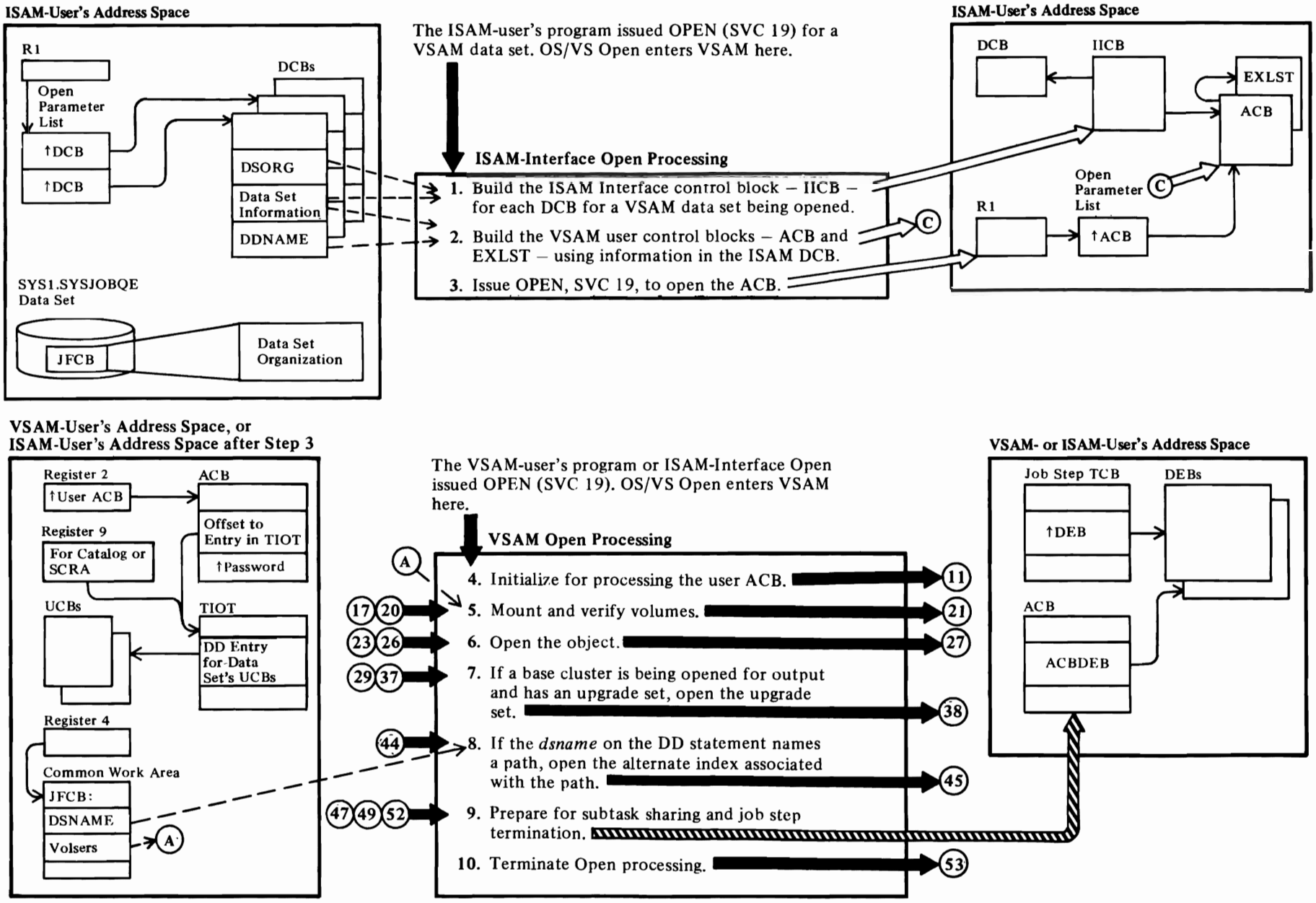
# Diagram AB1. VSAM Overview



## Diagram AB2. VSAM Overview



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



## Notes for Diagram AC1

**OS/VS Open: Initial Processing:**

When the caller issues the OPEN macro instruction, SVC 19, IGC0001I (OS/VS Open) is entered by the OS/VS SVC Interruption handler.

OS/VS Open reads the JFCB from the SYS1.SYSJOBQE data set.

**ISAM Interface Open Processing:**

If the JFCB data-set organization (JFCDSORG) field indicates a VSAM data organization and the DCB data-set organization (DCBDSORG) indicates indexed sequential organization, IFG0196V (OS/VS 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 VSAM 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.

When VSAM Open processing completes (the ACB built in step 2 is open), ISAM Interface Open processing continues at step 58 (see Diagram AC7).

**VSAM Open Processing:**

If the open-parameter-list entry addresses a VSAM ACB, OS/VS 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 OS/VS Open processing is bypassed for each ACB entry until the VSAM Open routine returns control to OS/VS Open at step 57.

**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.

**Note:** Dynamic String Addition

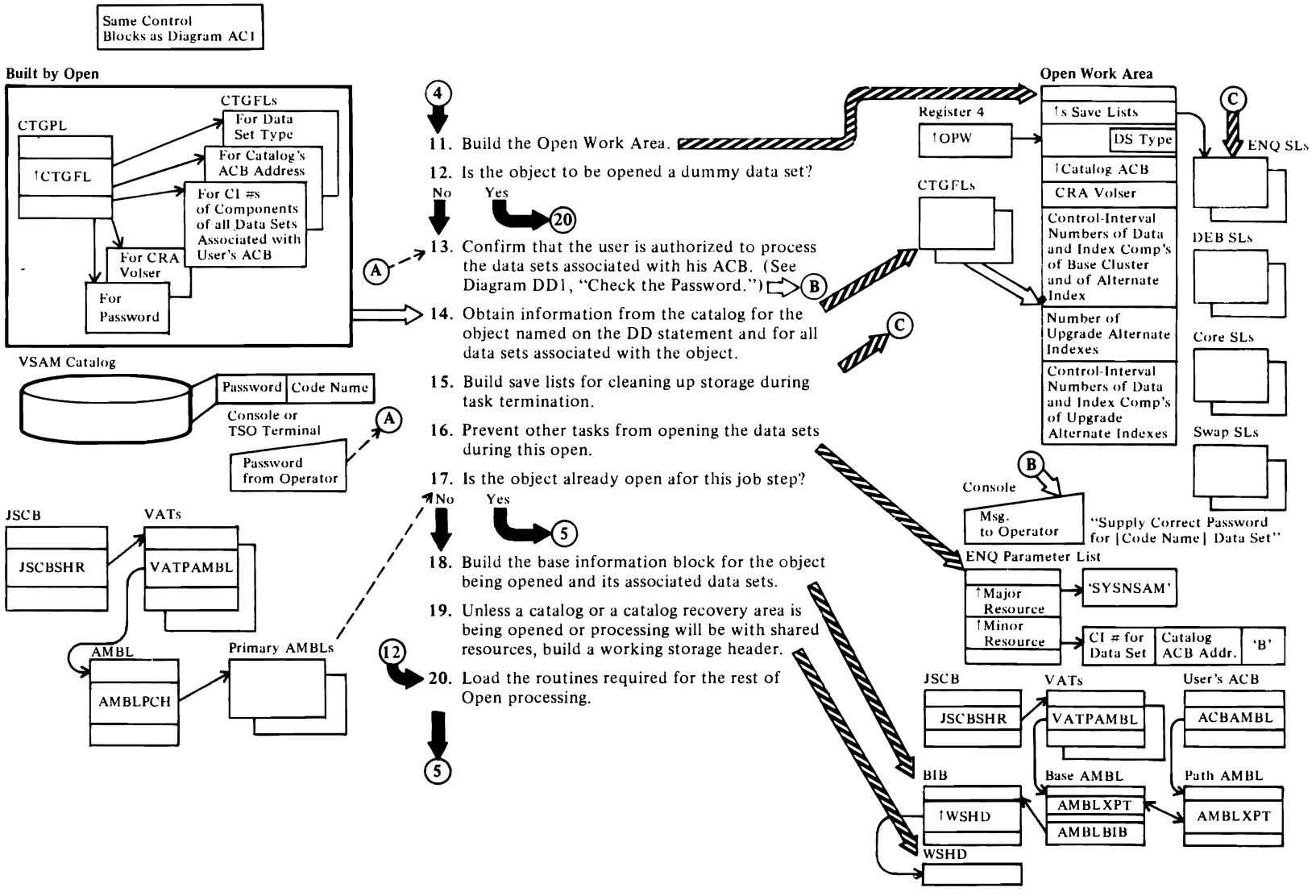
When OPEN is issued, not to open a data set, but to dynamically add a string to the user's capability to process multiple requests concurrently, the string is added and Open returns to the caller. VSAM Record Management requests dynamic string addition when more strings are required than the user specified.

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, IOB, 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



## Notes for Diagram AC2

**11 IDA0192A: INIT192A**

The open work area is mapped by the IDAOPWRK macro.

**13 IDA0192C**

The user establishes the numberof times the operator may attempt to supply the correct password, as described in *OS/VS2 SVS Independent Component: 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 IDA1092A: BLDLISTS**

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

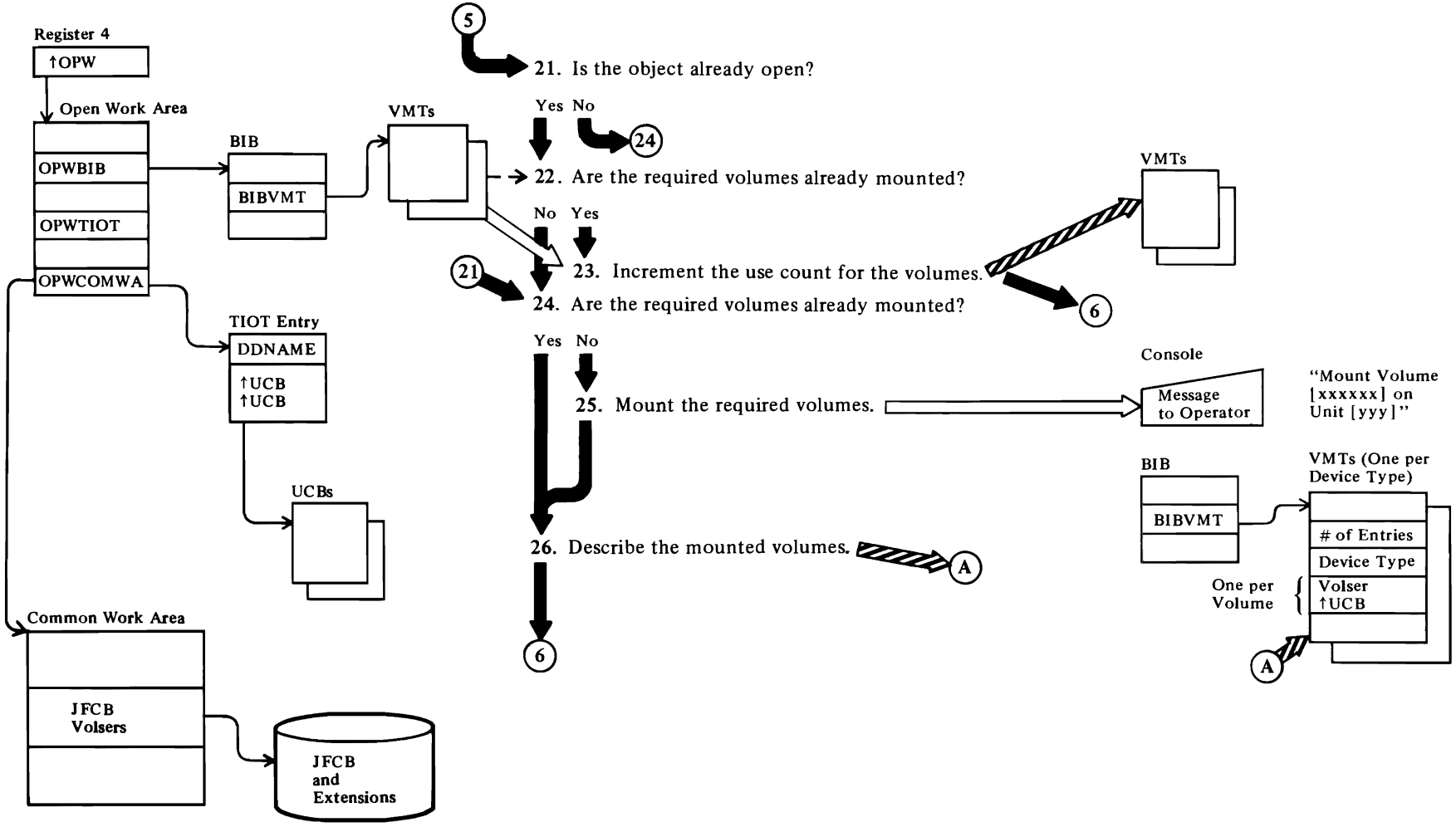
**16 IDA0192A: BLDENQPL, INIT192A**

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.

# Diagram AC3. VSAM OPEN: Mount and Verify Volumes



## Notes for Diagram AC3

**21 IDA0192F: VOLMNT**

**22 IDA0192F: OLDDEV**

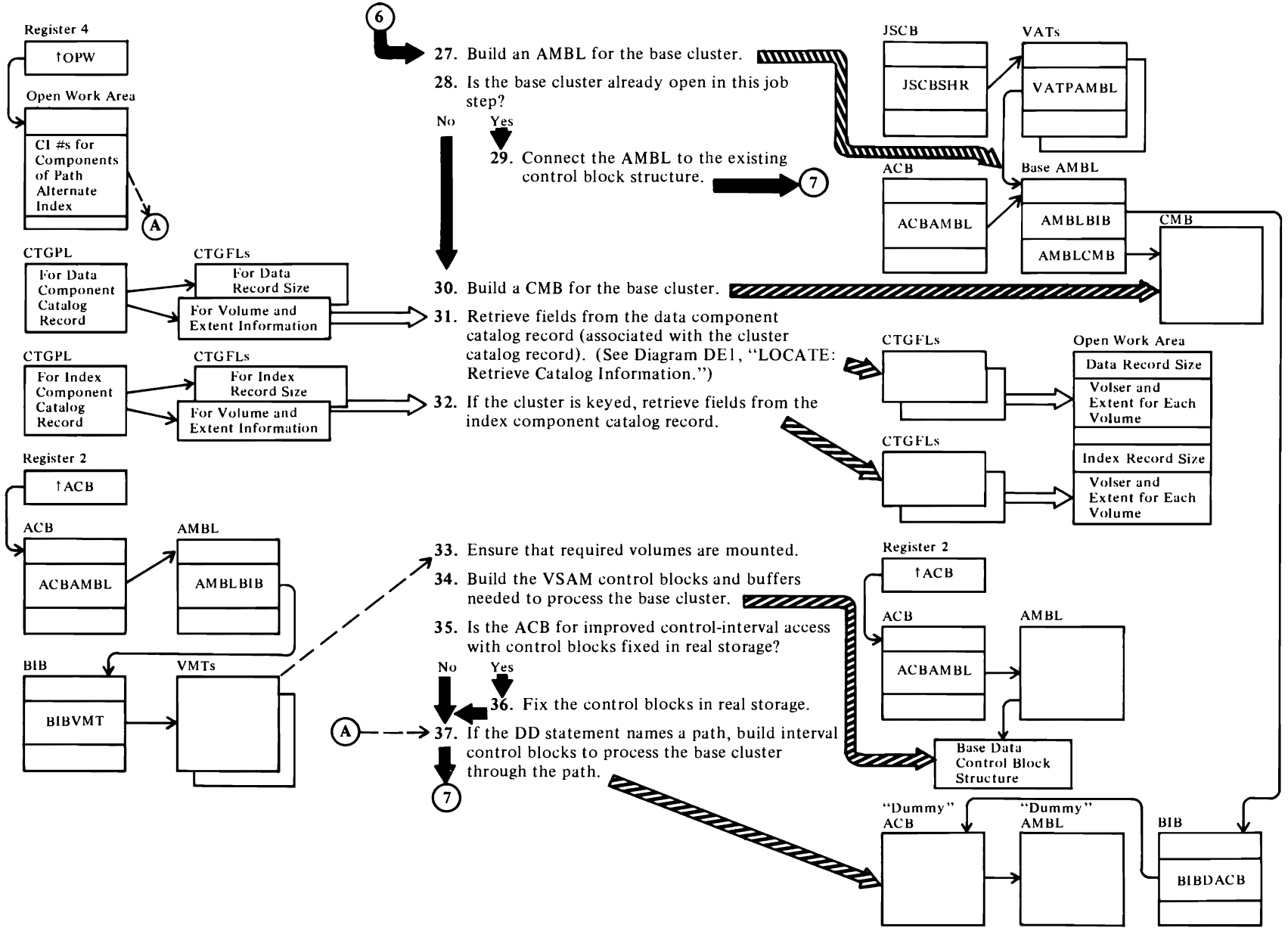
**24 IDA0192V**

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

**26 IDA0192V: 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.

# Diagram AC4. VSAM OPEN: Open the Base Cluster



## Notes for Diagram AC4

**27 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.

**28 IDA0192F: CHNAMBL**

**29 IDA0192F: CHNAMBL**

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

**30 IDA0192F: BLDCMB**

**31 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 "Data Areas" for details about the data set catalog record, the CTGPL, and the CTGFL.

**32 IDA0192B, IDA0192C: LOC2, LOC3**

The index catalog record is pointed to by the cluster catalog record. See "Data Areas" for details about the index catalog record.

**33 IDA0192B**

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

**34 IDA0192Z, 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
- Local Shared Resources Control Block Structure
- AMB Control Block Structure with Local Shared Resources

"Data Areas" also describes each VSAM control block.

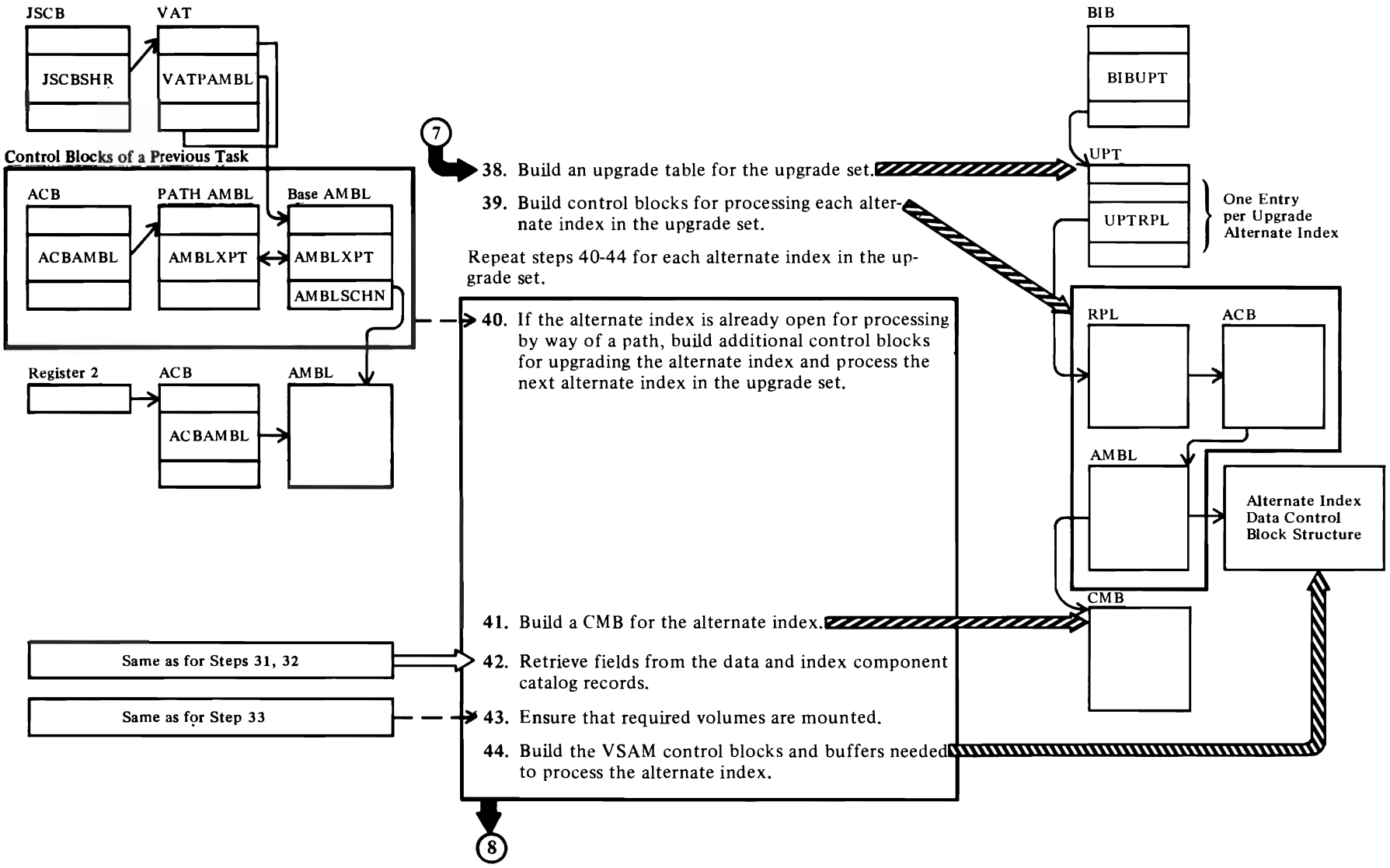
**36 IDA0192F: OPNBASE, PAGEFIX**

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.

**37 IDA0192F: OPNBASE**

# Diagram AC5. VSAM OPEN: Open the Upgrade Set



## Notes for Diagram AC5

**38 IDA0192F: OPNUPGR**

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

**39 IDA0192F: OPNUPGR, BLDAMBL**

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

**40 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, IOB, CPA, and buffers. These control blocks are described in "Data Areas."

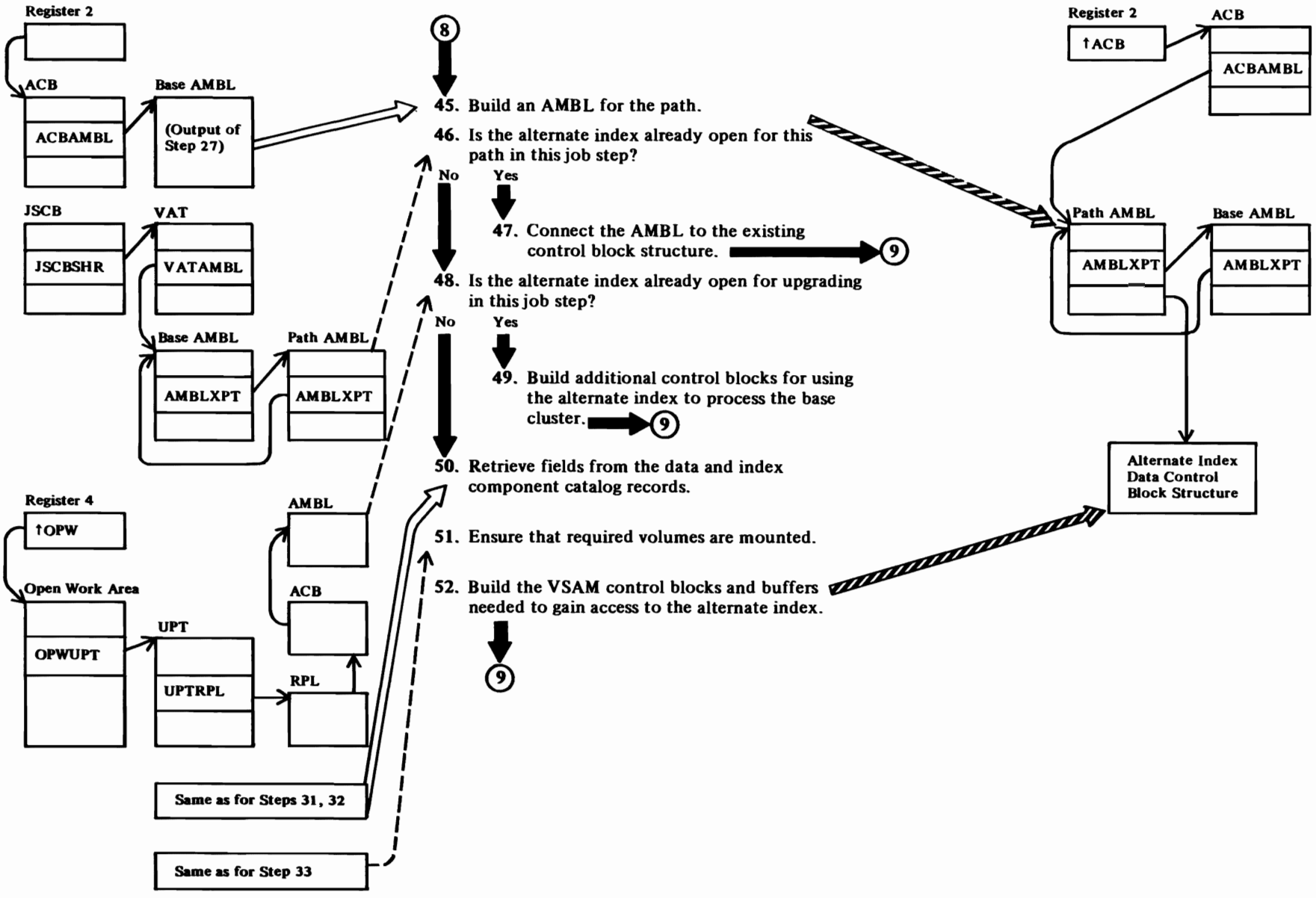
**41 IDA0192F: BLDCMB**

**42** See notes for steps 31 and 32.

**43** See note for step 33.

**44** See note for step 34.

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



## Notes for Diagram AC6

**45 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.

**46 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.

**47 IDA0192F: OPNPATH**

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

**48 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.

**49 IDA0192F: CONPATH**

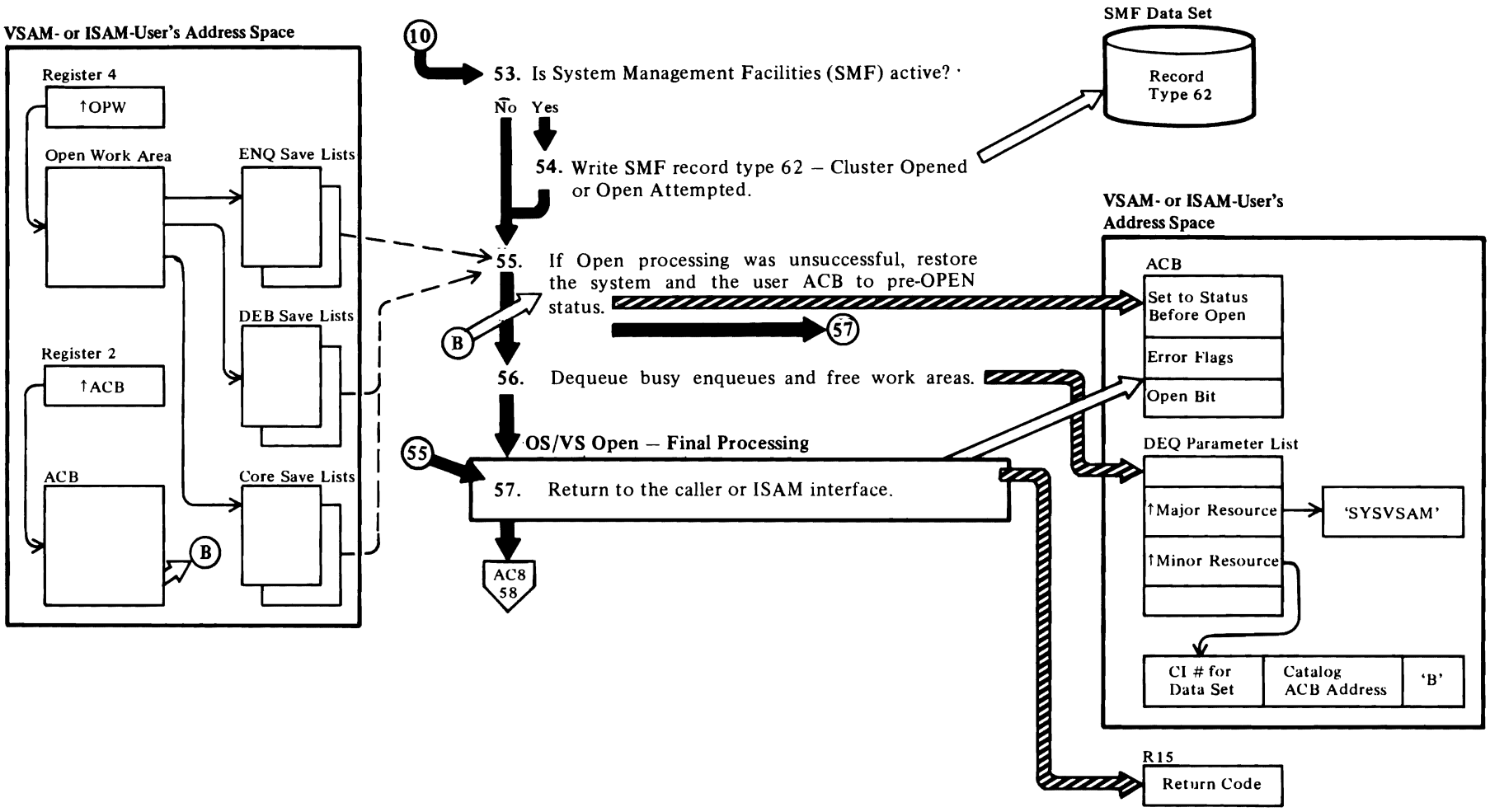
For each string required for processing the path, IDA0192F builds the PLH, BUFC, CPA, IOB, and buffers. These control blocks are described in "Data Areas."

**50** See notes for steps 31 and 32.

**51** See note for step 33.

**52** See note for step 34.

# Diagram AC7. VSAM OPEN: Terminate Open Processing



## Notes for Diagram AC7

**53 IDA0192A: TERM192A, UPSMF**

**54 IDA0192S**

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

**55 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.

**56 IDA0192A: DEQBUSY**

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

**57 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 OS/VS Open sets the appropriate error flag.

**OS/VS Open: Final Processing (after VSAM Open Processing completes):**

The VSAM Open routine returns control to OS/VS Open by putting the identifier of the Open Final Termination routine, C'8N', in the WTG table and transferring control (through the IECRES macro instruction) to the O/C/EOV 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 OS/VS Open Final Termination routine. If not, the next ACB entry in the open parameter list is processed (return to step 4, Diagram AC1).

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

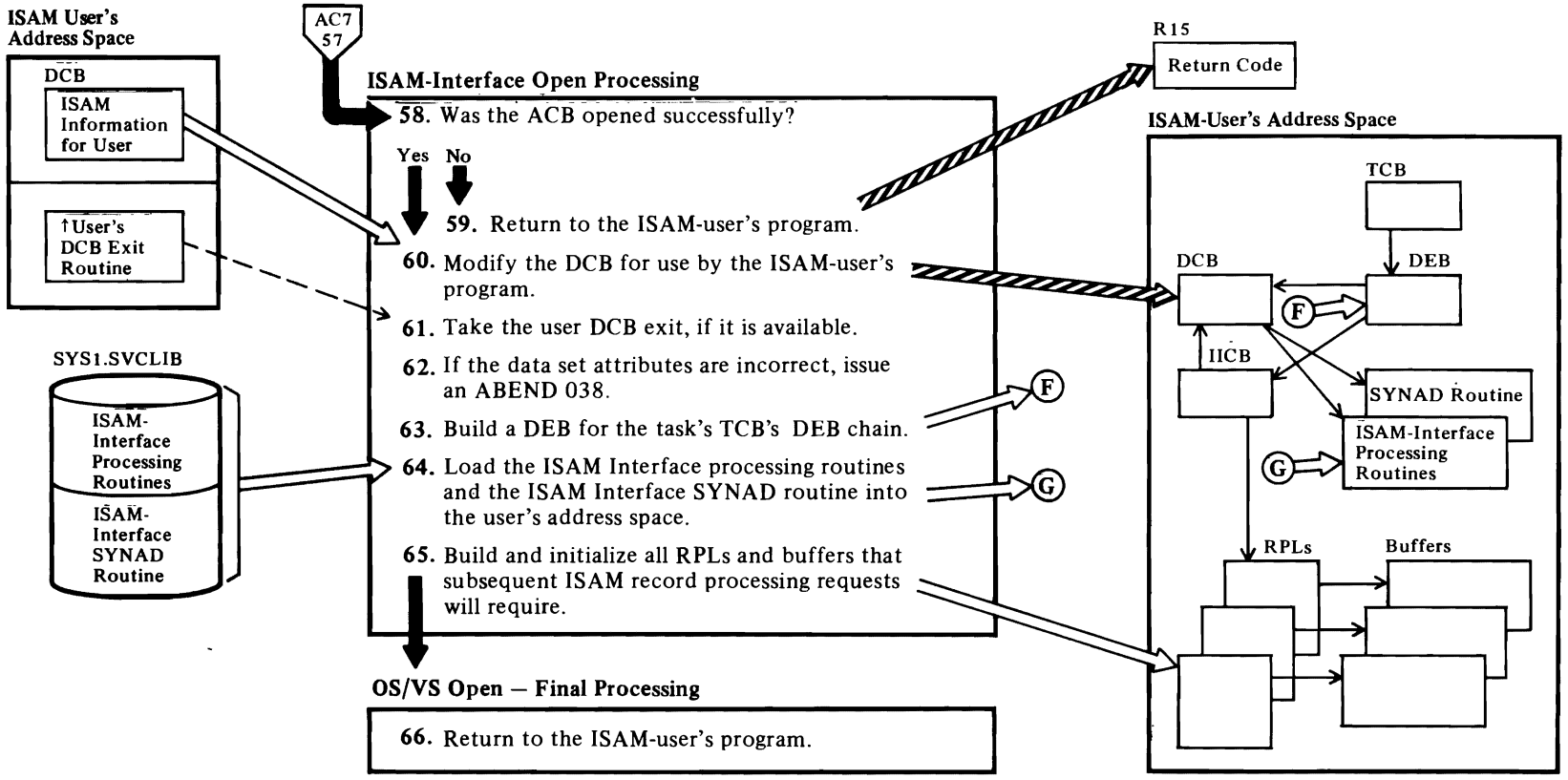
IFG0196V sets the identifier for each VSAM ACB entry in the WTG table to 0.

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

IFG0198N sets the return code in register 15.

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

# Diagram AC8. VSAM OPEN: Connect a User to a VSAM Data Set



## Notes for Diagram AC8

**ISAM Interface Open Processing (continued):**

**58 IDA0192I: OPENACB**

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

**60 IDA0192I: DCBMERGE and AMSMERGE**

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

**61 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.

**62 IDA0192I: VALIDCHK**

ABEND 038 is issued when:

- Access Method Services and DCB values for LRECL, KEYLE, and RKP are not equal, or when
- Reload is attempted—the DCB is opened for OUTPUT with DISP=OLD and the DCB's data set contains records.

**63 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 the ISAM System Integrity Feature 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.

**64 IFG0192I: LOADMOD**

The appropriate ISAM Interface modules are loaded. DCB fields are initialized to point to the 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, Diagram AC1) addresses ISAM Interface exit routines.

See "Data Areas" for details about the EXLST.

The DEB (built in step 63) is initialized to point to the ISAM Interface FREEDBUF routine.

**65 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 each QISAM DCB) 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 off (set to 0)
- Open bit on (set to 1)
- Lock bit off (set to 1)

**OS/VS Open: Final Processing (after ISAM Interface Open Processing completes):**

**66**

OS/VS 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.

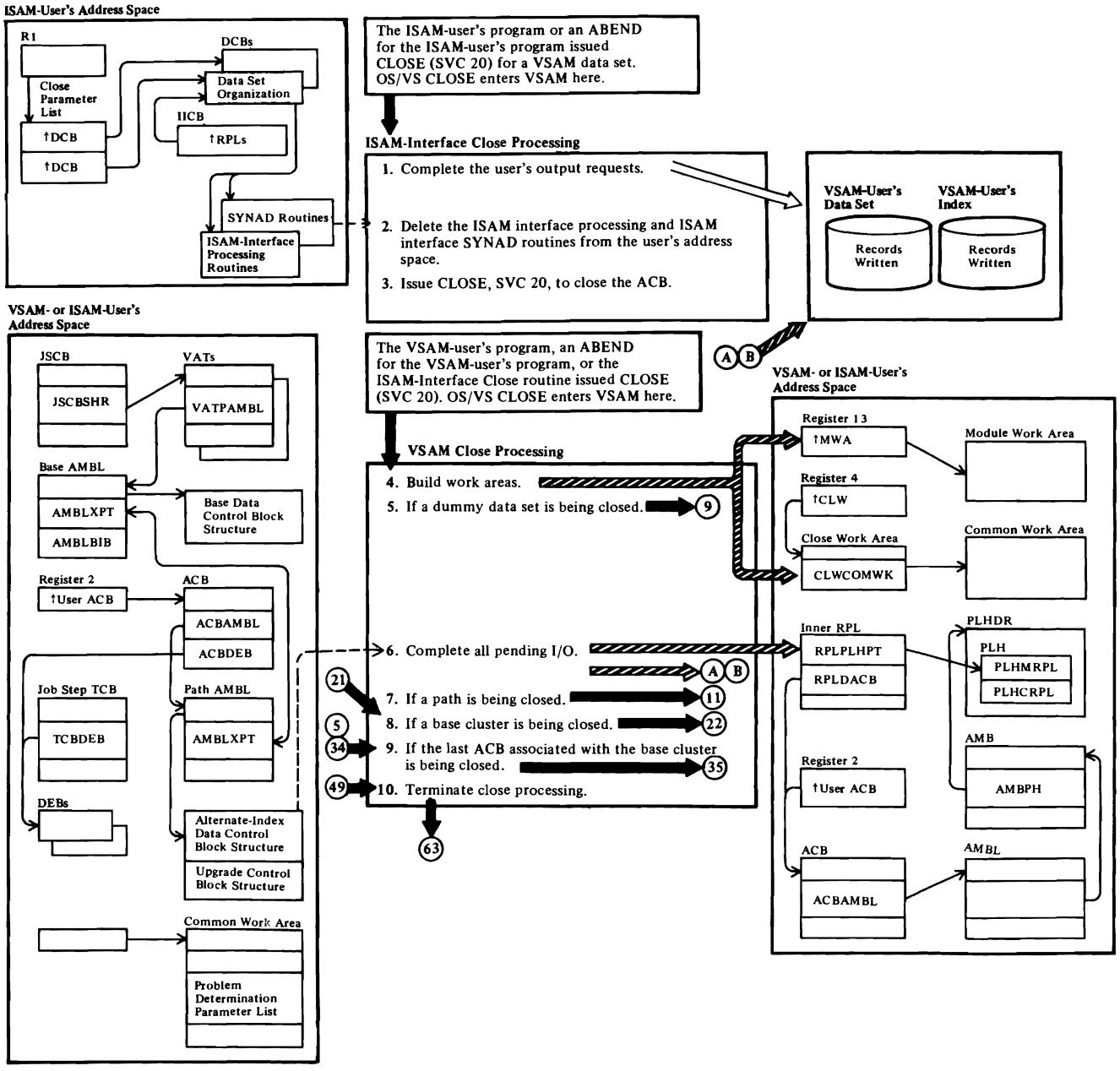
IFG0196V sets the ID field for each DCB-for-VSAM entry in the WTG table to 0.

IFG0196W sets the identifier field for each DCB-for-VSAM entry in the WTG table to C'8N', the identifier of the OS/VS 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, Diagram AC1) 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 results, caused by a branch to address 000.

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



## Notes for Diagram AD1

**Note:** If CLOSE (TYPE=T) is issued, the data set's catalog information is updated to reflect its current status and an SMF record is written. See "Temporary Close (TYPE=T) of a VSAM Cluster" in the Program Organization compendiums for details on the CLOSE (TYPE=T) process.

**ISAM Interface Close Processing:**

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, OS/VS 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 instruction 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 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 instruction 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 instruction (SVC 20) to close the VSAM ACB. When VSAM Close processing completes (the ACB built during ISAM Interface Open processing is closed), ISAM Interface Close processing continues at step 68 (see Diagram AD6).

**VSAM Close Processing:**

OS/VS Close modules (IGC00020 and IFG0200V) allow an ACB to be closed.

IGC00020 bypasses the DEB validity check and the purging of outstanding EXCP requests and, if a VSAM 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 (i.e. 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.

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**

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

**8 IDA0200T: CLSBASE**

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.

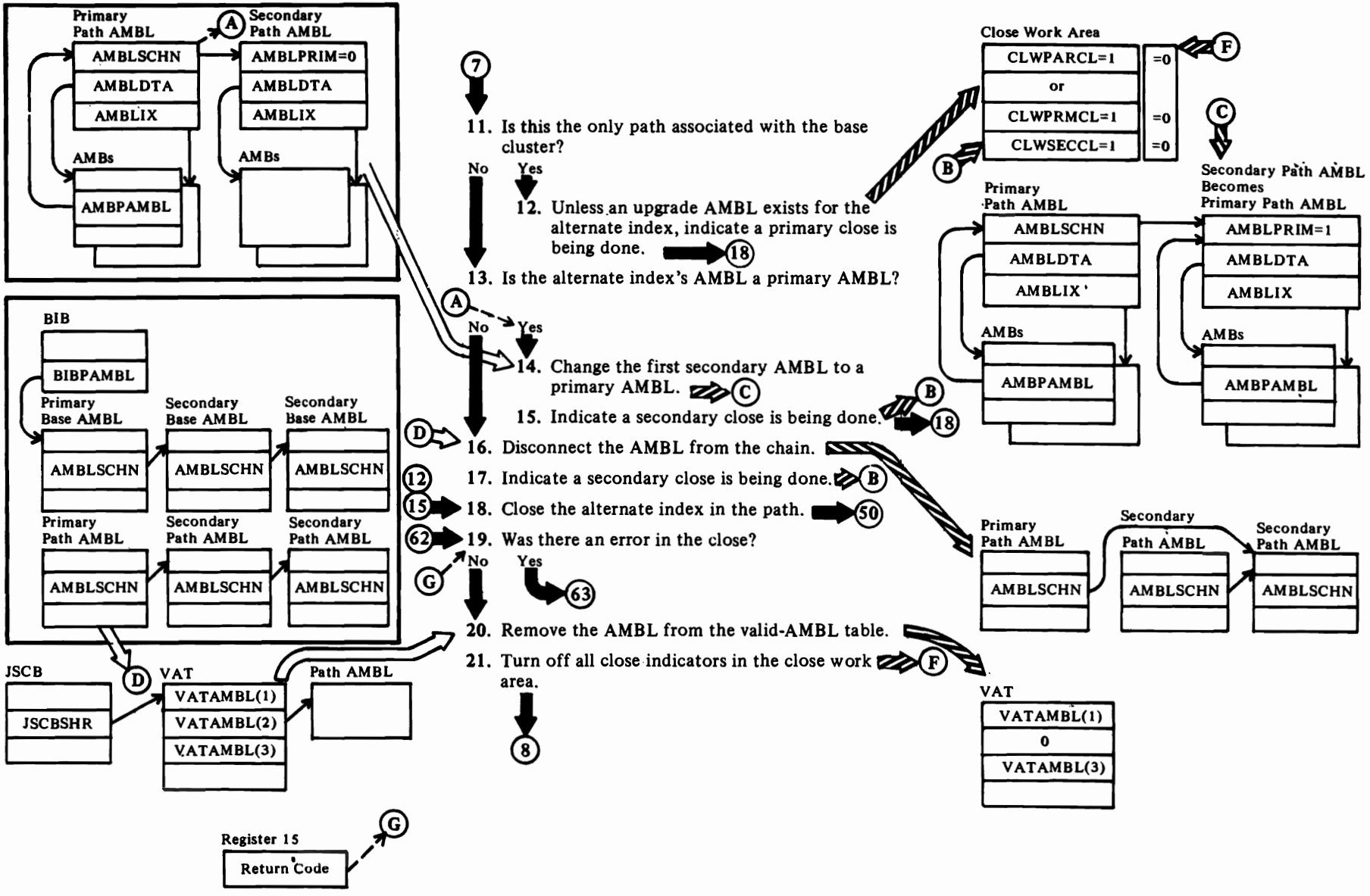
**9 IDA0200T: CLSPHERE**

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 Diagram AD4.

**10 IDA0200T: TERM200T**

Before termination processing, the base AMBL is freed. See Diagram AD6.

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



## Notes for Diagram AD2

Except for step 20, all the processing in this diagram is done by IDA0200B.

**12**

When an upgrade AMBL exists for the alternate indes being closed, a partial close is indicated for Diagram AD5 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**

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.

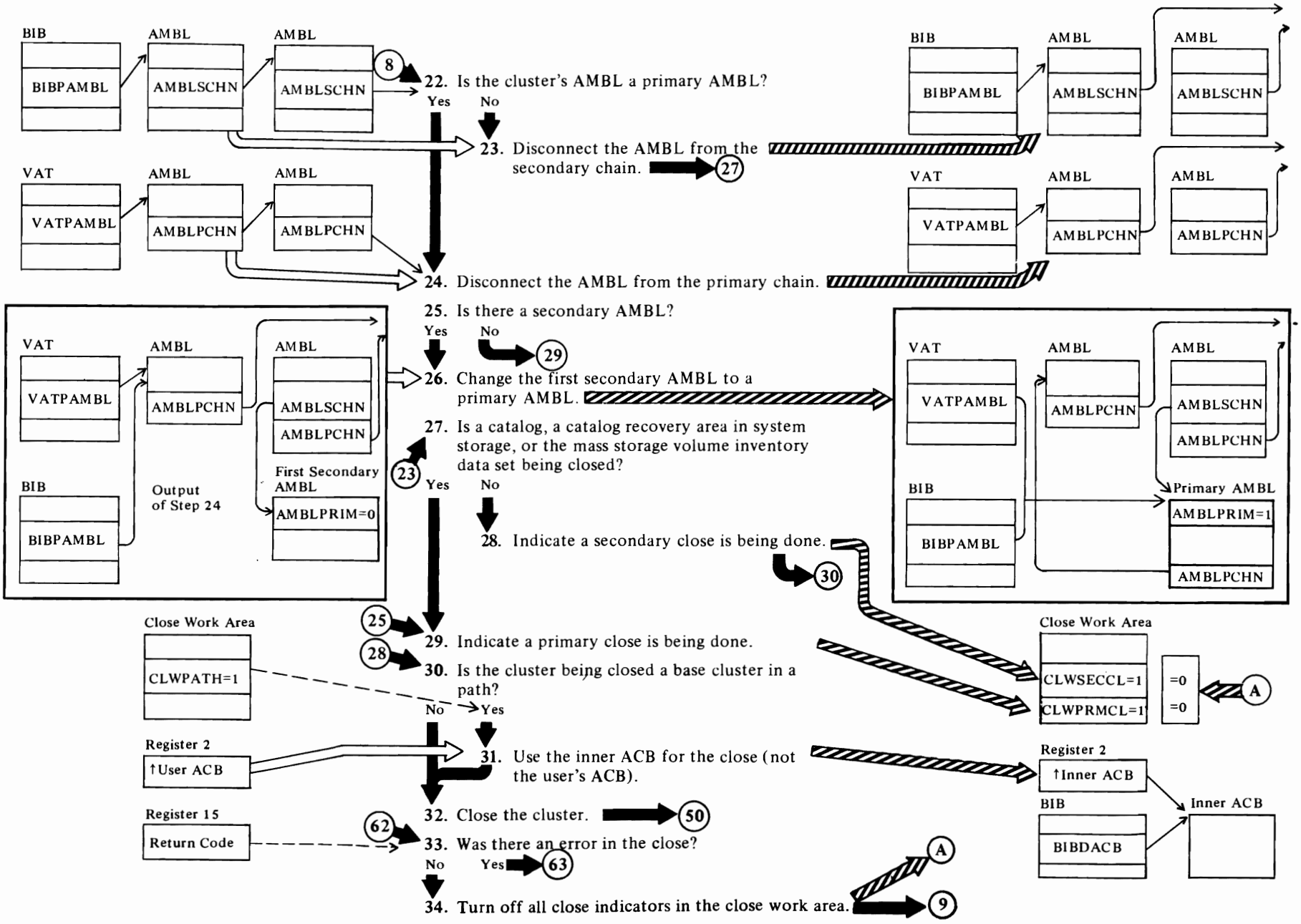
**18**

See Diagram AD5.

**20 IDA0200T: RMOVAMBL**

The storage for the AMBL is freed.

# Diagram AD3. VSAM CLOSE: Close the Base Cluster (or a Cluster Not in a Path)



## 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.

Except for some processing following step 33, IDA0200B does all the processing in this diagram.

**24**

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

**26**

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**

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

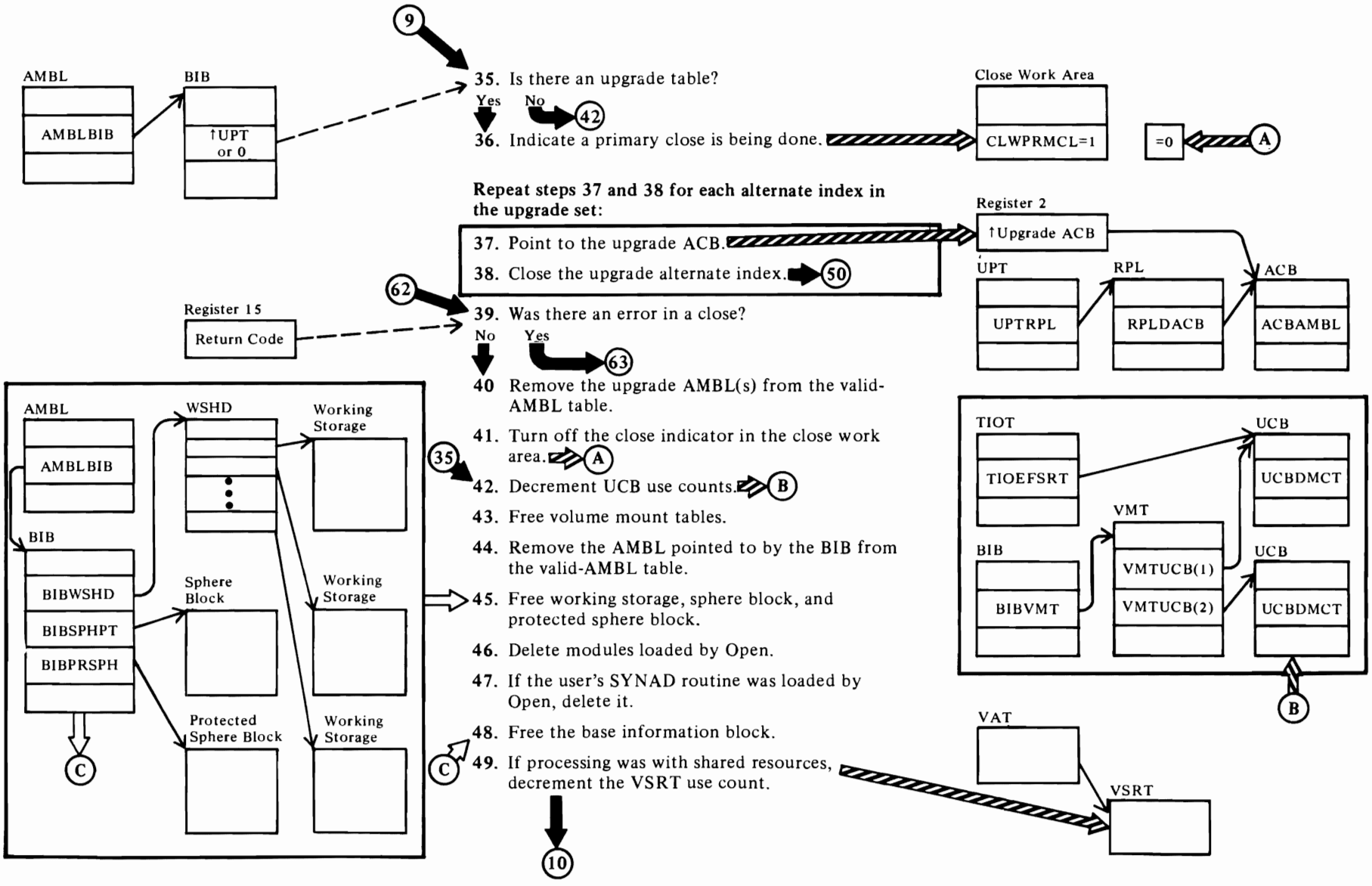
**32**

See Diagram AD5.

**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



## 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 AD5.

**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**

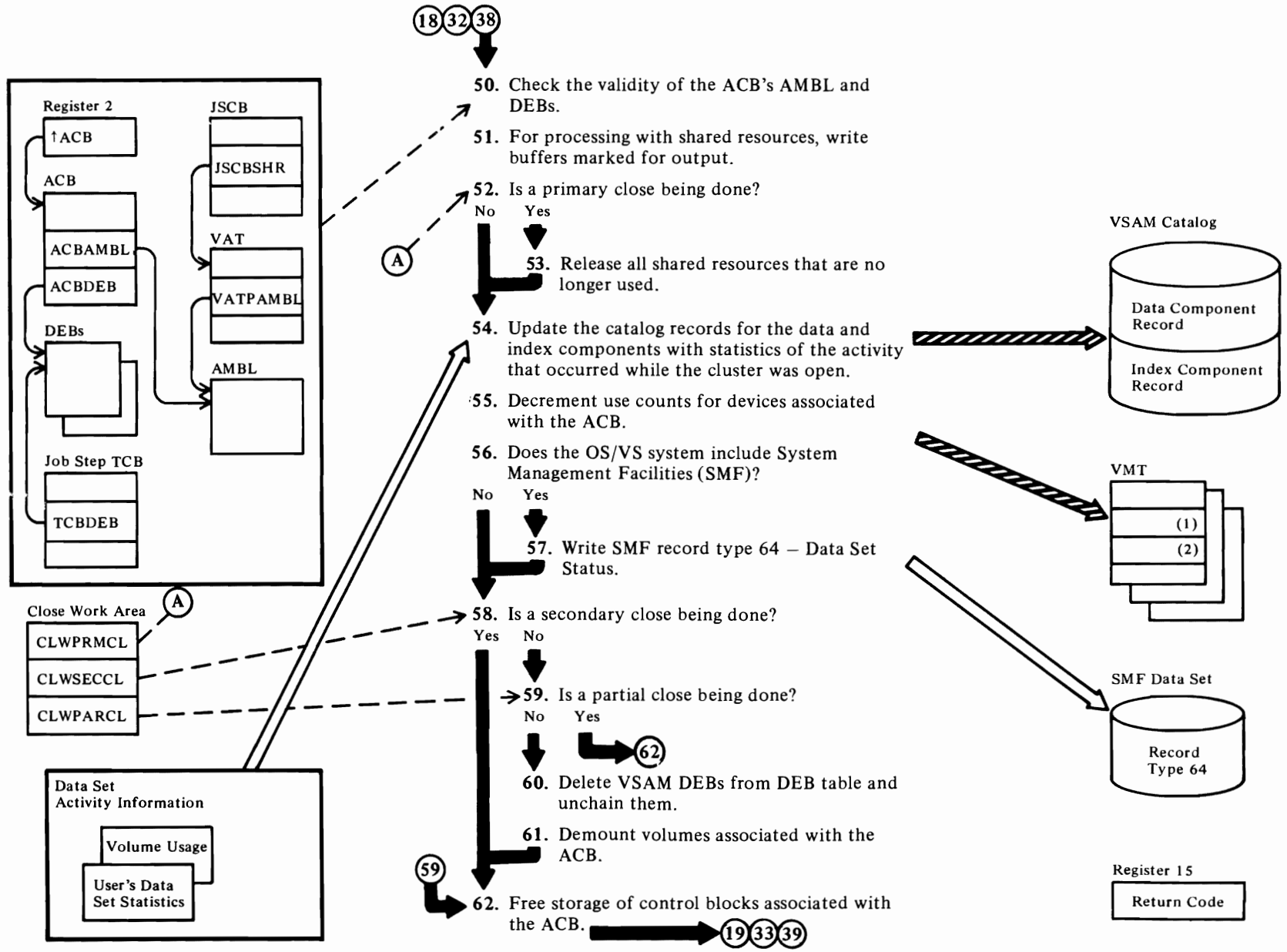
**45 IDA0200T: FREECORE, FREESPHR**

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 a Cluster



## Notes for Diagram AD5

**50 IDA0200B: INIT200B, VALCHECK, PROBDT** (calls **IDA0192P)**

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

**51 IDA0200B: WRITBUFR, GETCORE, WRBUFFER, CBINIT, FREECORE, PROBDT**

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

**52**

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

**53 IDA0200B: SHARE, SHAREDEQ**

DEQ is issued.

**54 IDA0200B: UPCATACB, UPCATDEQ (** calls **IDA0192C), PROBDT**

**55 IDA0200B: VMTPROC**

**57 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/VS 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.

**58**

See note for step 15.

**59**

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.

**60 IDA0200B: DEHOOK**

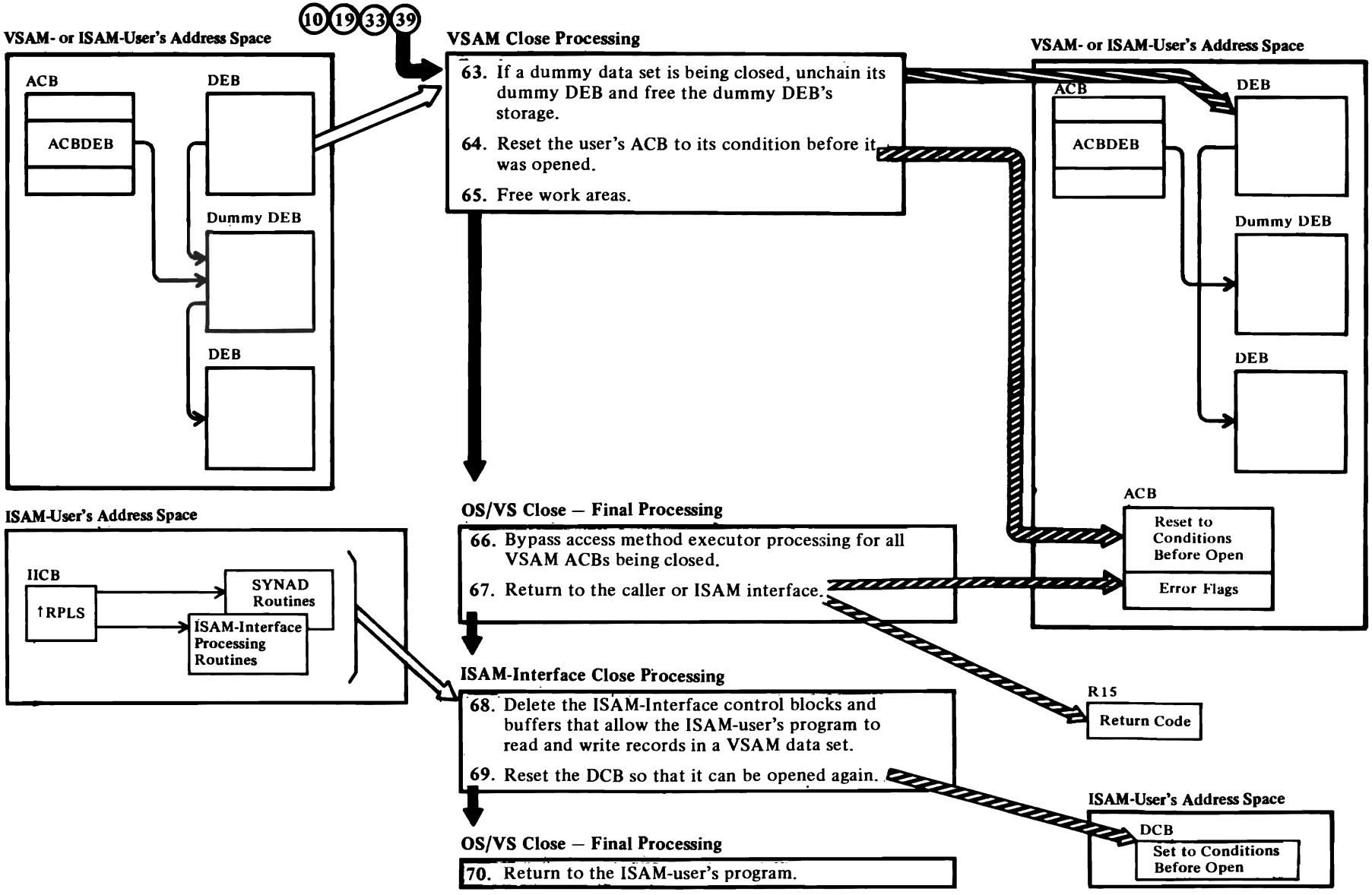
The DEBCHK SVC is used. It removes VSAM DEBs from the TCB DEB chain.

**61 IDA0200B:**

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

**62 IDA0200B: CBRELE**

# Diagram AD6. VSAM CLOSE: Terminate Close Processing



## Notes for Diagram AD6

**63 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.

**64 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

**65 IDA0200T: FREECORE**

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

**66 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 OS/VS Close sets the appropriate error flag.

The VSAM Close routine returns control to OS/VS Close by putting the identifier of the Close Final Termination routine, X'2L', in the WTG table and transferring control (through the IECRES macro instruction) to the O/C/EOV 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 OS/VS Close Final Termination routine. If not, the next ACB entry in the close parameter list is processed (return to step 4).

OS/VS 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 OS/VS Close Final Termination routine.

IFG0202L sets the return code in register 15.

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

**ISAM Interface Close Processing (continued):**

**68 IDA0200S: FREEBFRS, FREEDEB, RESETDCB, FREEWA, FREEMAIN**

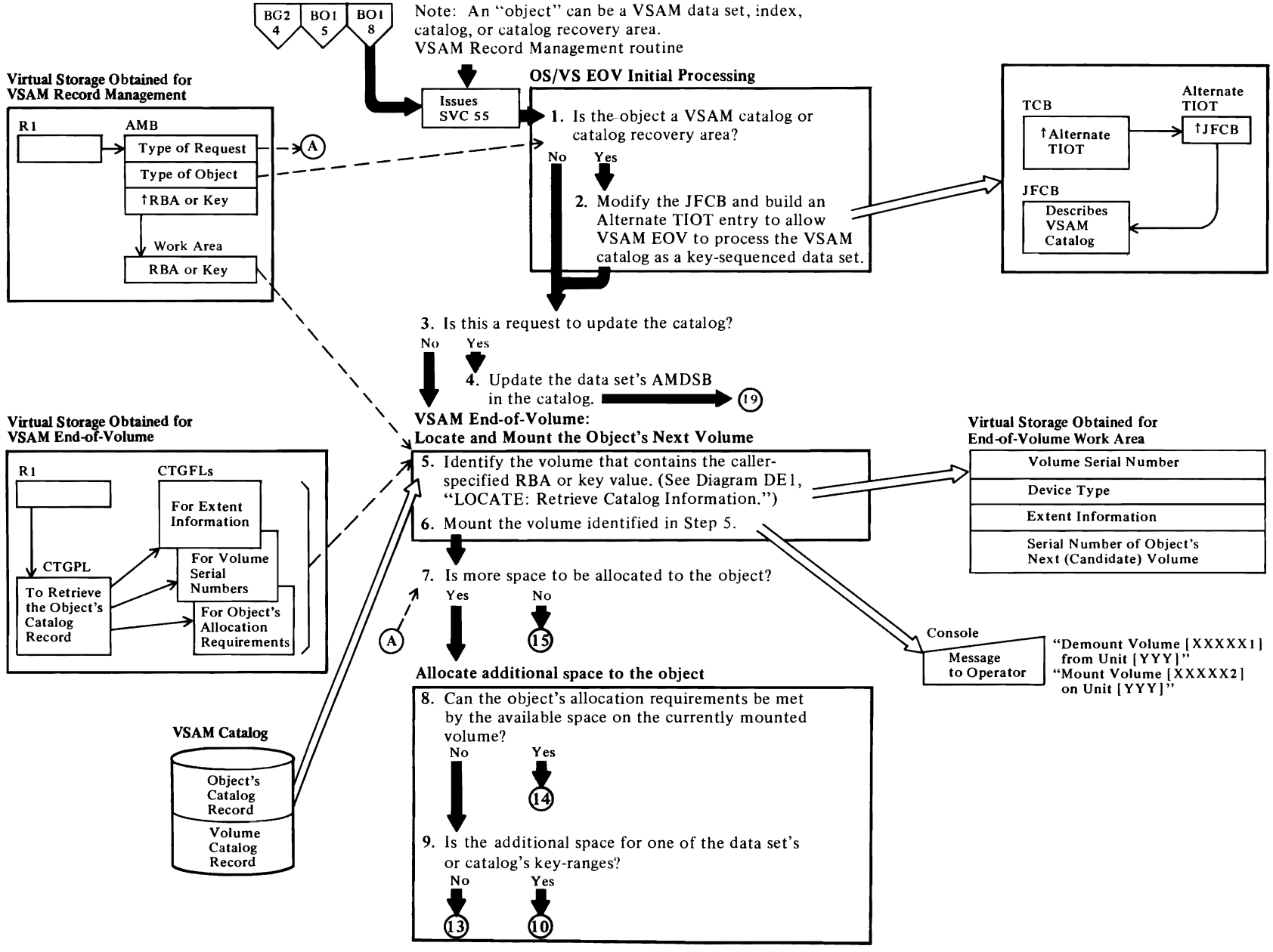
The ISAM Interface Close routine releases the virtual storage obtained for the ACB, the IICB, the DEB, the RPLs, and the ISAM Interface buffers.

**69 IDA0200S: RESETDCB**

The DCB conditions before open are:

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

# Diagram AE1. VSAM End-of-Volume: Obtain the VSAM Object's Next Volume



## Notes for Diagram AE1

Diagram AE1 describes VSAM end-of-volume (EOV) processing. VSAM end-of-volume is called by OS/VS EOV when SVC 55 is issued by VSAM Record Management routines. 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 IGC0005E, IFG0551F**

If register 1 addresses an AMB (for VSAM EOV processing), the OS/VS EOV routine sets the ID field in the where-to-go (WTG) table to C'7A', the identifier of the VSAM EOV routine. The WTG table built for an EOV request contains only one entry. All further OS/VS EOV routines are bypassed.

**2 IFG0550Y**

**3 IDA0557A**

The request is either to handle an end-of-volume condition or to update information in the catalog.

**4 IDA0557A: CATUPD** (which calls **IDA0192C**)

The AMDSB contains statistics for the data set.

**5 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.

**6 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.

**7 IDA0557A: ALLOCSPC**

If the AMB's allocate-space request option indicator is on, the VSAM end-of-volume routine gets more space for the object.

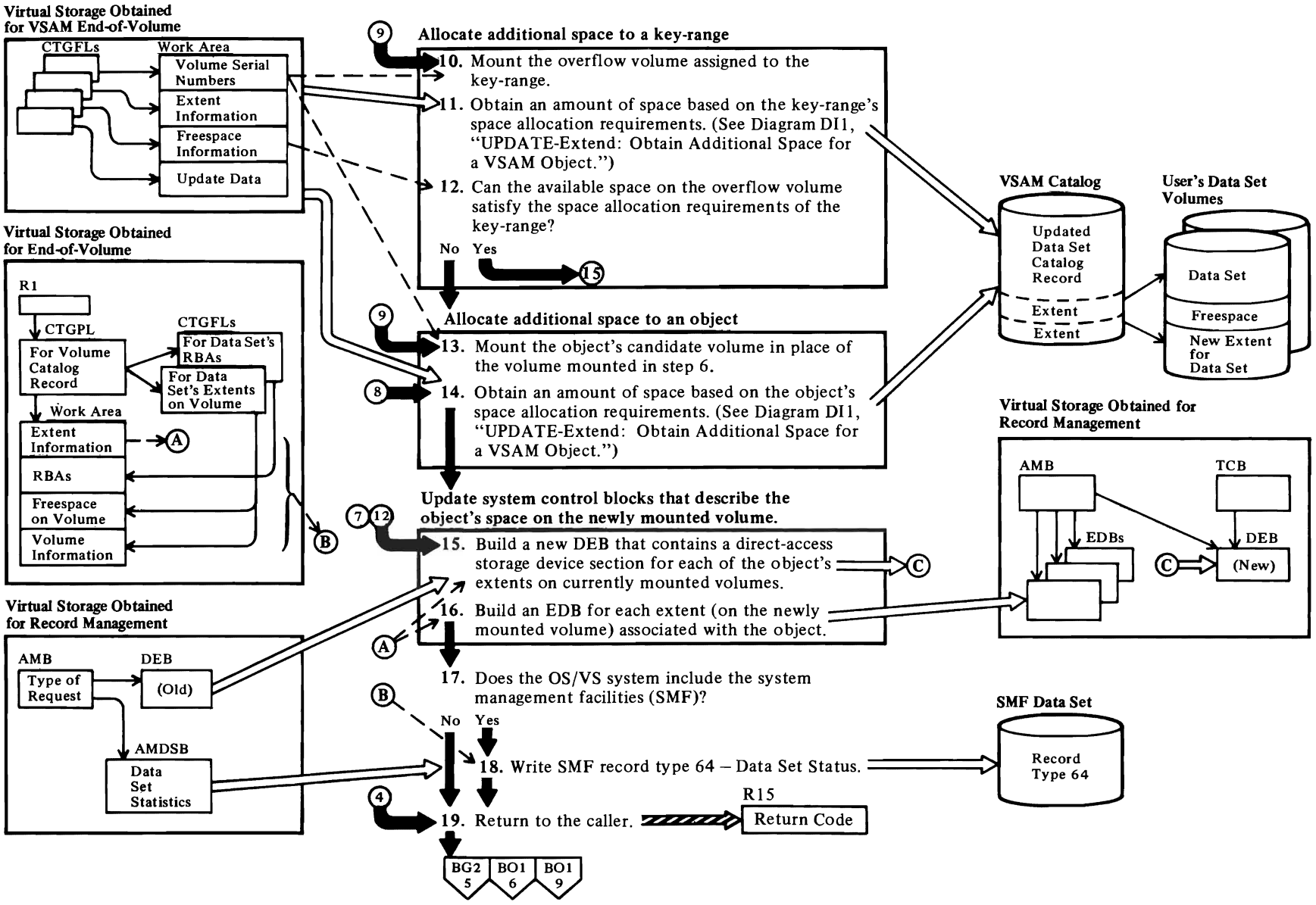
See "Data Areas" for details about the AMB.

**8 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 "Data Areas" for details about the volume catalog record.

# Diagram AE2. VSAM End-of-Volume: Obtain the VSAM Object's Next Volume



## Notes for Diagram AE2

**10 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.

**11 IDA0557A: VOLSW** (calls **CATALC** and **CATUPDVO)**

The object's catalog record describes its space allocation requirements.

**12 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.

**13 IDA0557A: ALLOCSPC** (calls **VOLSW)**

If the volumes are full, and no other volume (candidate) is assigned to the object, the VSAM EOV routine sets the space-not-allocated return code and returns to the caller.

See *OS/VS2 SVS Independent Component: Access Method Services* for a description of how candidate volumes are assigned to VSAM objects.

**14 IDA0557A: CATALC**

The object's catalog record describes its space allocation requirements.

See "Data Areas" for details about the catalog record details, and the volume information set-of-fields.

**15 IDA0557A: CTLBLK** (calls **DSCTLBLK)**

See "Data Areas" for details about the ACB and EDB. See *OS/VS2 Data Areas* for details about the DEB.

The VSAM EOV routine builds a new DEB and EDB that replaces 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 EOV process.
- Each extent that defines the object's newly obtained space (if any).
- None of the object's extents on volumes that were demounted.

**16 IDA0557A: DSCTLCLK** (calls **CATLOCXT** and **CATLOCRB)**

See "Data Areas" for details about the data set catalog record, the volume information set of fields, and the EDB.

**18 IDA0557A: SMFUPD** (calls **CATLOCDS)**

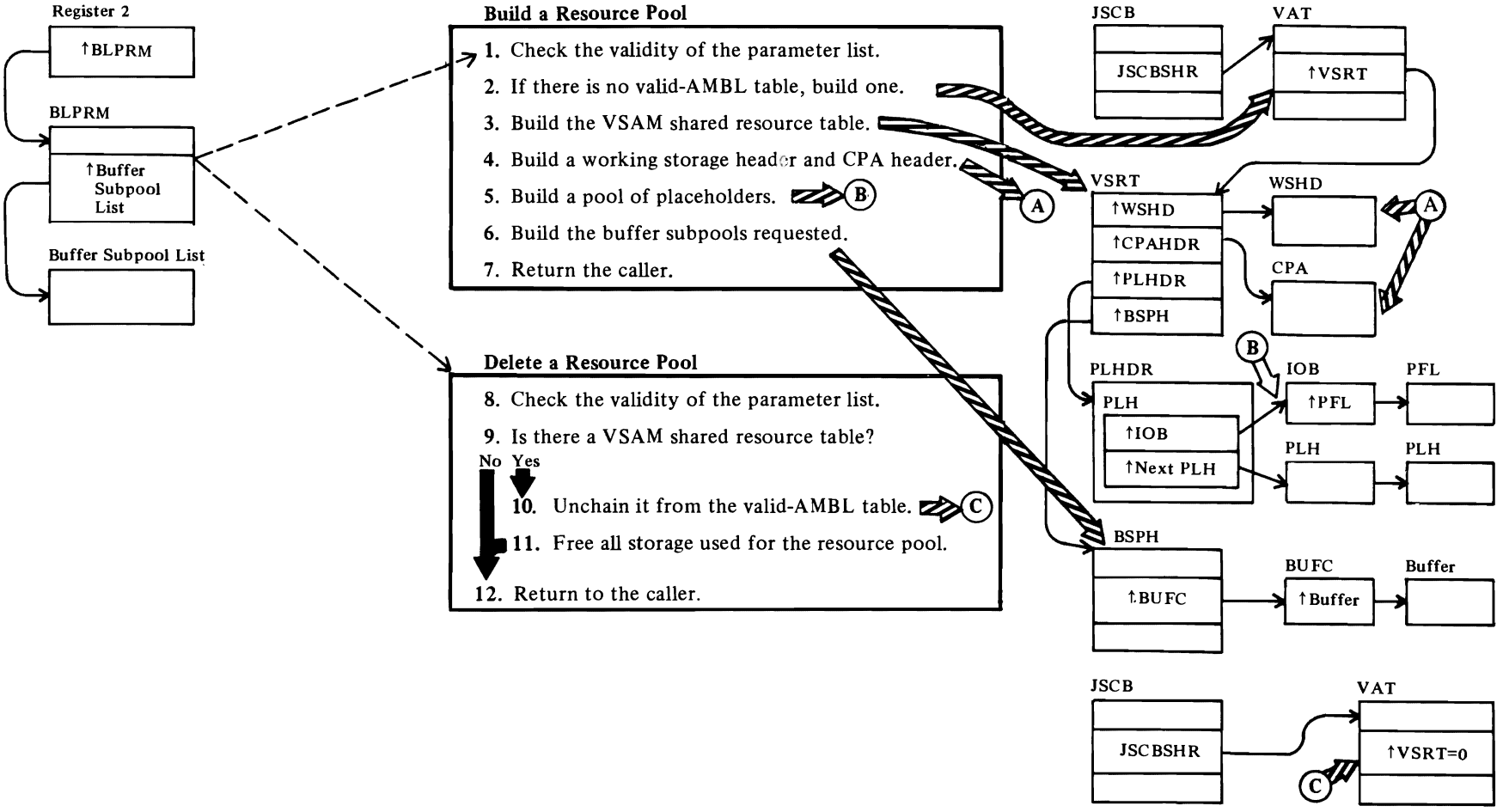
See *OS/VS System Management Facilities (SMF)* for a description of SMF record type 64.

**19 IDA0557A: TERM, PROBDET**

See "Diagnostic Aids" for details about the VSAM End-of-Volume return codes and error codes.

If an error is detected, the VSAM End of Volume routine attempts to determine the type of error and builds a message describing the error.

# Diagram AF1. BLDVRP/DLVRP: Build or Delete a VSAM Resource Pool



## Notes for Diagram AF1

**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.

**8 DLVRP**

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

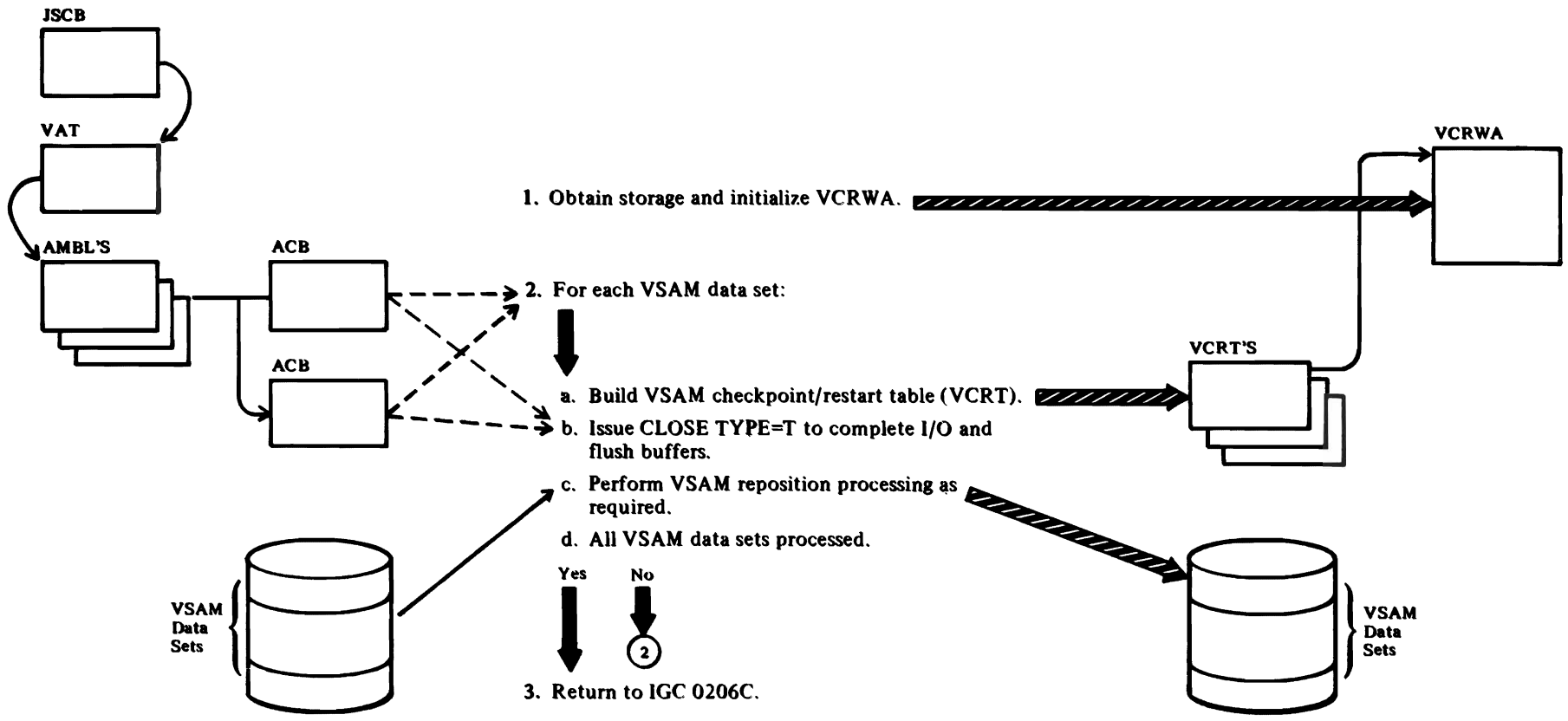
**9**

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

**10 IDA0192Y: DELVRP**

**11 IDA0192Y: FREEVSRT**

# Diagram AG1. VSAM Checkpoint: Checkpointing VSAM Control Blocks



## Notes for Diagram AG1

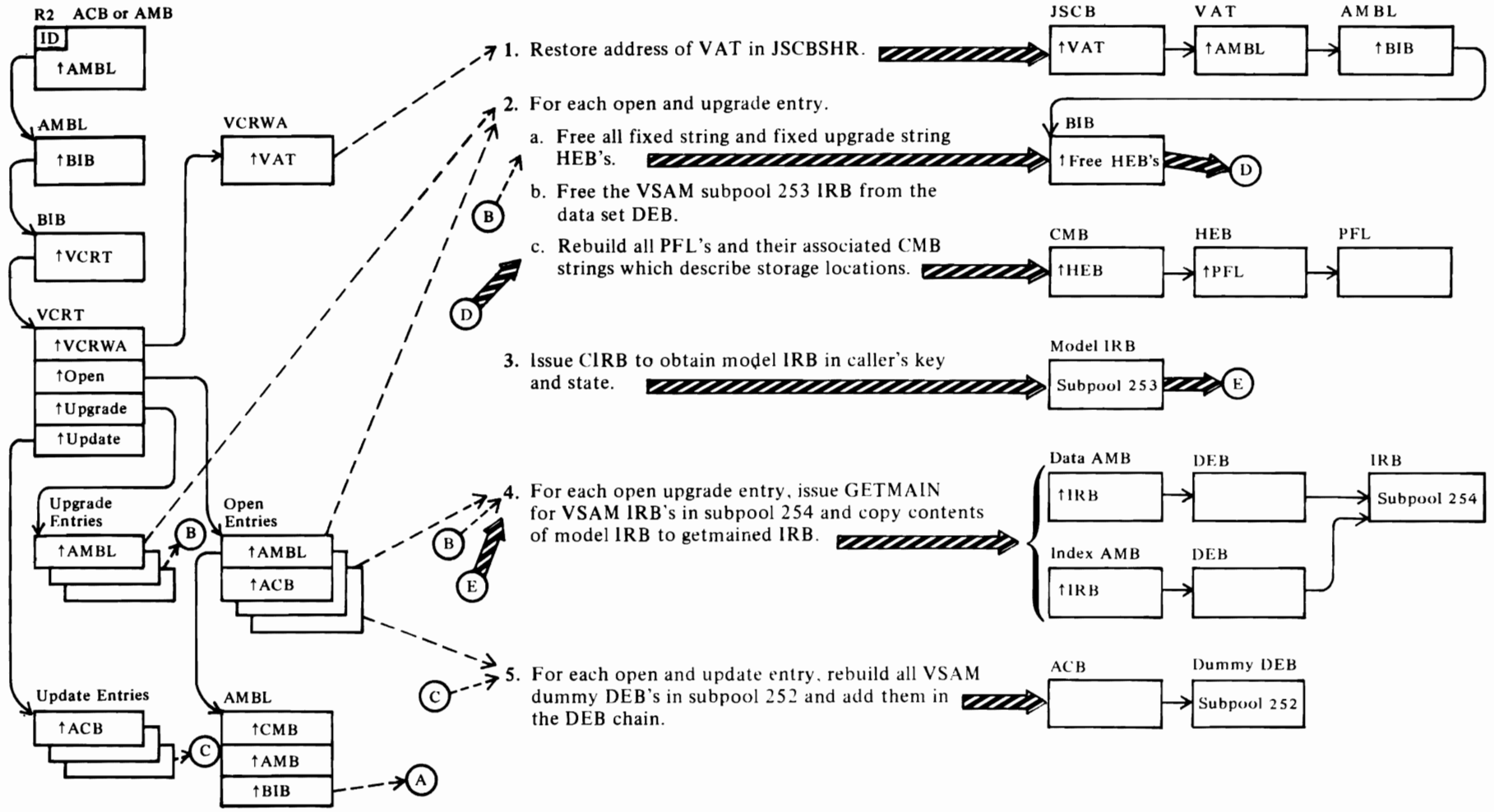
IDA0C06C receives control from IGC0206C via BALR14,15.

1 Obtain and initialize VCRWA. The JSCBSHR field is saved in the VCRWA so that VSAM restart can restore it.

2 For each VSAM data set, build a VCRT and chain to the primary AMBL's BIB (each VCRT points to the VCRWA), issue CLOSE TYPE=T, and perform reposition processing. The current CI is gotten from the data set and is reflected to the restart half of checkpoint/restart.

7 Control returns to IGC0206C.

# Diagram AH1. VSAM Restart: Rebuild VSAM Control Blocks



## Notes for Diagram AH1

**VSAM restart**

**1** IDA0A05B restores the address of the VAT in the JSCB. IDA0A05B is called from IGC^A05B.

**2a IDA0A05b: INITLSQA,FREEHEBS.**

For each primary AMBL, free all fixed string and fixed upgrade string HEBs associated with all VCRT open and upgrade entries.

**2b IDA0A05B: INITLSQA,FREEIRBS.**

For each primary AMBL, free all subpool 253 IRBs associated with all VCRT open and upgrade entries.

**2c IDA0A05B: INITLSQA,BUILDBLK.**

For each primary AMBL, rebuild all PFLs and their associated CMB HEBs. Open module IDA0192M is called to obtain the storage.

**2c IDA0A05B: INITLSQA,BUILDBLK.**

For each primary AMBL, rebuild all PFLs and their associated CMB HEBs. Open module IDA0192M is called to obtain the storage.

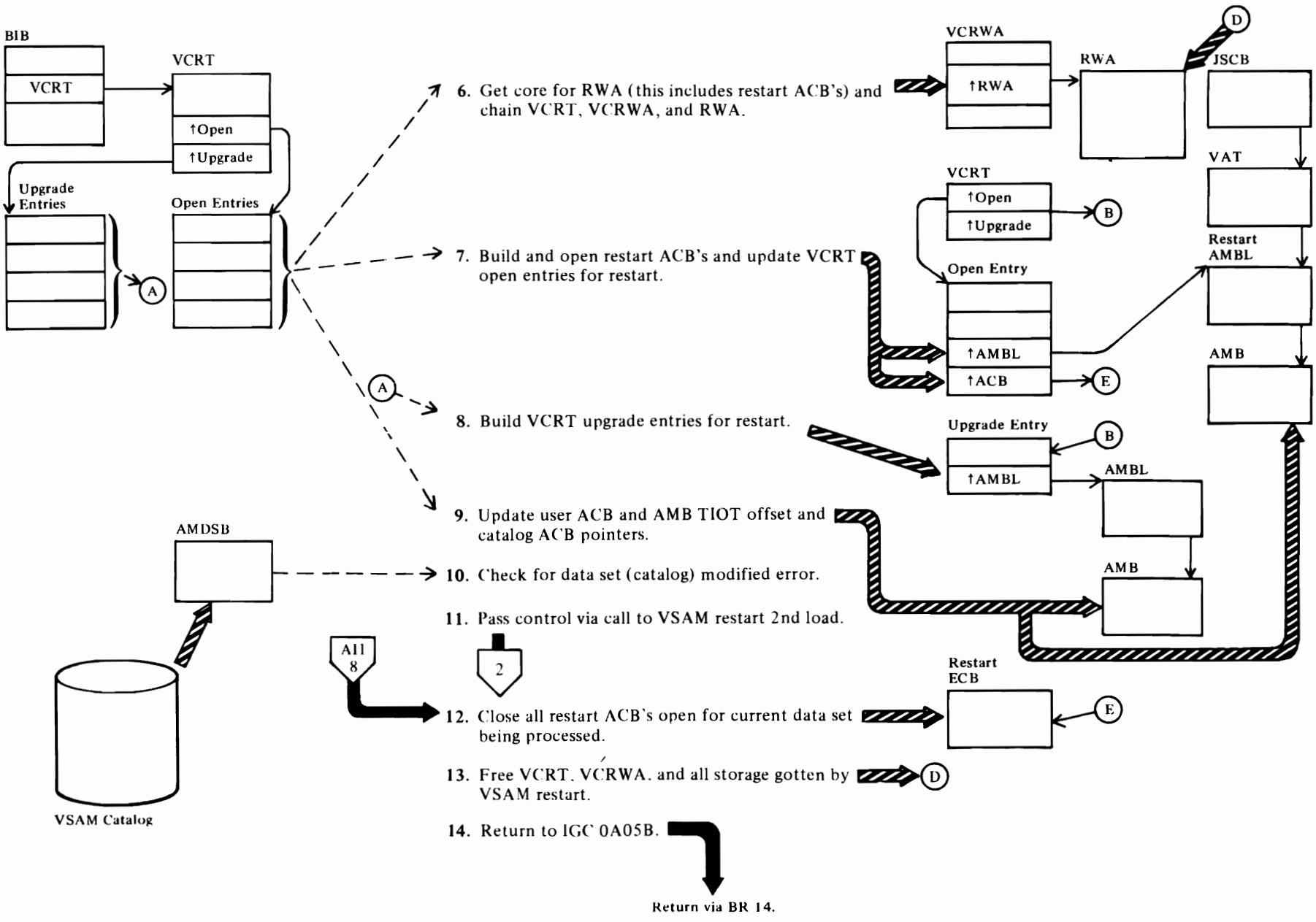
**3 IDA0A05B: INITLSQA,GETIRBS.**

Obtain a model IRB via CIRB in the caller's key and state, and reconstruct all VSAM DEBs in subpool 254.

**5 IDA0A05B: INITLSQA,GETDEBS.**

Rebuild all VSAM dummy DEBs in subpool 252. The DEBs obtained by VS restart are freed.

# Diagram AH2. VSAM Restart: Rebuild VSAM Control Blocks



## Notes for Diagram AH2

**VSAM Restart**

**6 IDA0A05B: INITLZ**

Get storage for the restart ACBs, RWA, and RPL.

**7 IDA0A05B: OPENACB**

The restart ACBs are opened, and the VCRT open entries are updated to reflect the locations of the restart AMBLs and ACBs.

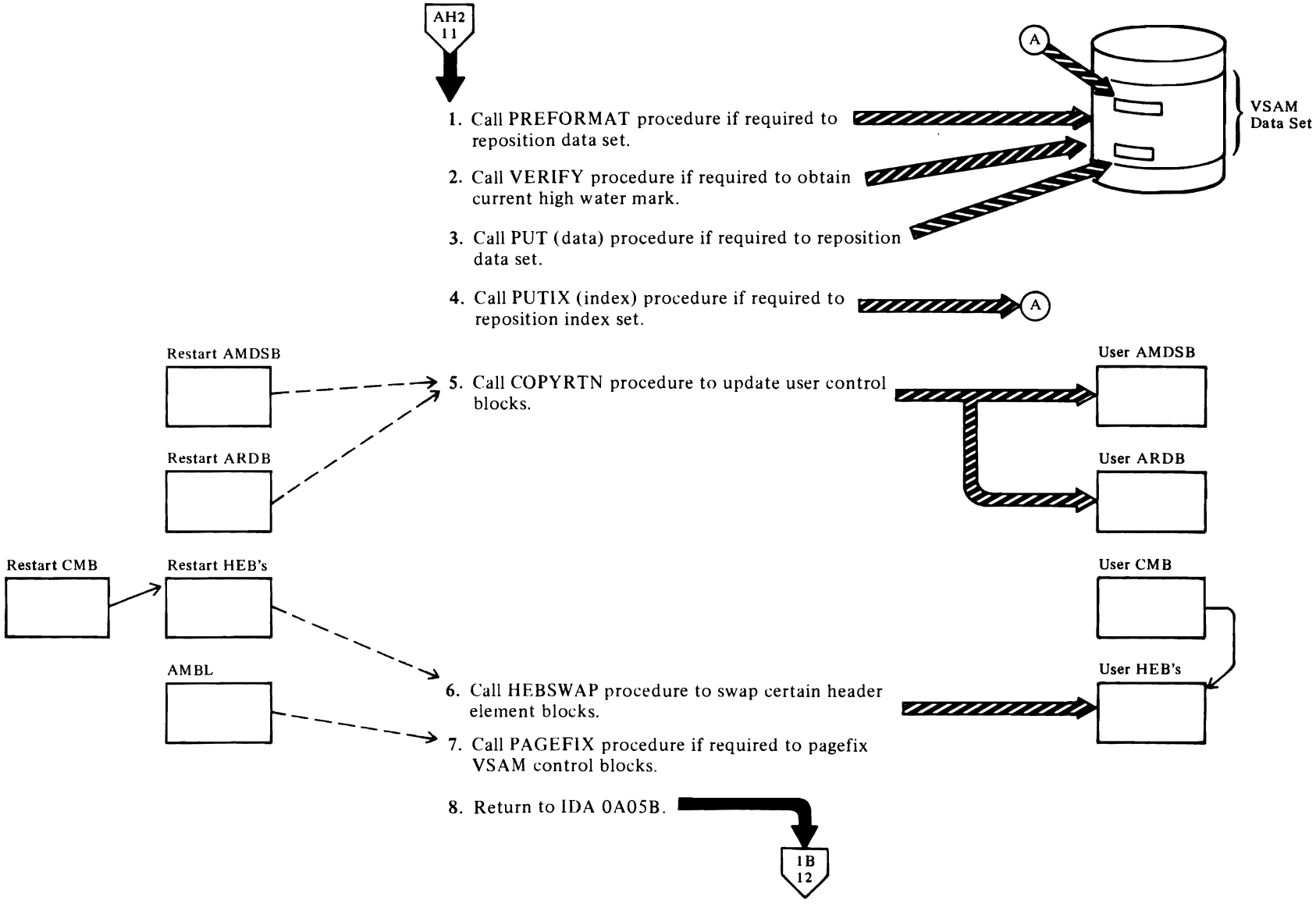
**8 IDA0A05B: UPGRADE,UPDATE**

The immediate upgrade entries are built, and the VCRT upgrate entries are updated. The catalog ACB address and AMB/ACB TIOT offsets are updated to reflect restart time pointers.

**11 IDA0A05B:**

Restart load 2 (IDA0B05B) is called to complete repositioning or data set verification. On return, close all restart ACBs, free restart work areas, and return to IGC0A05B. IGC0A05B will XCTL to VS restart module IGC0V05B.

# Diagram AI1. VSAM Restart: Rebuild VSAM Control Blocks



## Notes for Diagram AI1

**IDA0B05B receives control from IDA0A05B**

**1 IDA0B05B: PREFRMAT**

PREFRMAT is called if reposition is required (create mode KSDS or ESDS output CRPS = NCK,NRE, or speed not specified and data set has been used).

**2 IDA0B05B:VERIFYHU**

If data set is not in create mode and the immediate upgrade data sets exist or if data set is KSDS/RRDS, call VSAM VERIFY to obtain current high used RBA for data set.

**3 IDA0B05B:PUTRTN**

Call VSAM PUT to rewrite the data CI for ESDS (noncreate mode) data sets that require repositioning.

**4 IDA0B05B:IDXPUT**

Call VSAM PUTIX to rewrite the highest index CI of each index level that existed at checkpoint time.

**5 IDA0B05B:COPYRTN**

Copy, the restart AMDSB and ARDB to the user's AMDSB and ARDB respectively, and also swap the restart AMB DEB and EDB pointers with the DEB and EDB pointers in the user's AMB.

**6 IDA0B05B:HEBSWAP**

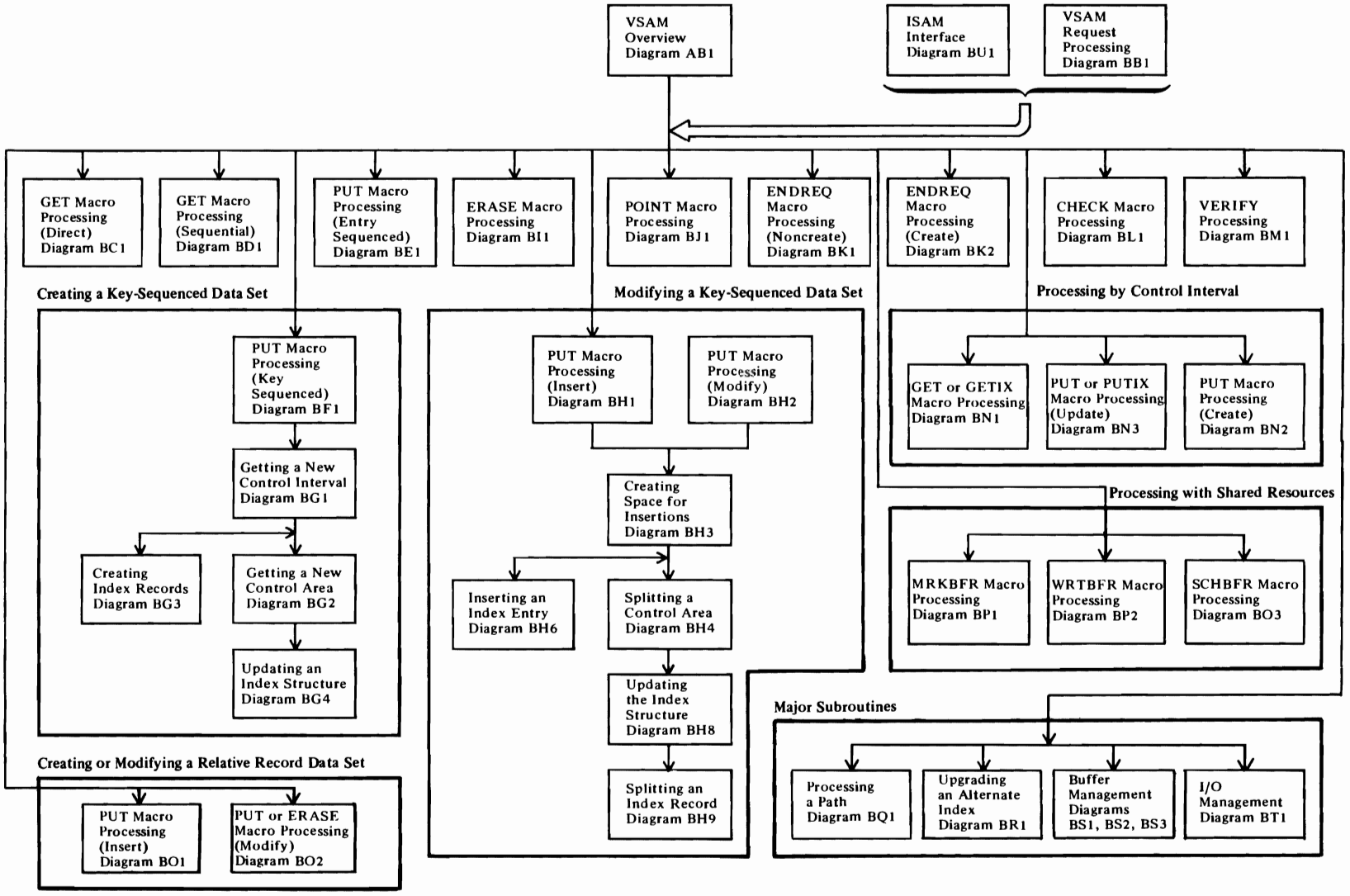
Swap certain user HEBs with restart HEBs. The HEBs to be swapped are the protected string, the DEB blocks, and the EDB blocks.

**7 IDA0B05B:PAGEFIX**

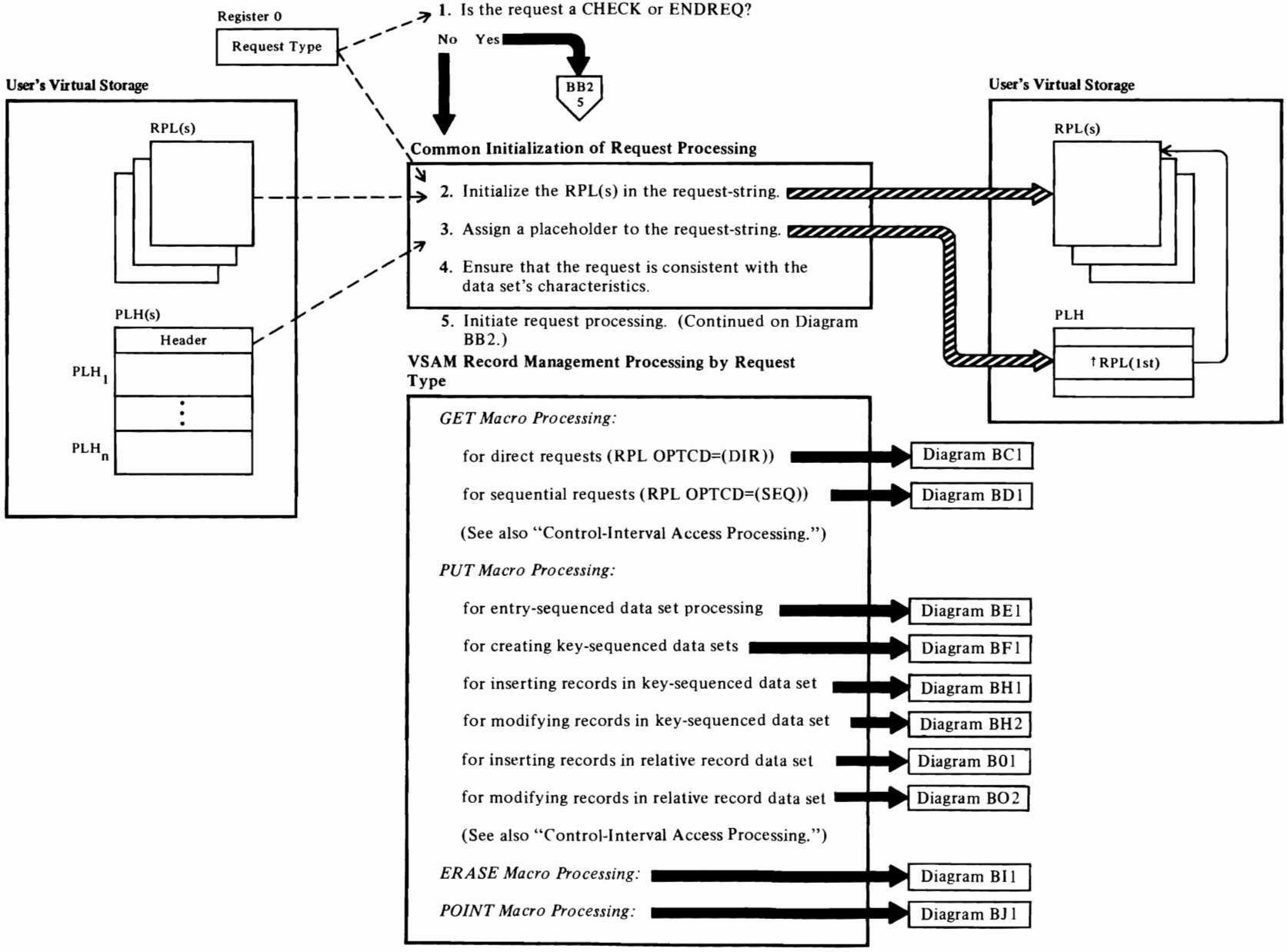
Perform page fix if ICIP with page fix option specified.

**8** Return to IDA0A05B via BR14.

## Diagram BA1. Record Management Table of Contents



# Diagram BB1. VSAM Request 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 or processing is loading a data set that was empty when it was opened, an error code is set and a return is made to the caller. Otherwise, IDA019R1 calls IDAXGLPH in module IDA019RU 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.

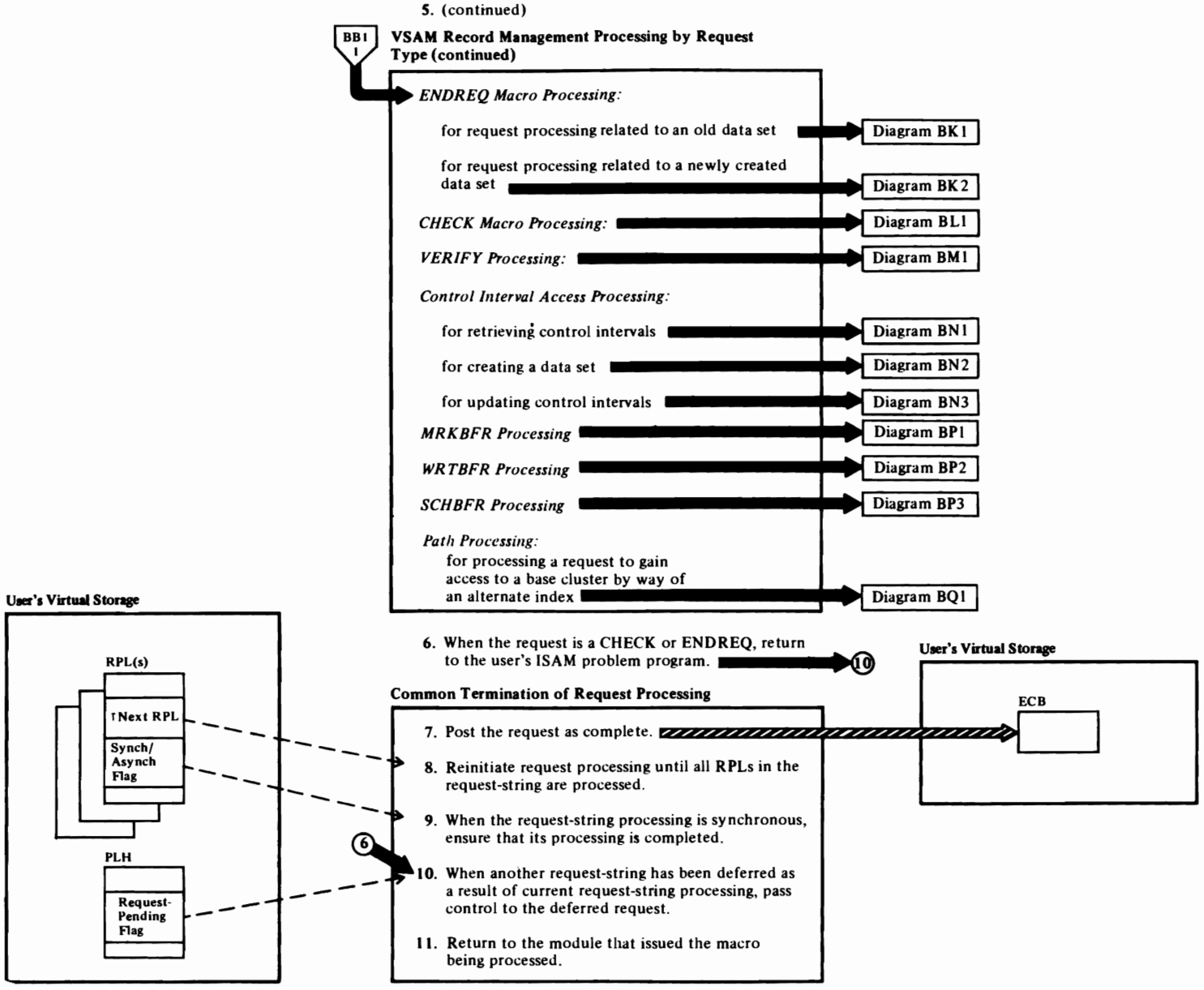
An ERASE request against an entry-sequenced data set is not allowed.

An address 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 Request Processing



## Notes for Diagram BB2

**10**

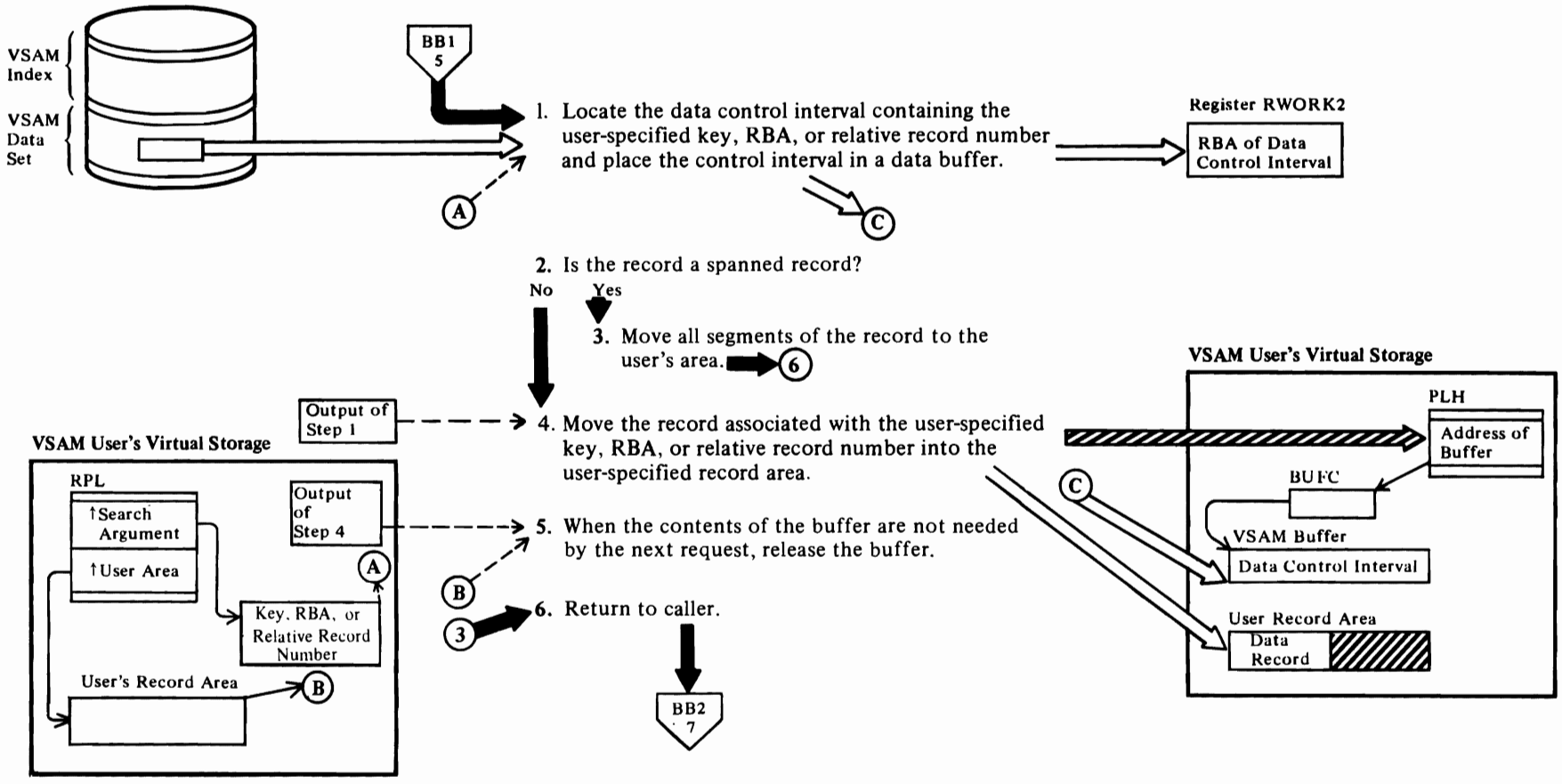
When two request strings are competing concurrently for a serially reusable resource, the second string is deferred.

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 BC1. GET-Direct Processing: Direct Retrieval



## Notes for Diagram BC1

**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 and 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** Doesn't apply to a relative record data set.

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

A spanned record is delivered.

**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 record that contains the prime key and all alternate key). (See Diagram BR1.)

**Relative Record Processing**

**IDA019RR**

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

**5 IDA019R4: RLSEBUFS** calls **IDA019RZ**

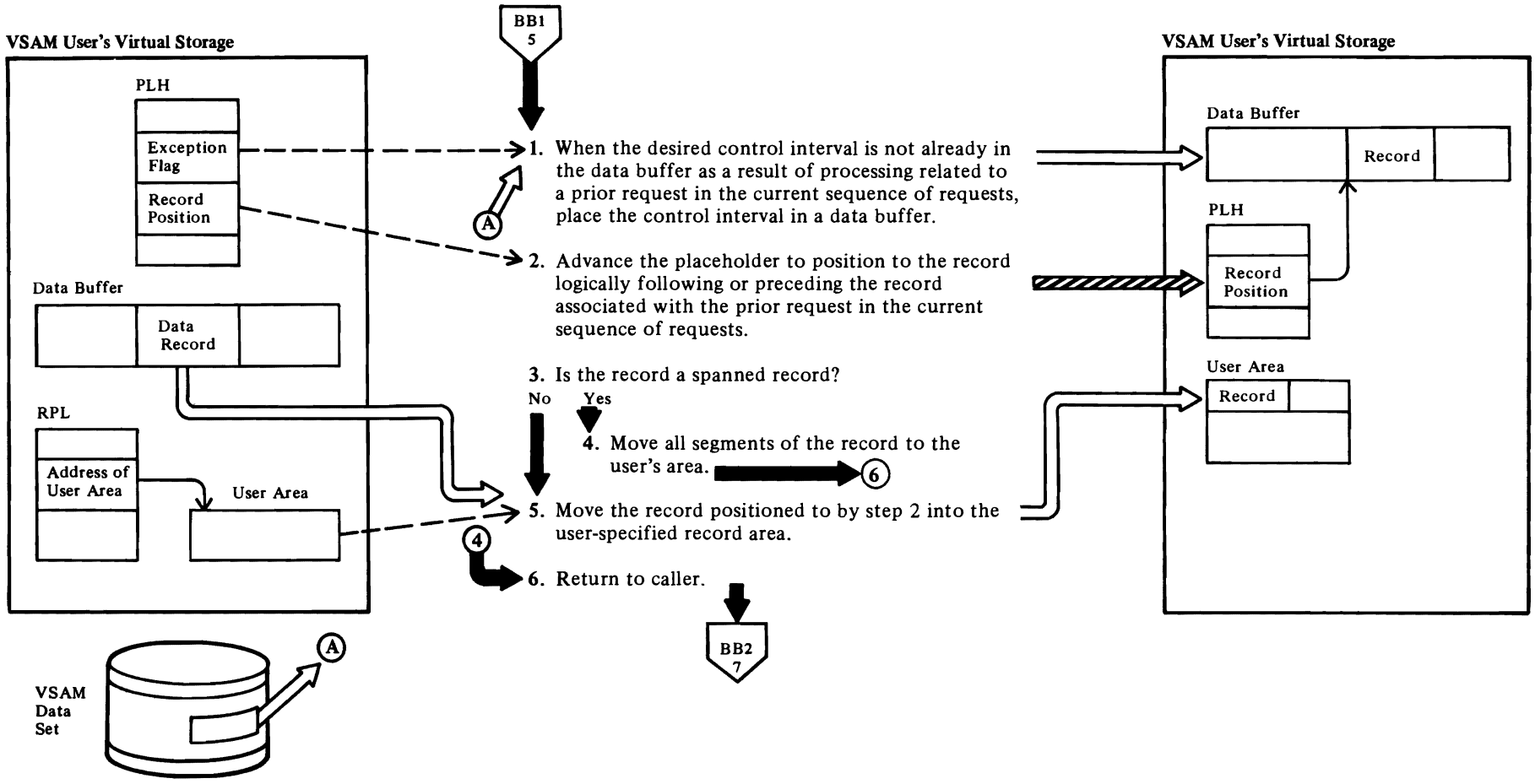
If the request is direct and update, note-string-position, or locate mode processing options is not specified, the contents of the buffer are not needed by the next request and the buffer is released. If the user's processing with shared resources, any index buffer is freed.

**Relative Record Processing**

**IDA019RR** calls **IDA019RZ (IDAFREEB)**

If the request is direct and update, note-string-position, or locate mode is not specified, the buffer is released.

# Diagram BD1. GET-Sequential Processing: Sequential Retrieval



## Notes for Diagram BD1

**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 RBA sequence in the data buffer.

If the record pointer points 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 preceding record in RBA sequence in the data buffer.

If the record pointer points to the beginning of a control interval, the following processing is performed:

**IDAADVPH** calls **IDA019RZ (IDAFREEB)**

The current is released.

**IDAADVPH** calls **IDA019RZ (IDAGNXT)**

The preceding control interval is retrieved.

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

A spanned record is delivered.

**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 record that contains the prime key and all alternate keys). (See Diagram BR1.)

**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 is performed:

**IDA019RR** calls **IDA019RZ (IDAFREEB)**

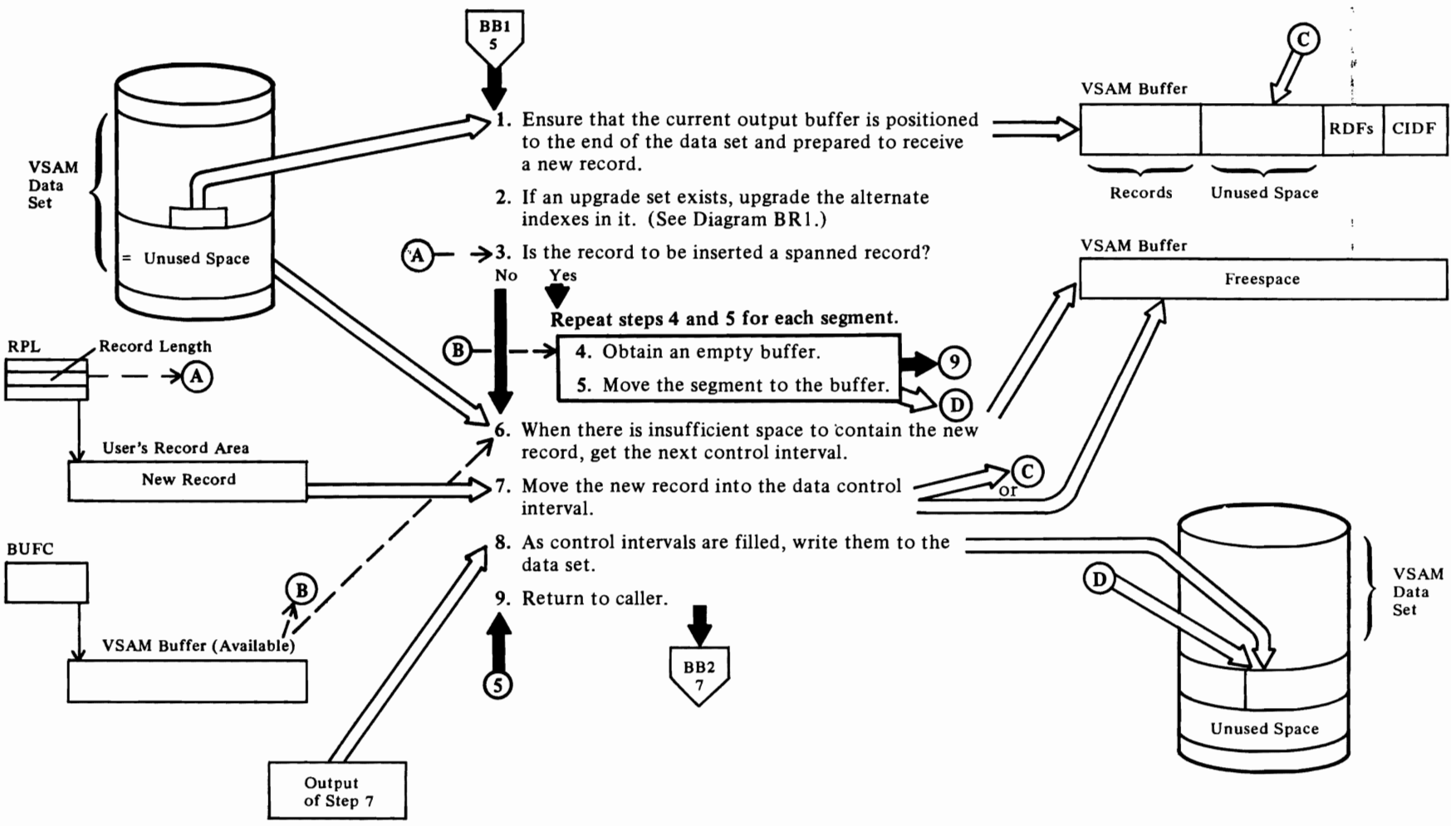
The current buffer is released.

**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 BE1. PUT-Entry-Sequenced Processing: Create or Insert at End of Data Set



## Notes for Diagram BE1

**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 IDA019R2 (IDASBF)) and the desired control interval is moved into the buffer (IDA019RA calls IDA019R2 (IDAGRB)).

**2 IDA019RH** 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.

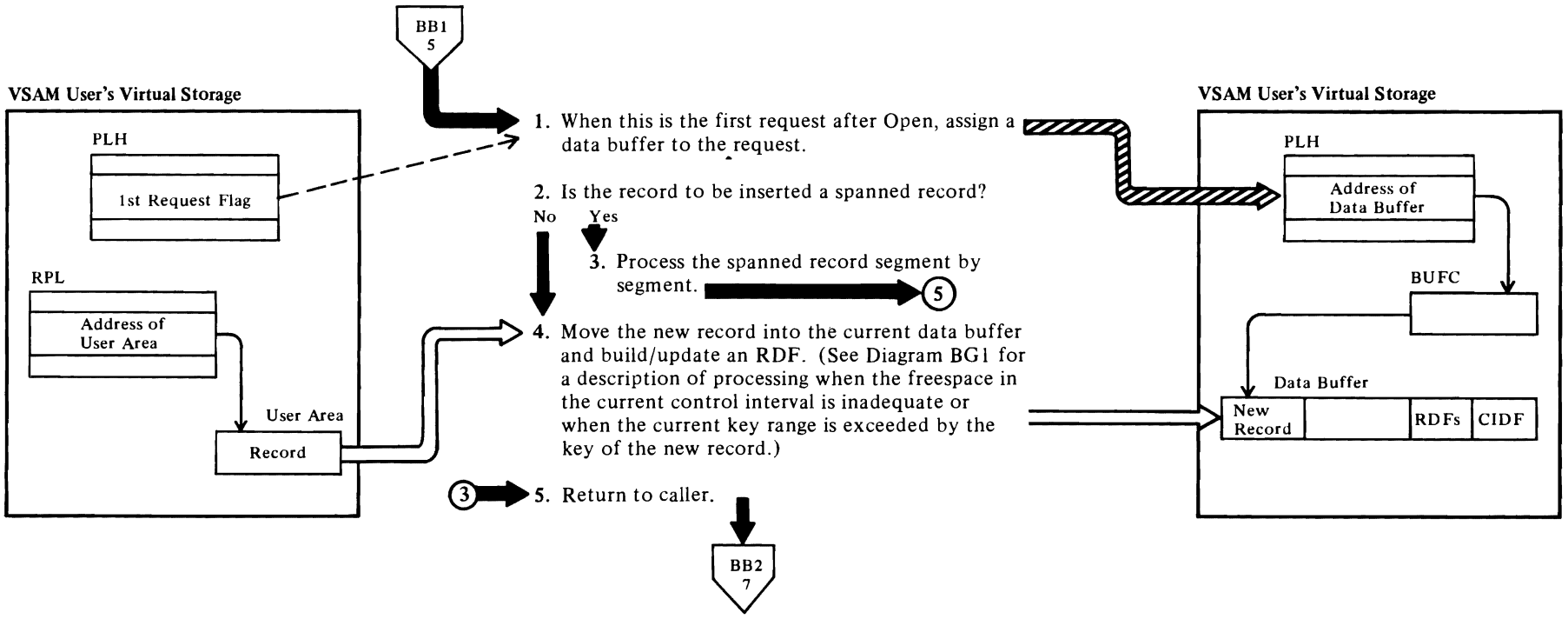
**7 IDA019RM**

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 BF1. PUT-Key-Sequenced Processing: Create



## Notes for Diagram BF1

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.

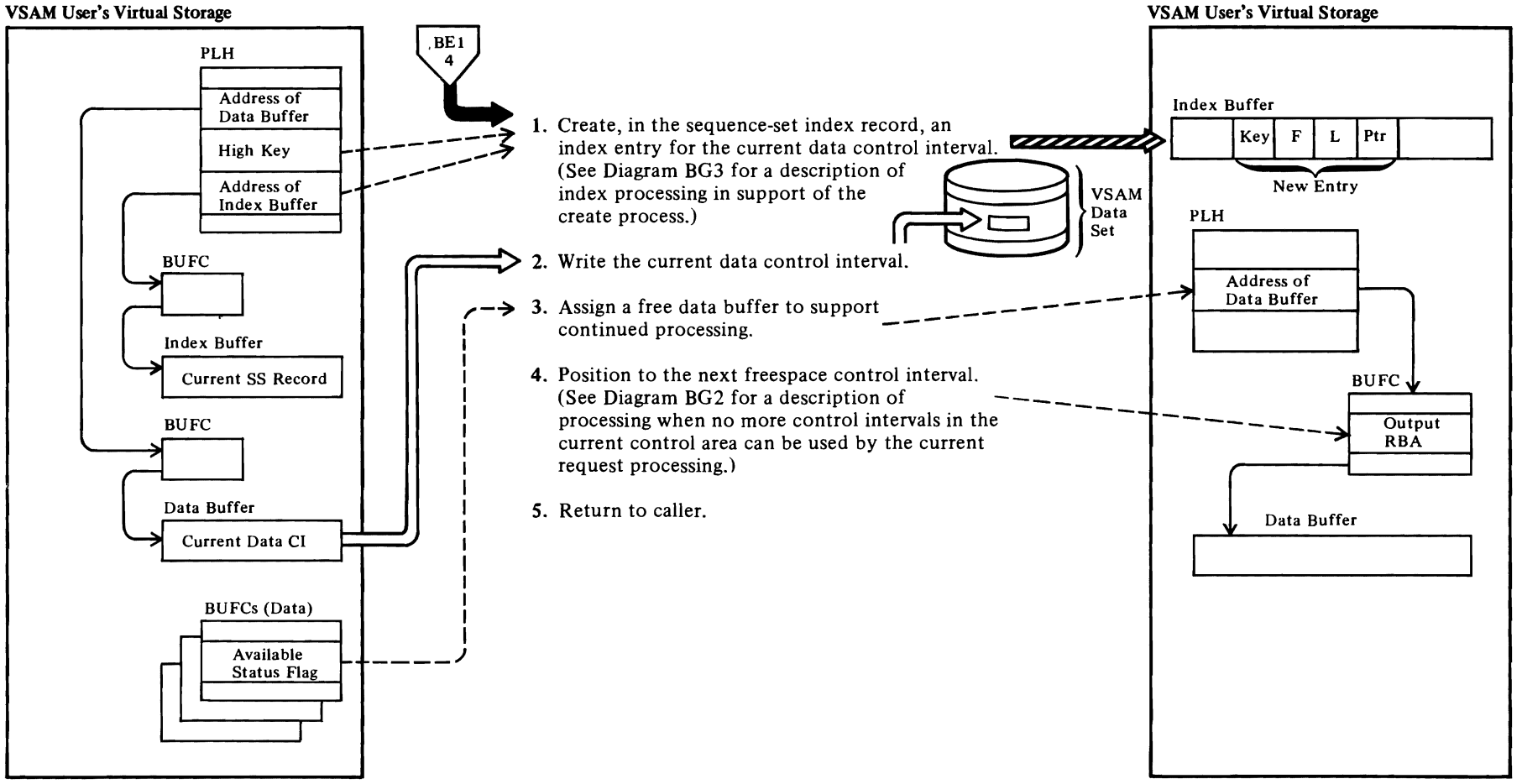
3 **IDA019RM** calls **IDA019RT**, which calls **IDA019SA**

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



## 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, a call is made to the I/O Manager, IDA019R3, 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.)

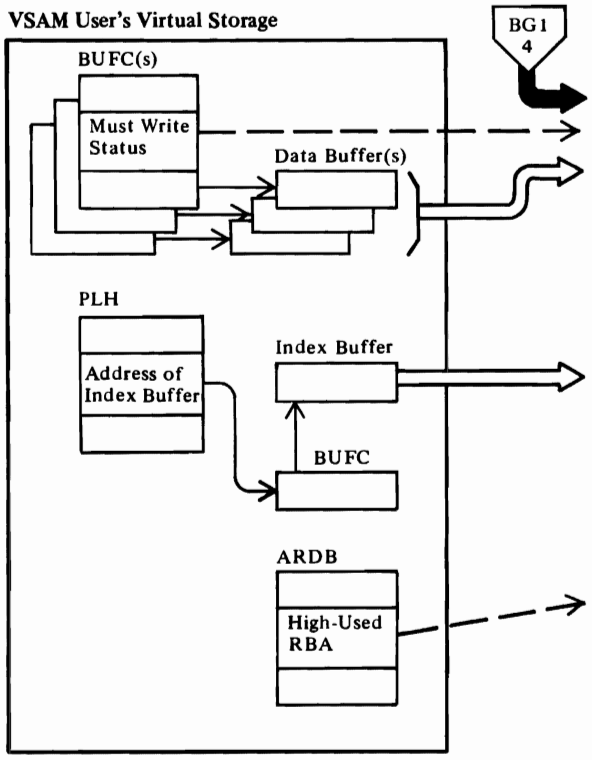
4 **IDA019SA**

**IDA019SA: EOCA**

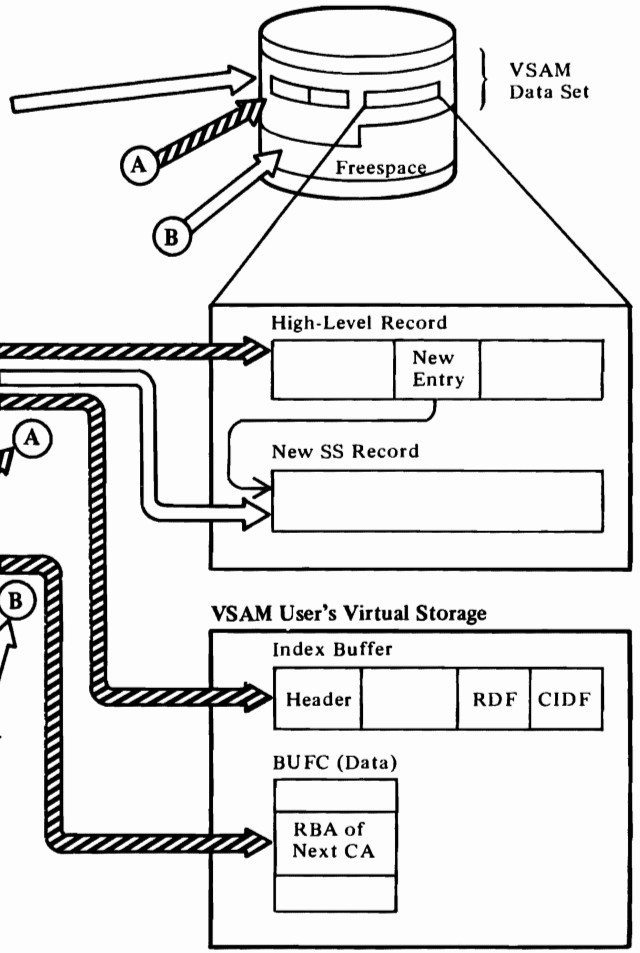
More control intervals cannot be added to the current control area if the key of the last record in the last data control interval equals the high key of the current or only key range or if there aren't enough freespace control intervals remaining in the control area to hold the new record and to maintain 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




1. Write all completed and unwritten data control intervals associated with the current control area.
2. 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. (See Diagram BG4.)
3. When the nonrecovery option is specified, preformat any unused control intervals in the current control area.
4. Establish the RBA of the next control area in physical sequence. (See Diagram AE1 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.)
5. When the recovery option is specified (SPEED=OFF), preformat the next control area.
6. Return to caller.



## Notes for Diagram BG2

1 **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.

2 **IDA019SA** calls **IDA019RG**

3 **IDA019SA** calls **IDA019RK**

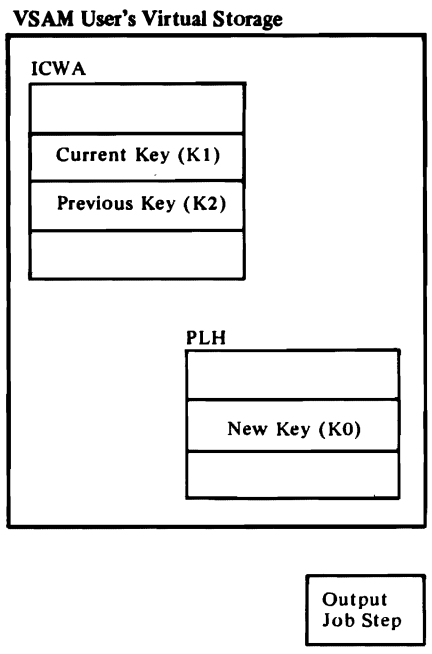
4 **IDA019SA**

**IDA019SA** calls **IDA019R5 (IDAEOVIF)**

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

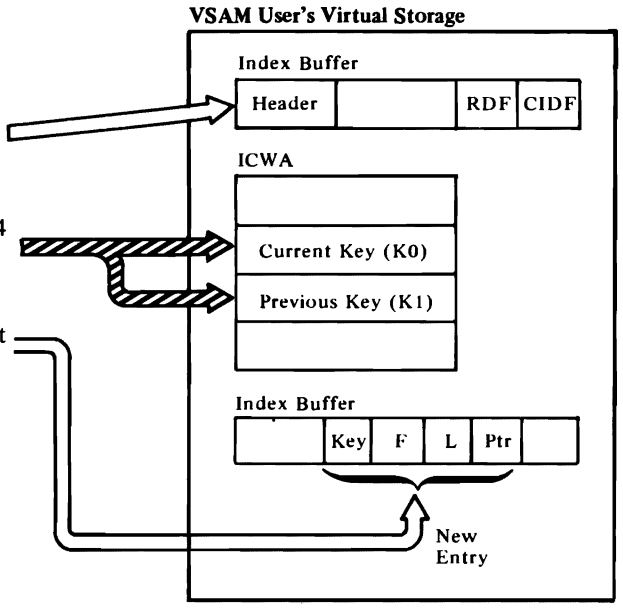
5 **IDA019SA** calls **IDA019RK**

# Diagram BG3. Creating a Key-Sequenced Data Set





1. When this is the first entry in the index, obtain a freespace control interval and initialize an index buffer.
2. Compress the new entry's key. (See Diagram BG4 for a description of processing when the index entry won't fit in the index record.)
3. Build a complete entry in the current sequence-set index buffer.
4. Return to caller.



## Notes for Diagram BG3

**1 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).

**2 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.)

**3 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.

**4 Normal (or Nonsection) Entry Processing**

**IDA019RG: IDAIST**

The current key is front compressed relative to the previous key. The front compression performed in step 2 is based on the assumption that the new entry is a section entry. Only the rear compression performed for step 2 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.

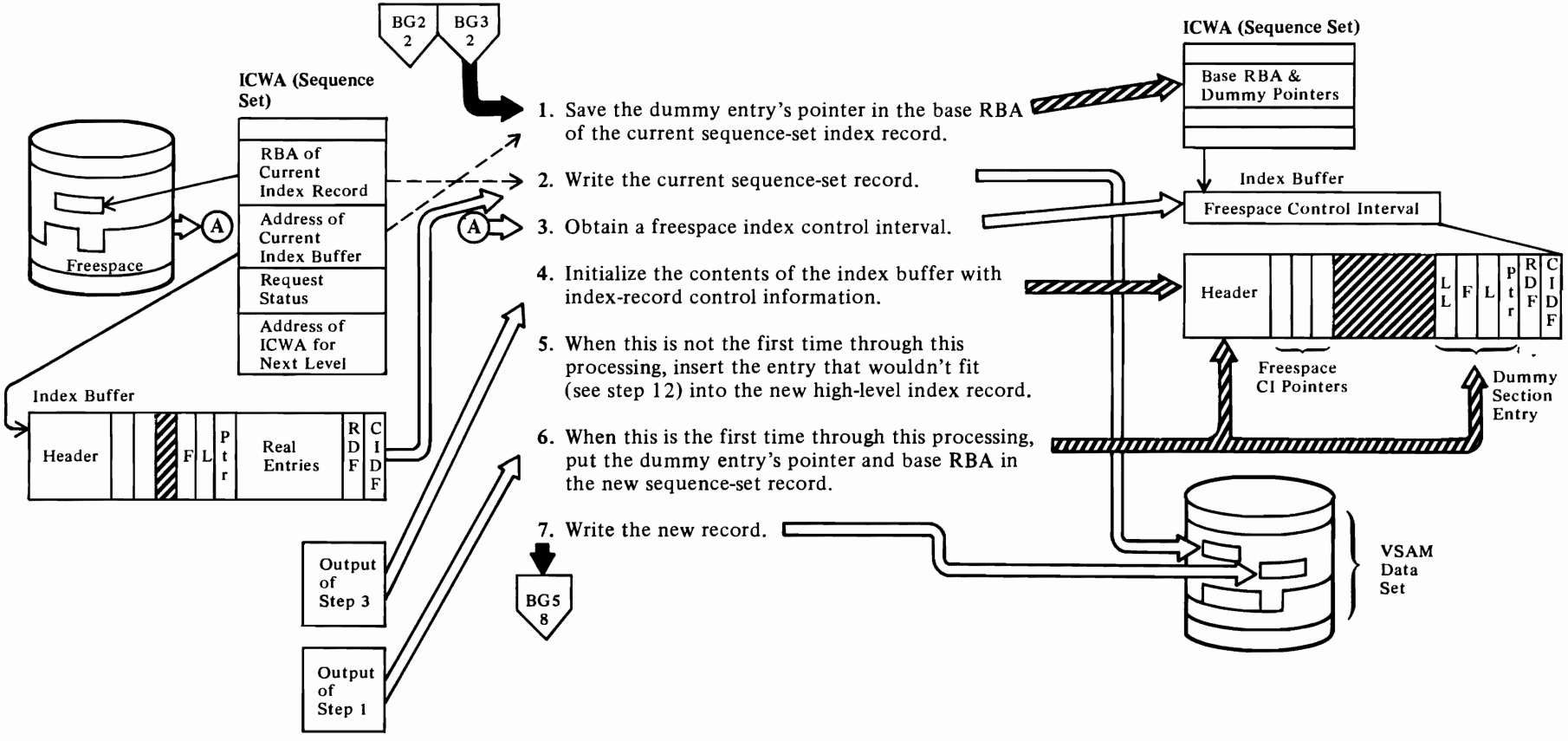
When a section entry has not been built, the section entry pointer in the index record header is advanced to point to the F field in the new dummy entry.

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 note 3, "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



## Notes for Diagram BG4

**1 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.

**2 IDA019RG:** calls **IDA019RJ (IDAWR)**

This operation overlays the index record that was generated by step 7 when this procedure was previously entered.

**3 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.

**4 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 5), a pointer to the lower-level index record just written (see note 7) is moved into the new higher-level index record as the dummy entry representing the highest key of the current level of the index.

**5**

Steps 3 through 13 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 4.

**6**

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 4), 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.

**7 IDA019RJ: IDAWR**

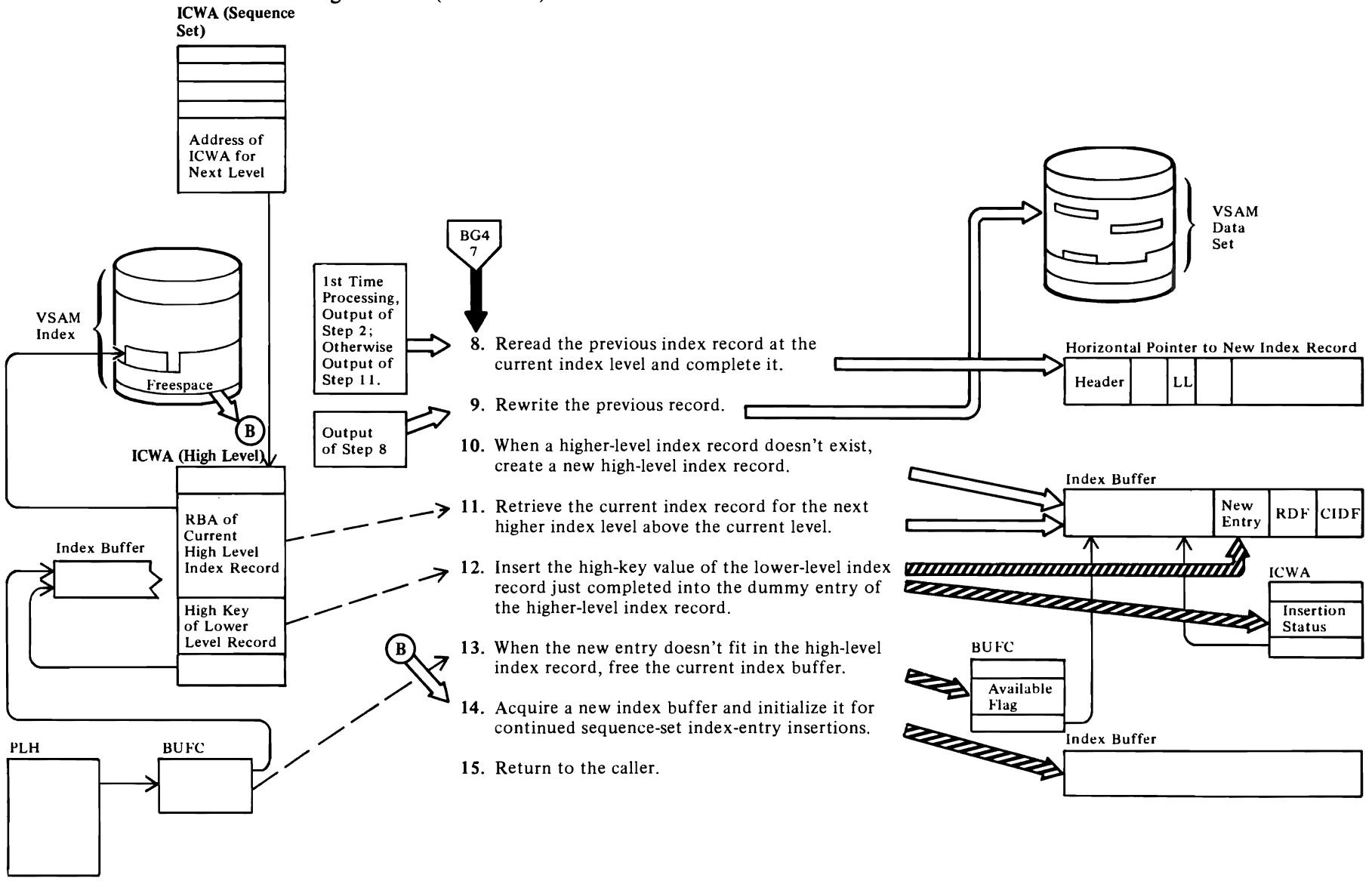
The new sequence-set or high-level index record is written to the data set.

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 2).

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)



## Notes for Diagram BG5

**8 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 6 and 8 (for high-level and sequence-set records, respectively) and because the horizontal pointer in the previous record makes the dummy entry accessible.

**9 IDA019RJ: IDAWR**

**10 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 8) is moved into the new higher-level index record as the dummy entry representing the highest key of the current level of the index.

**11 IDA019RJ: IDAR**

**12 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.

**13 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.

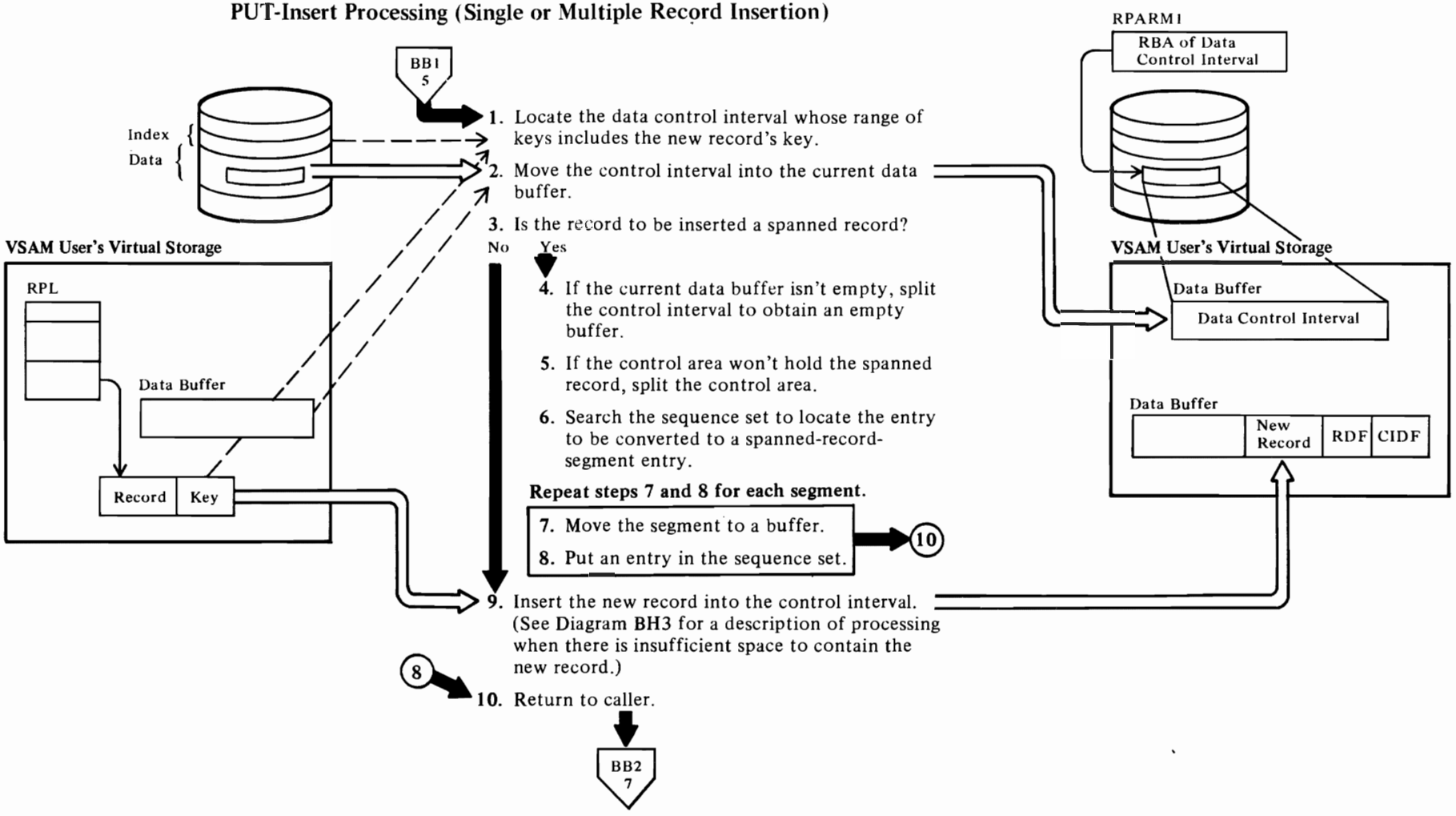
The processing of steps 3 through 13 is repeated until an index entry is successfully inserted in an index record on the index level above the level on which a new index record is created.

**14**

When this sequence-set record is completed and this routine is reentered, this record will be written at step 2, overlaying the dummy sequence-set record written at step 9.

# Diagram BH1. Modifying a Key-Sequenced Data Set

## PUT-Insert Processing (Single or Multiple Record Insertion)



## 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 BR1.)

**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 **IDA019RE**

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

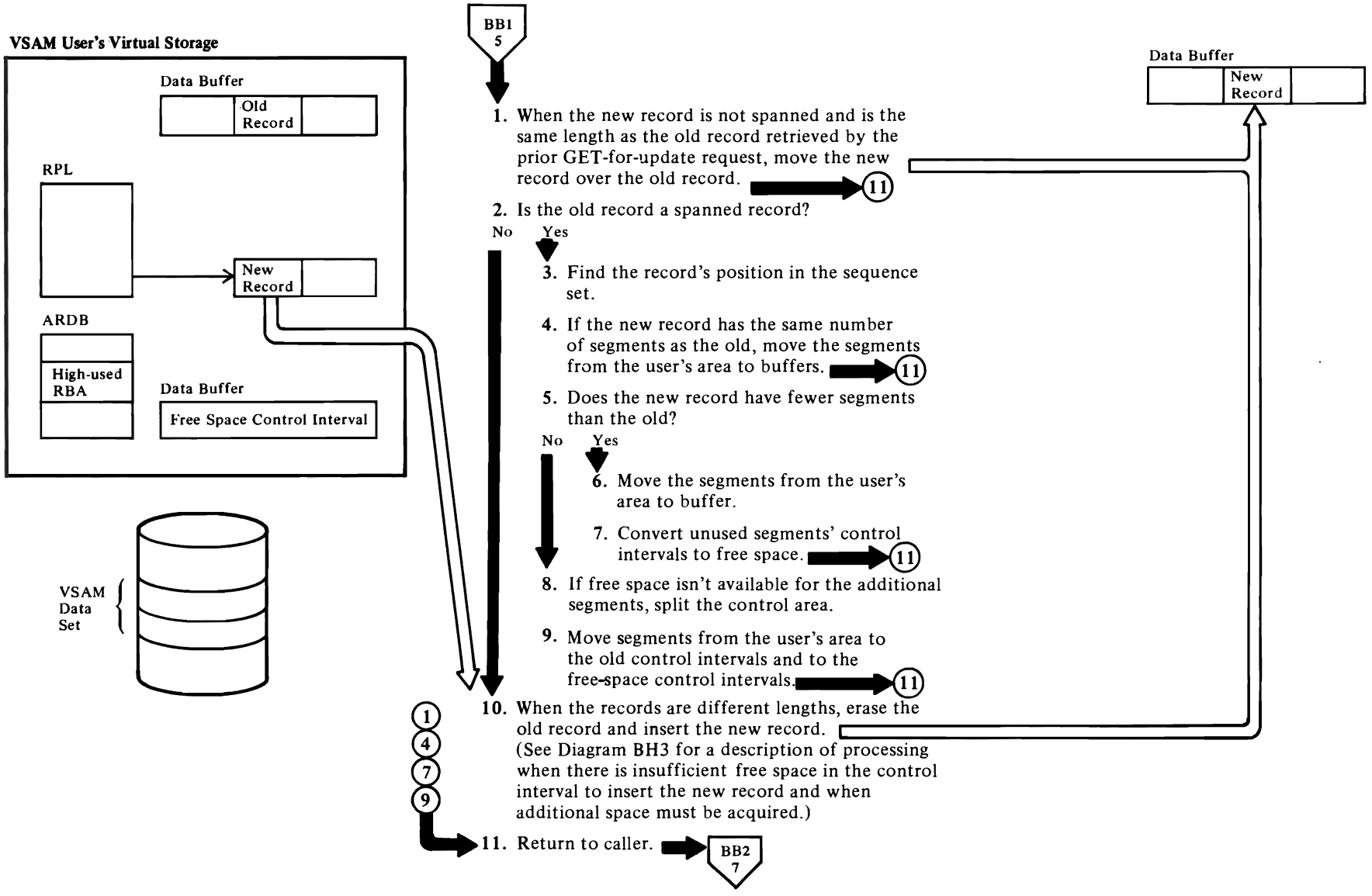
**6 IDA019RT** calls **IDA019RC**

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

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

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

# Diagram BH2. Modifying a Key-Sequenced Data Set



## Notes for Diagram BH2

**1 IDA019RL**

**2 IDA019RL** calls **IDA019RS**

Only if old record is a spanned record.

**3 IDA019RS** calls **IDA019RC**

**4 IDA019RS: IDAMVSEG**

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

**6** See note for step 4.

**7 IDA019RS: CLEARSEG**

An unused buffer is got 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.

**8 IDA019RS** calls **IDA019RF**

**9** See note for step 4

**IDA019RS: IDAADSEG**

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

**10 IDA019RL**

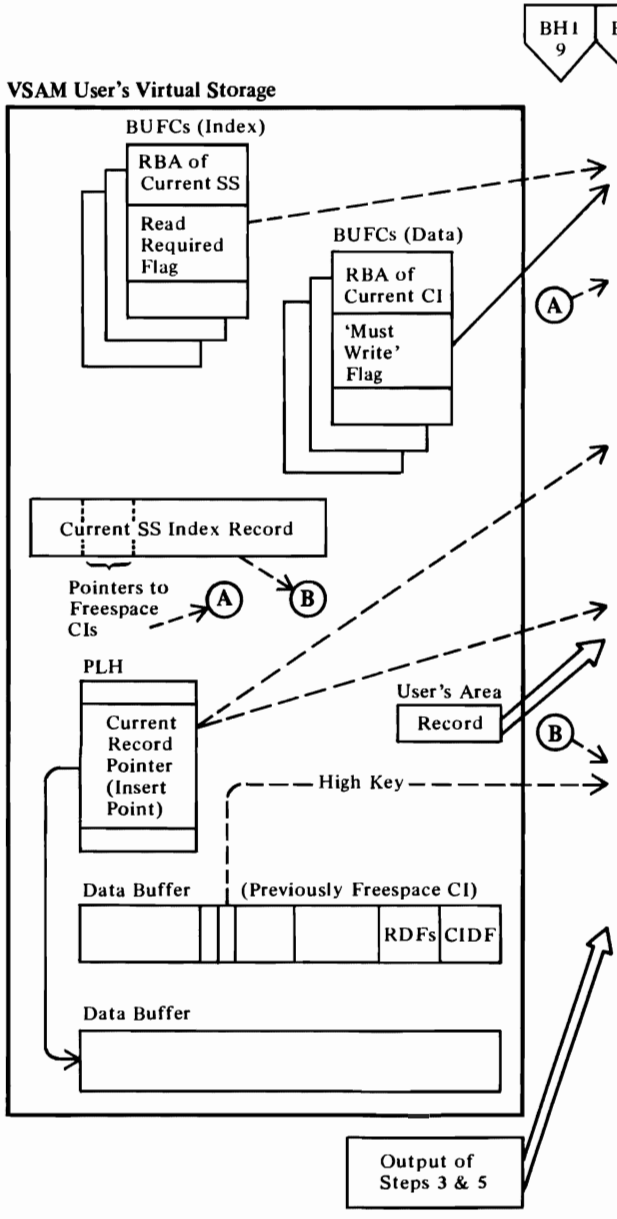
The old (unspanned) record is erased by overlaying it with records to its right. If the record is the last record in the control interval, it is cleared with zeros. IDA019RL then calls IDA019RM to insert the new (unspanned) record.

**IDA019RM**

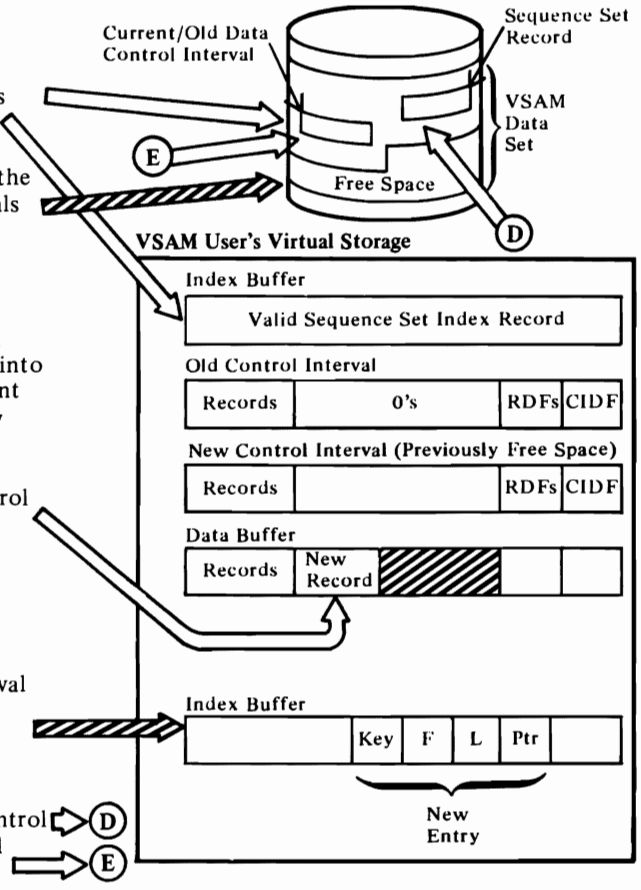
If an upgrade set exists, the alternate indexes in it are upgraded. (See Diagram BR1.)

# Diagram BH3. Modifying a Key-Sequenced Data Set

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




1. Ensure that the current sequence-set and the current data control interval reflect any changes made by other requests strings (if any).
2. 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. (See Diagram BH4 for a description of processing related to splitting a control area.)
3. When the current control interval must be split, move records from the current control interval into the next free-space control interval in the current control area and adjust RDFs to reflect the new distribution of records.
4. Insert the new record into the appropriate control interval.
5. Create an index entry for the new control interval in the sequence-set index record. (See Diagram BH6 for a description of index-entry insertion processing.)
6. Write the updated index record, and when a control control interval split has occurred, write the old data control interval.
7. When record insertion performed by step 4 is unsuccessful, repeat steps 2 through 6.
8. Return to caller.



## Notes for Diagram BH3

1 **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 (IDAWRBFR)**

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.

2 **IDA019RE** calls **IDA019RF**

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.

3

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.

4 **IDA019RE** calls **IDA019RM**

5 **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.

6 **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.

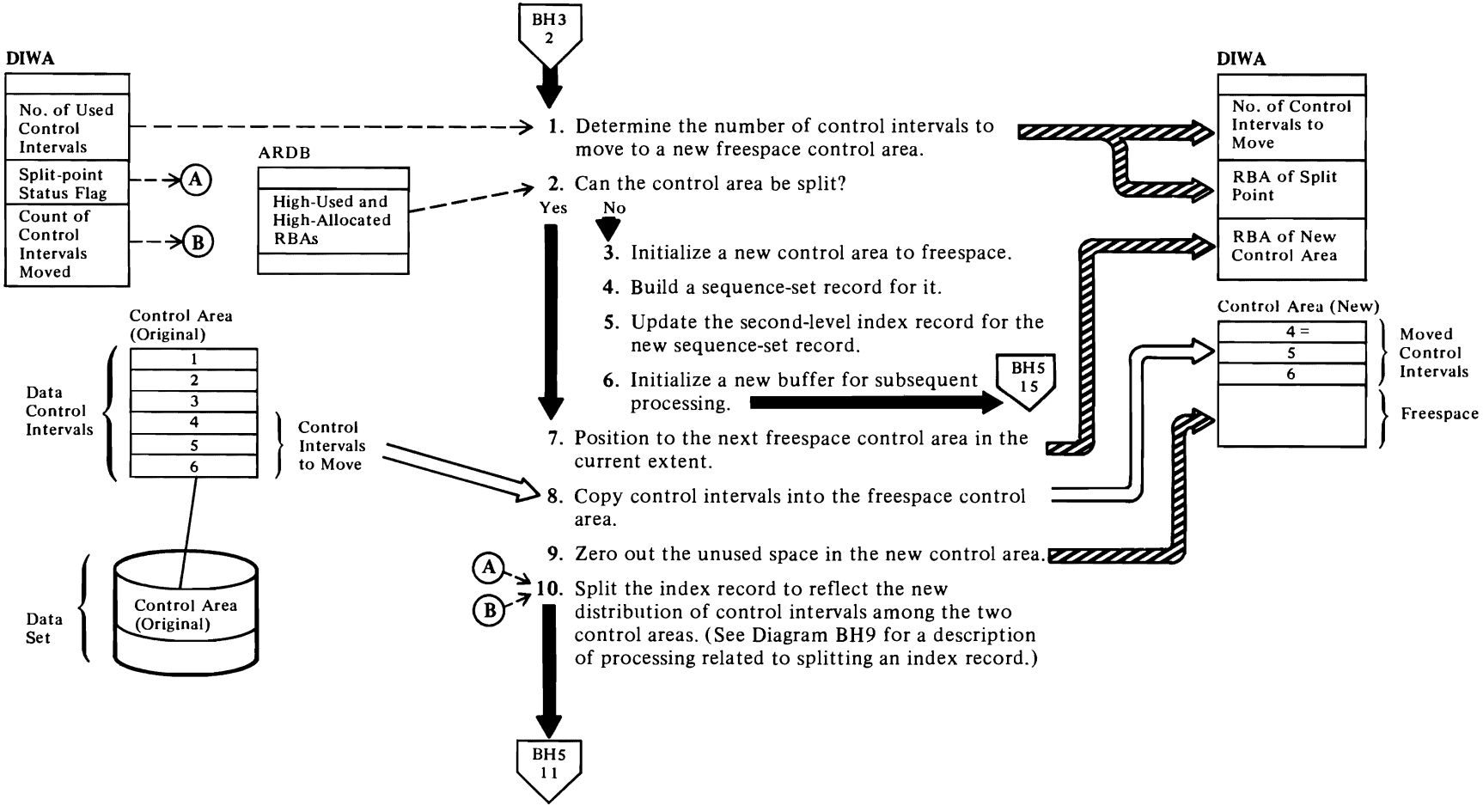
**IDA019RE** calls **IDA019RZ (IDAWRBFR)**

When a control interval split occurs (see note 3), the old data control interval associated with the current placeholder is written.

7

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



## 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 7 through 11 are the only steps performed for add-to-end and end-of-key-range processing.

**1**

**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.

**2** The control area can be split if it is filled with the segments of only one (unspanned) record.

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

The header of the index record is initialized.

User's key less than key of record in old control area:

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 key of record in old control area:

**IDA019SF** calls **IDA019RZ (IDAWRGFR)**

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

**IDA019SF** calls **IDA019RZ (IDAWRBFR)**

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

**5** **IDA019SF** calls **IDA019RZ (IDAHLINS)**

**6** **IDA019SF** calls **IDA019RZ (IDAGNNFL)**

**7** **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 AE1, 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.

**8** **IDA019RF** calls **IDA019RZ (IDAGRB)**

The first control interval is retrieved as a direct request.

**IDA019RF** calls **IDA019RZ (IDAGNXT)**

Subsequent control intervals are retreived 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)**

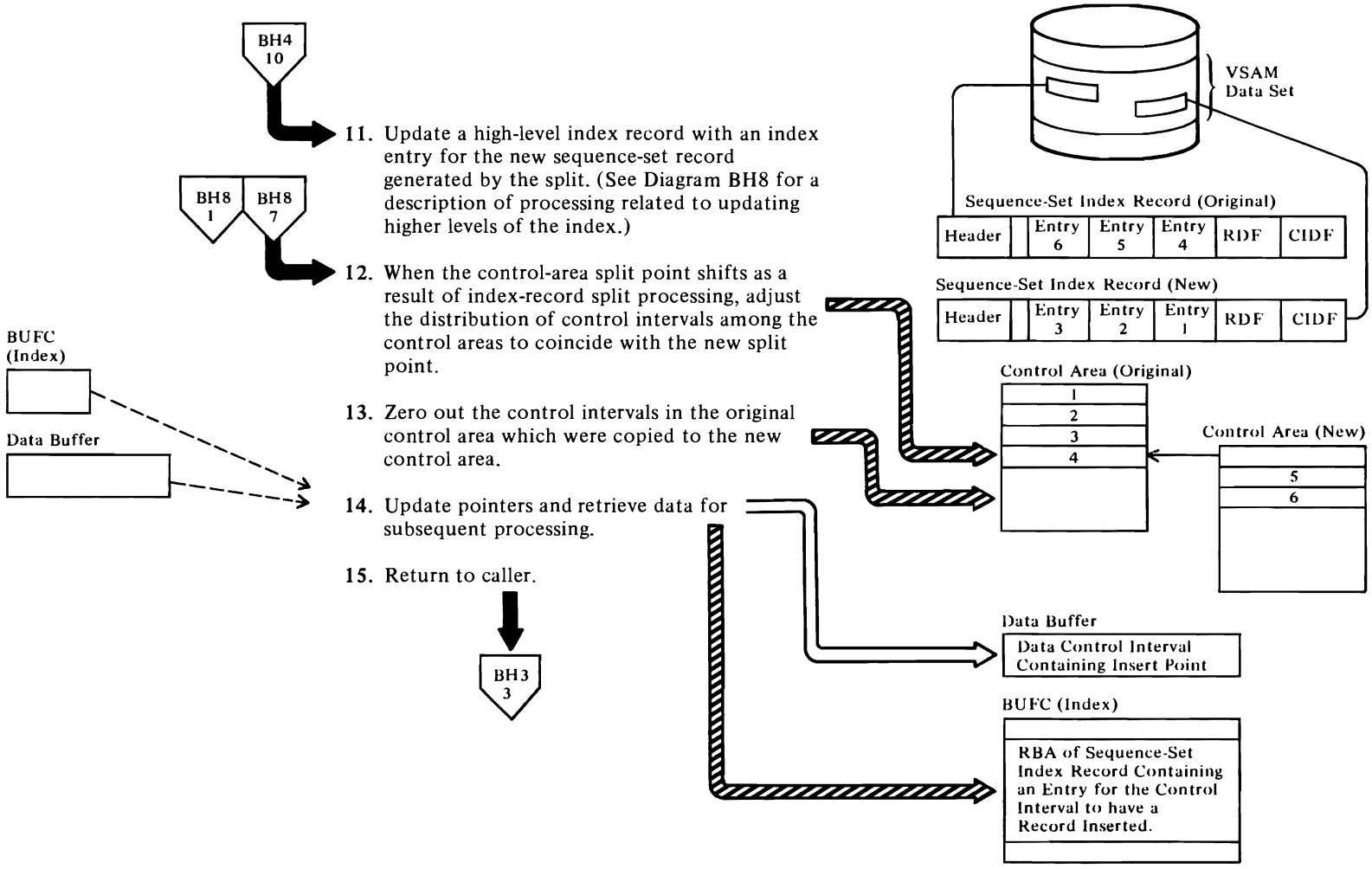
When all of the control intervals eligible for the move have been read into buffers, the buffers are written to the data set.

**9** **IDA019RF** calls **IDA019RK**

**10** **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



## Notes for Diagram BH5

**11 IDA019RI**

**12**

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.

**14 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**

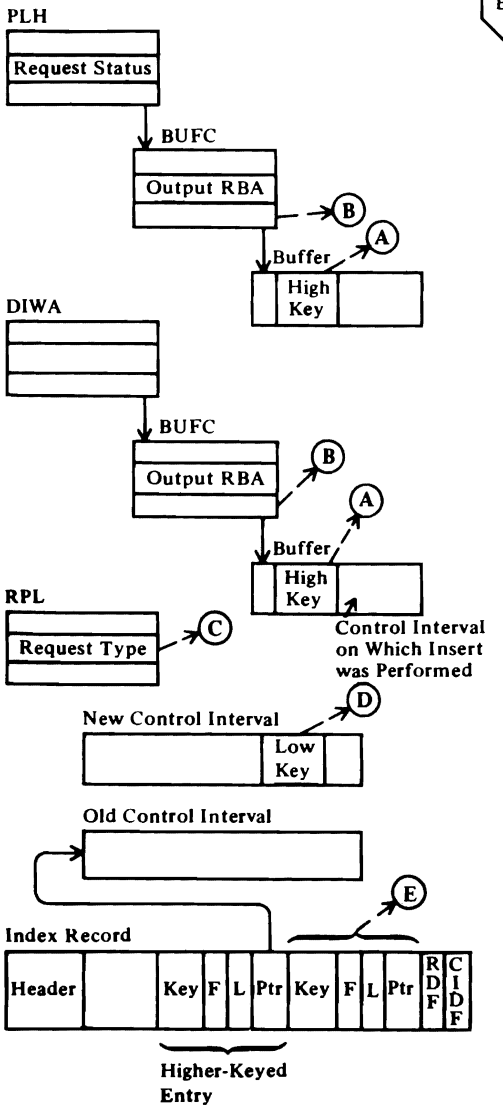
The buffers that were added to the placeholder's buffer chain to support the control-area-split process (see note 1 in Diagram BH4) are released from the chain.

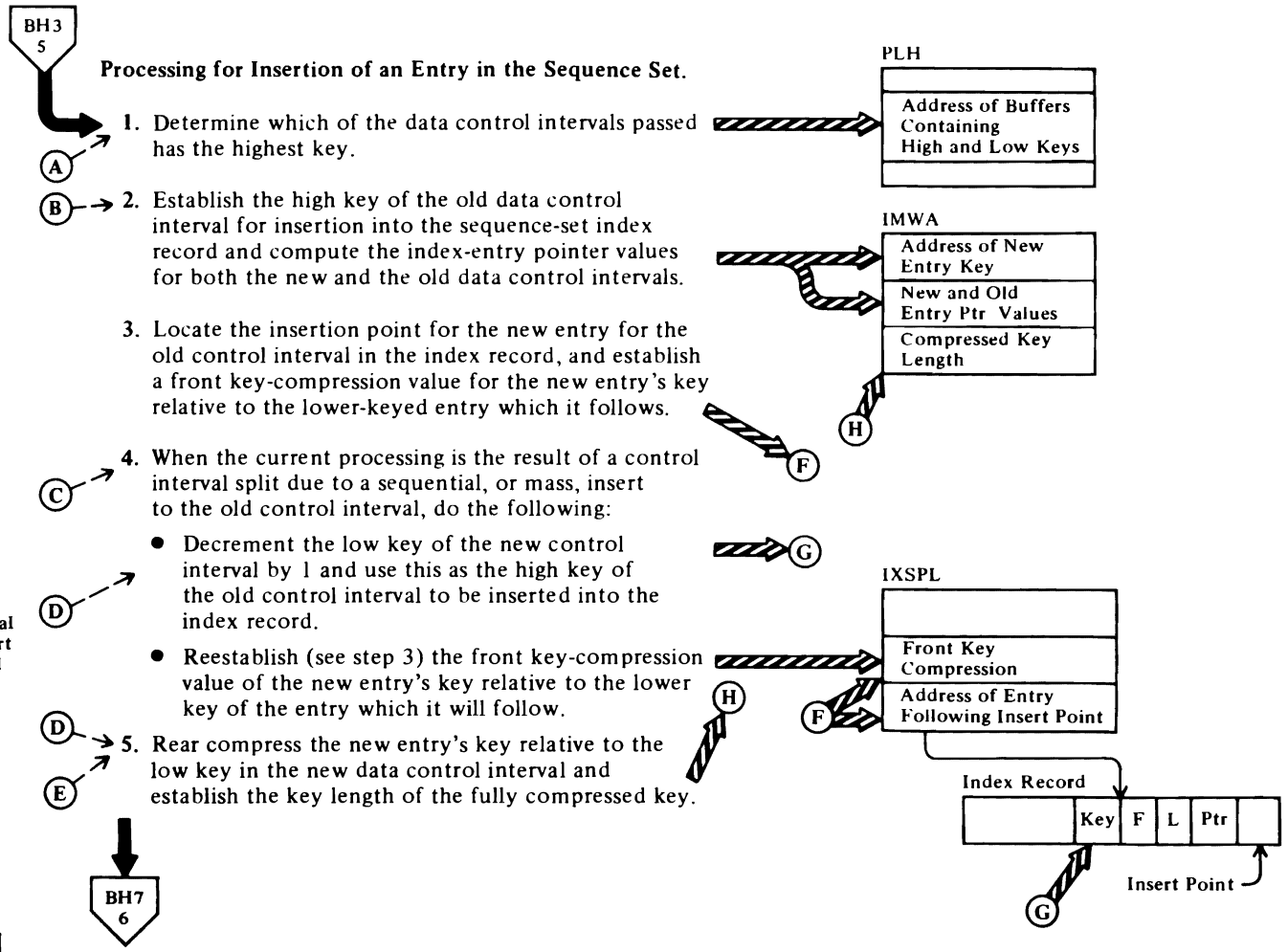
**IDA019RZ: IDAGRB**

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**



## Notes for Diagram BH6

**1 IDA019RH**

**2 IDA019RH**

**3 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.

**4**

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.

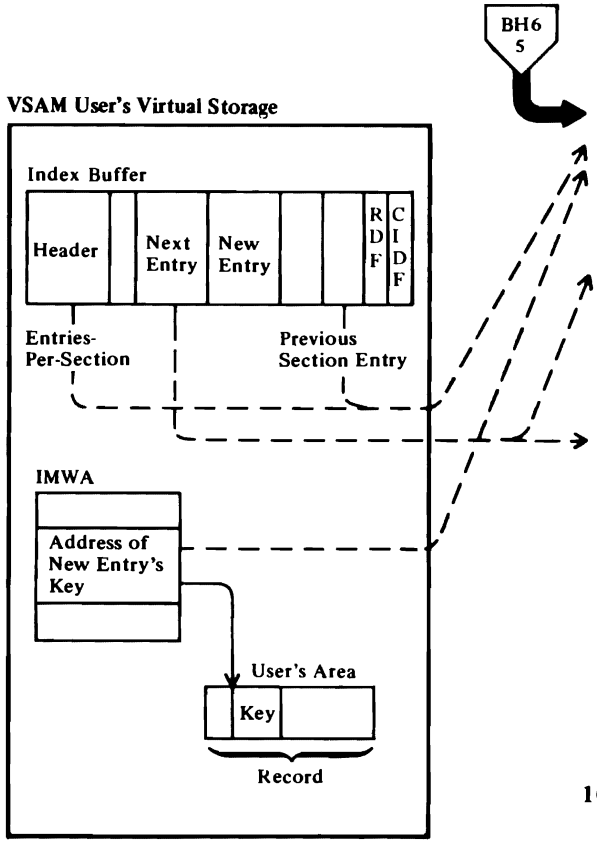
**5 IDA019RH: COMPRS**

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)**



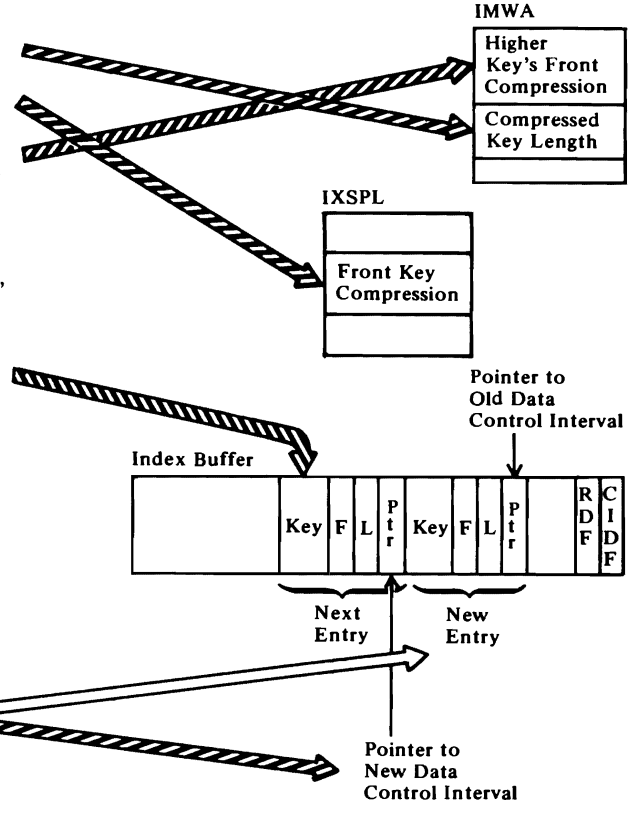

**Common Processing for High-Level and Sequence-Set Insertions**

6. 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 5 for sequence-set records.

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

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

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




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

11. Return to caller.

## Notes for Diagram BH7

**6 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.

**7 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.

**8 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 7. (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.

**9 IDA019RH**

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

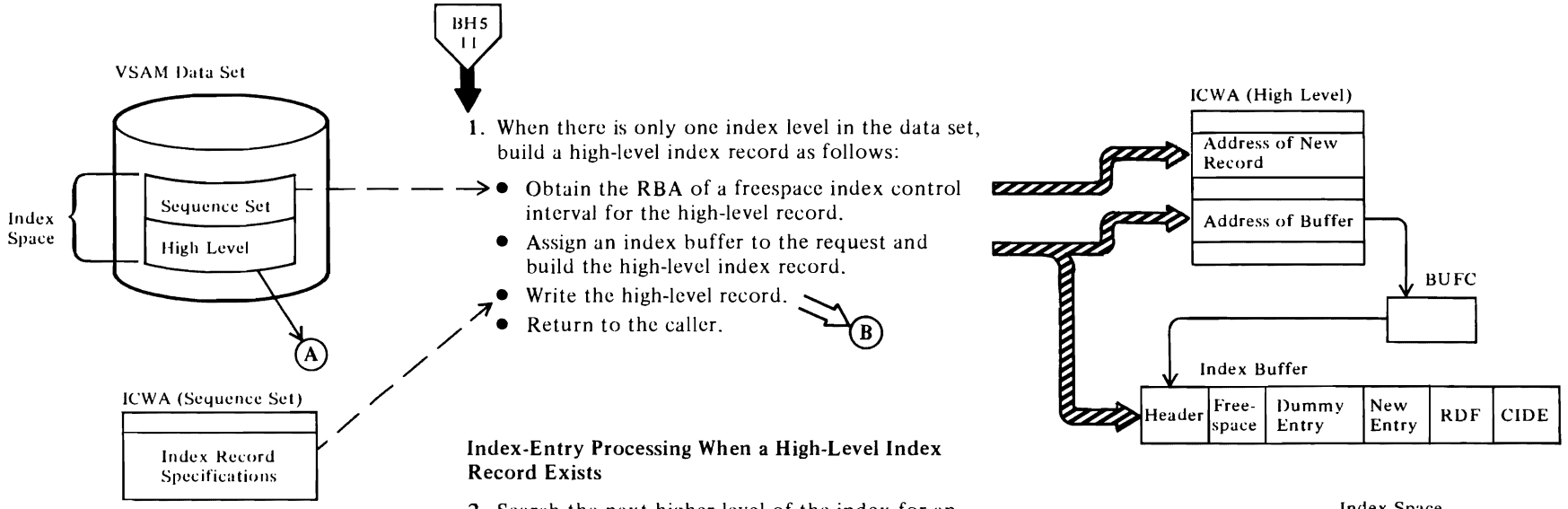
**10 IDA019RH**

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

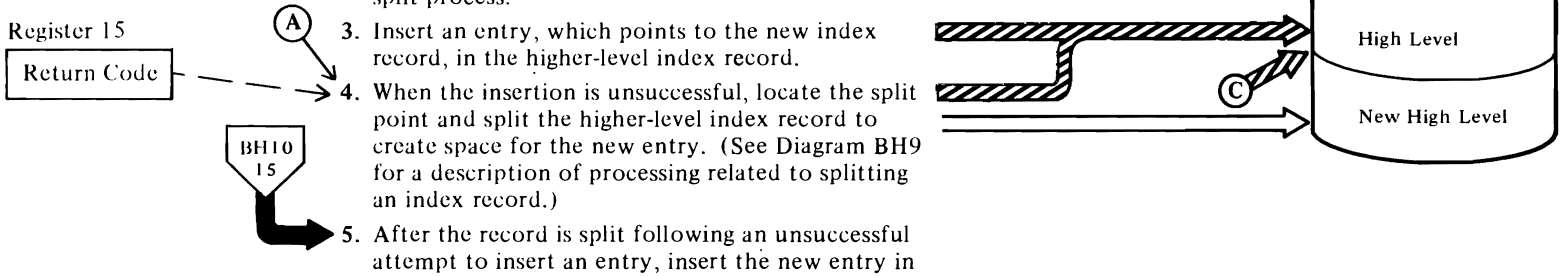
**Update a High Level of the Index with an Entry for the New Sequence-Set Record.**




1. 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**

2. 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.
3. Insert an entry, which points to the new index record, in the higher-level index record.
4. When the insertion is unsuccessful, locate the split point and split the higher-level index record to create space for the new entry. (See Diagram BH9 for a description of processing related to splitting an index record.)
5. 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 4).
6. Repeat steps 2 through 4 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.
7. Return to caller.



## Notes for Diagram BH8

**1** **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.

**2** **IDA019RI** calls **IDA019RB**

**3** **IDA019RI** calls **IDA019RH**

**4**

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.

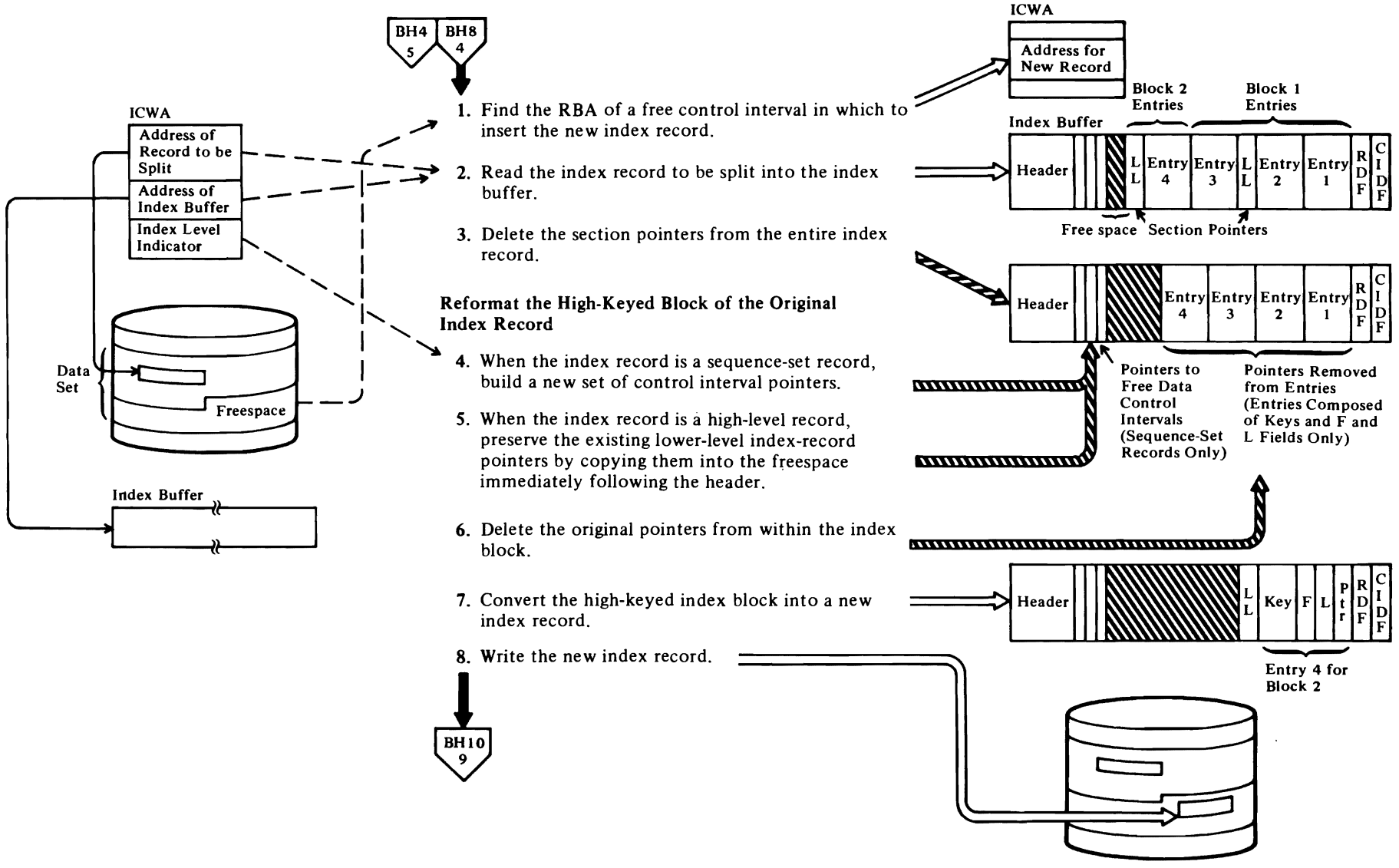
**IDA019RI** calls **IDA019RJ**

The index record is split to create space for the index entry associated with the new index record created by the split process.

**5** **IDA019RI** calls **IDA019RH**

# Diagram BH9. Modifying a Key-Sequenced Data Set

**Split an Index Record to Create Space for a New Index Entry**



## Notes for Diagram BH9

**1 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.

**2 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 1). Accordingly, the index record must be reread.

**3 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.

**4 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.

**5 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.

**6 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.

**7 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.

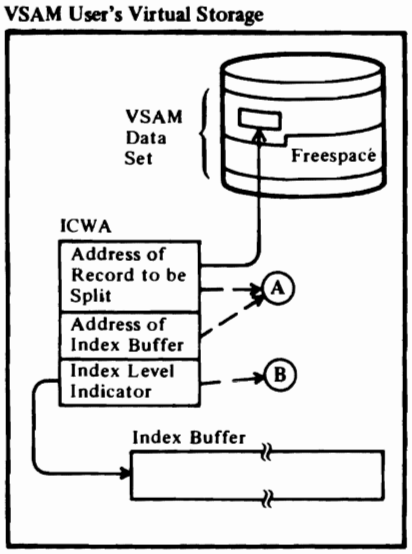
**g)** Steps 7d, e, and f are repeated until the test in step 7f is not satisfied.

**8 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)




BH9 8

**Reformat the Low-Keyed Block of the Original Index Record**

9. Reread the original index record into the index buffer.

10. Delete the section pointers from the entire index record.

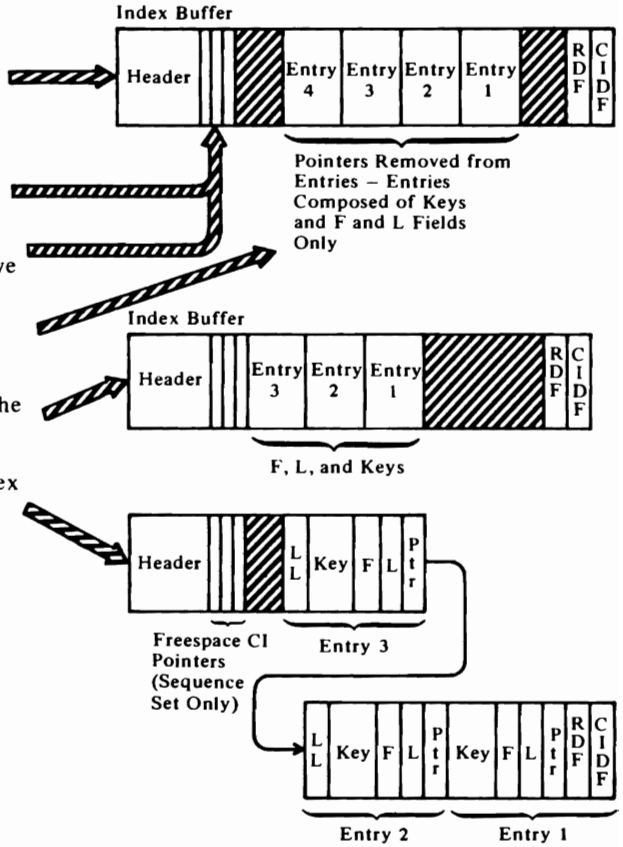
11. Compress the low-keyed index block as follows:
    - Count the entries to the left of the split point and preserve their pointers.
    - When the record is a sequence-set record, count entries to the right of the split point and preserve their pointers.
    - Delete the entry pointers in both blocks of the index record.

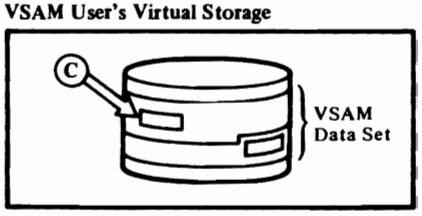
12. Move the entries to the right of the split point to the left to adjoin the pointers.

13. Convert the low-keyed index block into a new index record.

14. Rewrite the original index record.

15. Return to caller.



## Notes for Diagram BH10

**9 IDA019RJ: IDAR** (calls **IDA019RZ (IDAGRB)**)

The original index record is reread.

**10 IDA019RJ: DELSECT**

See note 3 of Diagram BH9.

**11 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: MOVEPTRI**

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 6.

**12**

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.

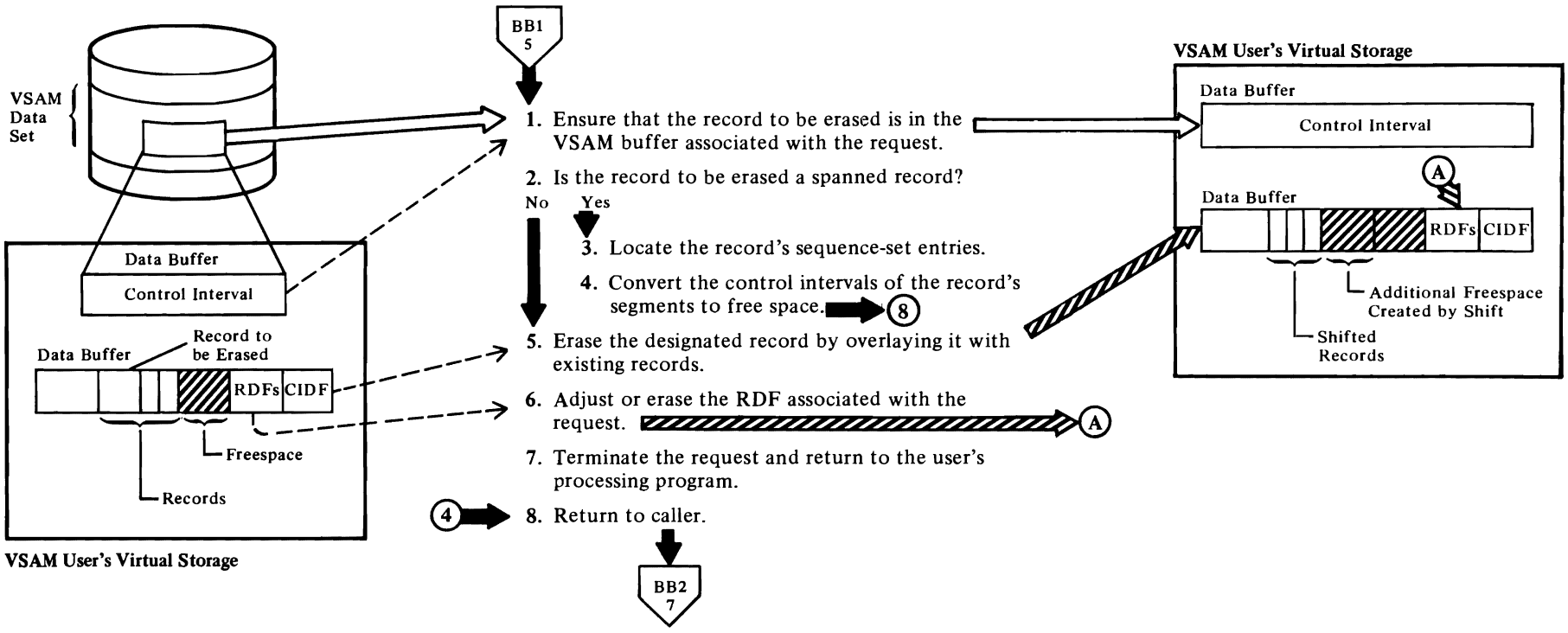
**13 IDA019RJ: BUILDREC**

See note 7.

**14 IDA019RJ: IDAWR**

The index buffer containing the revised index record is written to the data set, overlaying the original index record.

# Diagram BI1. ERASE Processing: Key-Sequenced



## Notes for Diagram BI1

**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, upgrade the alternate indexes in it. (See Diagram BR1.)

**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.

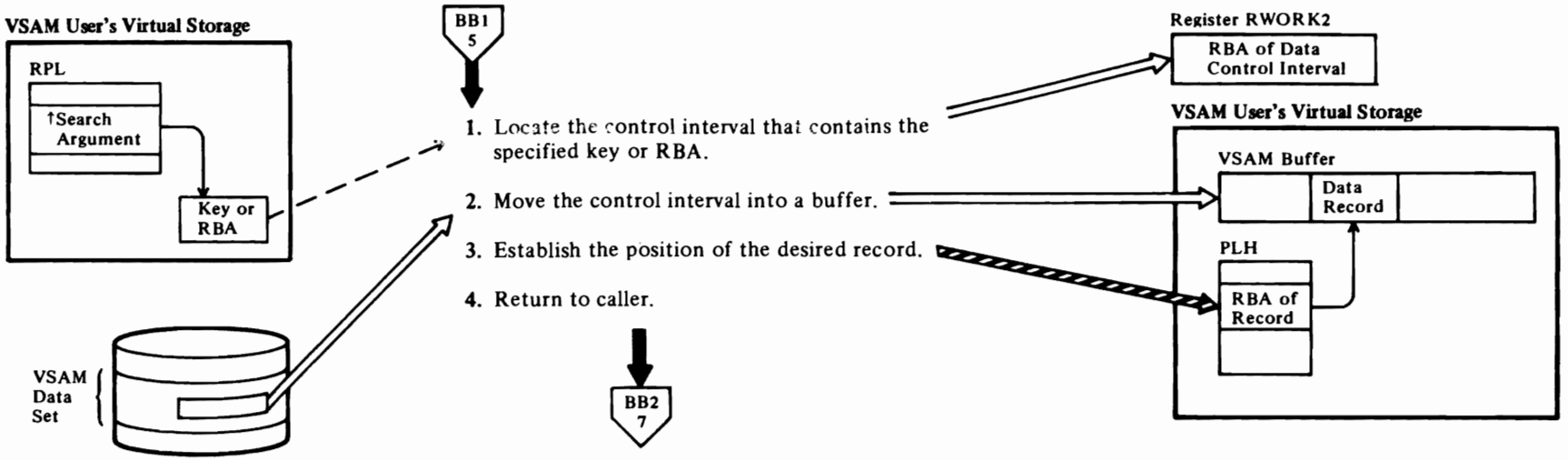
**5 IDA019RL**

**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 BJ1. POINT Processing



## Notes for Diagram BJ1

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 into the RBA of the control interval that contains the record and the offset of the record in the control interval.

**IDA019RR** calls **IDARRDRL**

The control interval is read, unless its RBA falls beyond the end of the data set. If the RBA isn't 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:**

**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.

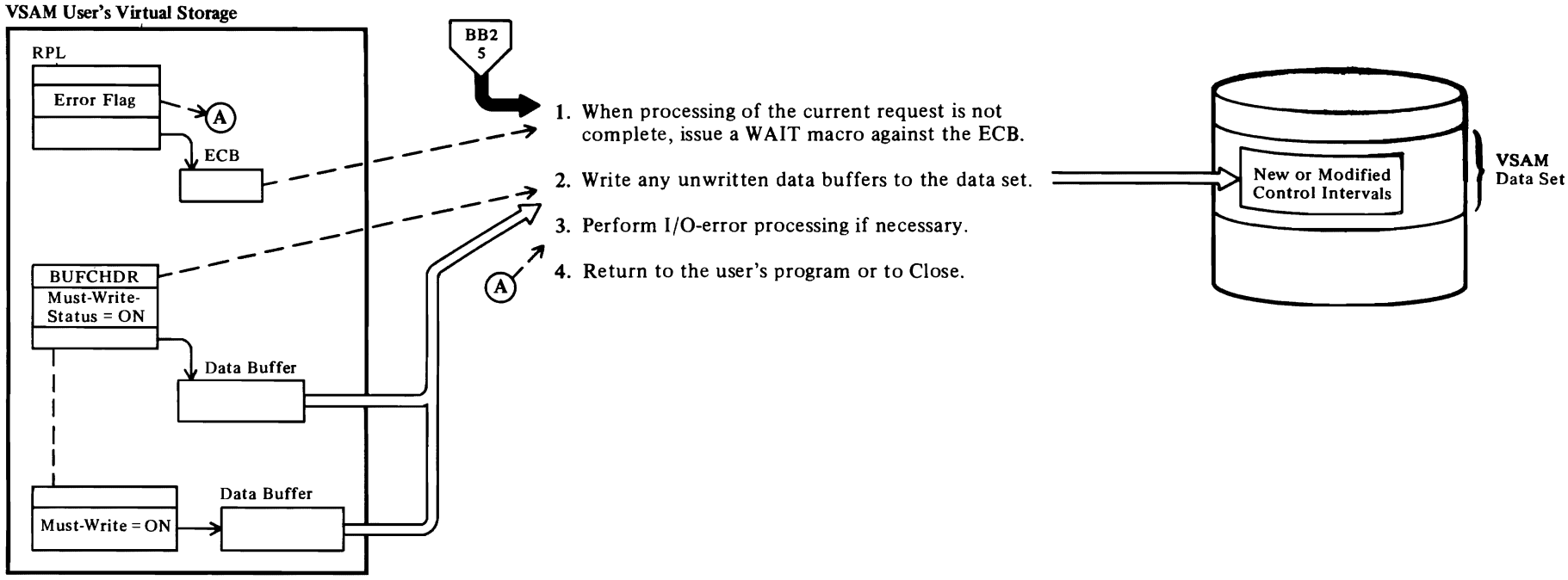
**Relative Record Processing:**

**IDARRDRL**

Positioning is established by saving in the PLH pointers to the record and the RDF and the RBA of the control interval.

# Diagram BK1. ENDREQ: Terminate A Record-Processing Request

**Noncreate**



## 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 block (IOB) in error is searched for a BUFC with an error indicator.

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)**)

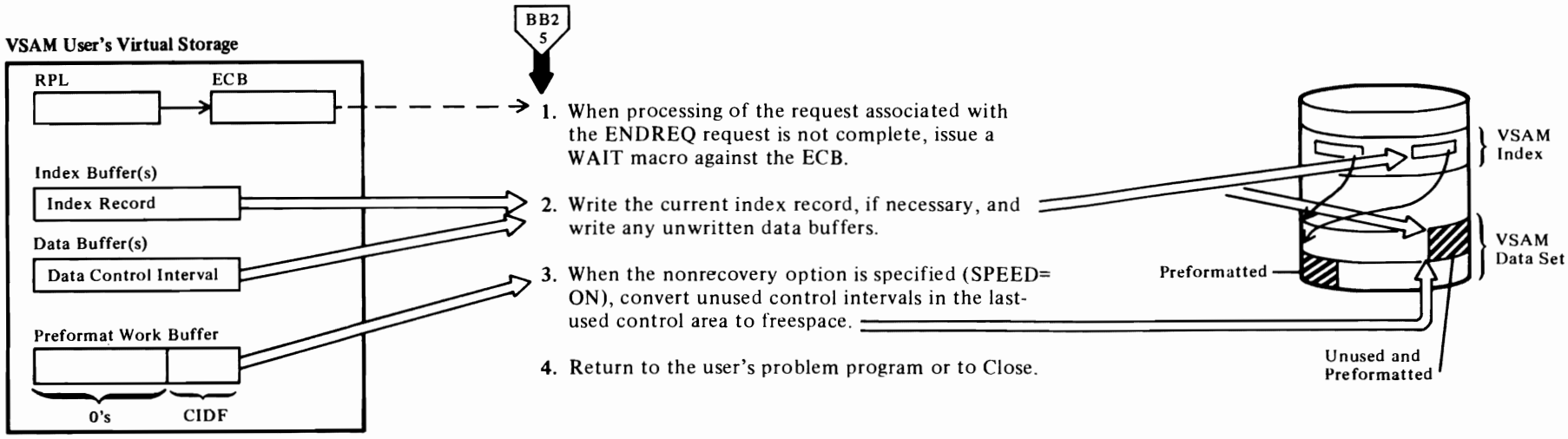
Excess data buffers are released from the current placeholder.

**IDA019RP: IDAENDRQ**

The placeholder is released from the current request string.

# Diagram BK2. ENDREQ: Terminate A Record-Processing Request

**Create**



## Notes for Diagram BK2

**1 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.

**2** The processing for step 2 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**

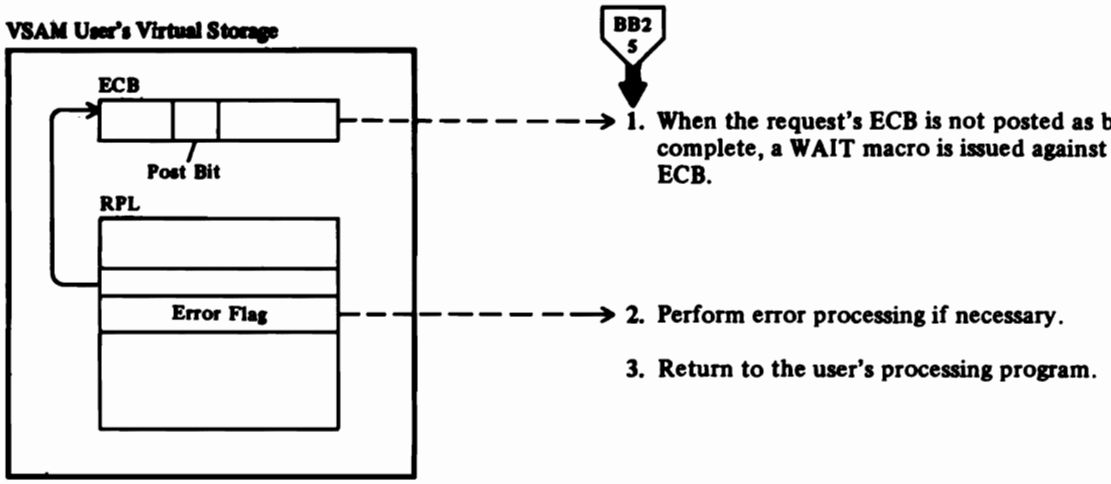
Before writing the index buffer, the following processing is performed: IDA019RG checks the leftmost entry, a dummy entry for the current control interval, in the index record to determine whether a maximum length key will fit in the remaining index record freespace. If there is adequate space to insert 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.

**3 IDA019RP** calls **IDA019RZ (IDAWRBFR)**

**4 IDA019RP** calls **IDA019RK**

# Diagram BL1. CHECK Processing



## Notes for Diagram BL1

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 instruction 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 **IDA019RS (IDAEXITR**

For processing if a SYNAD routine exists.

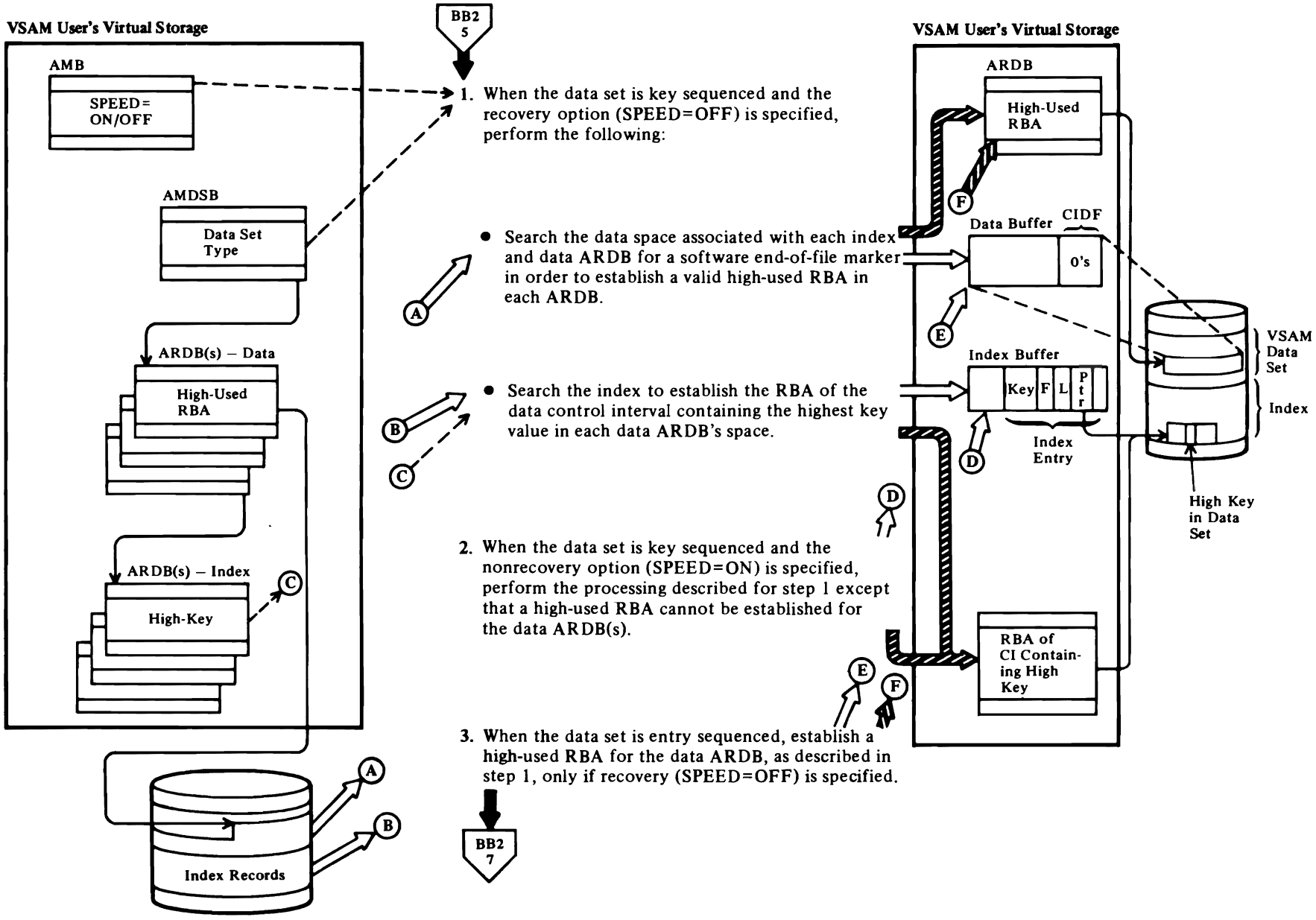
3 **IDA019R1: R1CHECK**

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 BM1. VERIFY Processing



## Notes for Diagram BM1

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. (IDA019RB then calls IDA019RZ (IDAGRB) to retrieve another index record and the search process repeats itself.)

**IDA019RO**

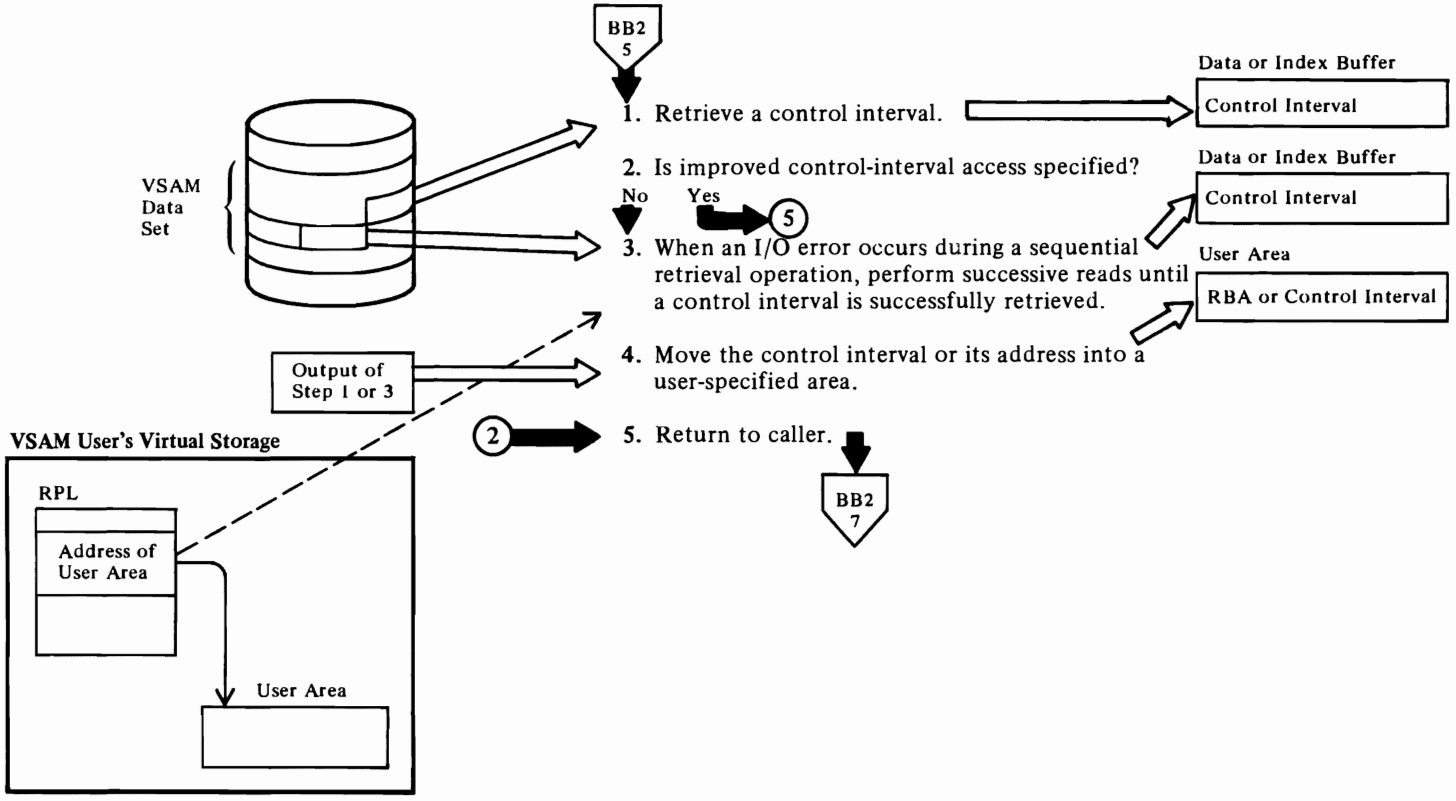
When the search is successful, the pointer in the index entry is converted into a valid RBA and moved into the ARDB.

2 See note 1.

3 See note 1.

# Diagram BN1. Processing by Control Interval

## GET or GETIX Processing (Control Interval Retrieval)



## 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 Request Processing (GET) 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).

2 **Improved Control-Interval Processing (ICI:**

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)**

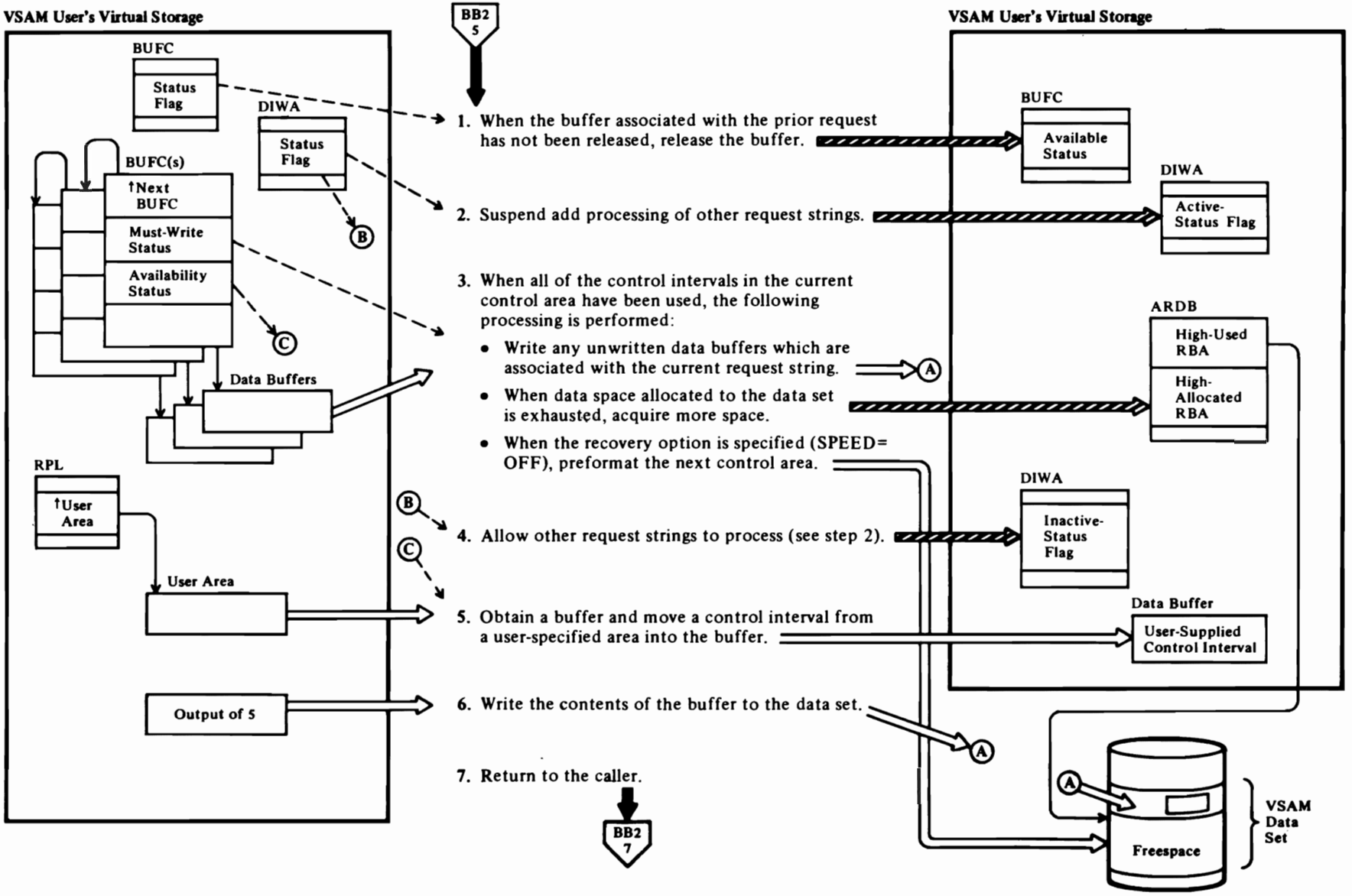
Journaling is performed when a journal exit routine exists.

**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)



## Notes for Diagram BN2

1 **IDA019R8** calls **IDA019RZ (IDAFREEB)**

2 **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.

3 **IDA019R8** calls **IDA019RZ (IDASBF)**

**IDA019R8** calls **IDA019R5 (IDAEOVIF)**

**IDA019R8** calls **IDA019RK**

4 **IDA019R8**

See note 2.

5 **IDA019R8** calls **IDA019RZ (IDAGNNFL)**

An available buffer is assigned to the request and written if necessary.

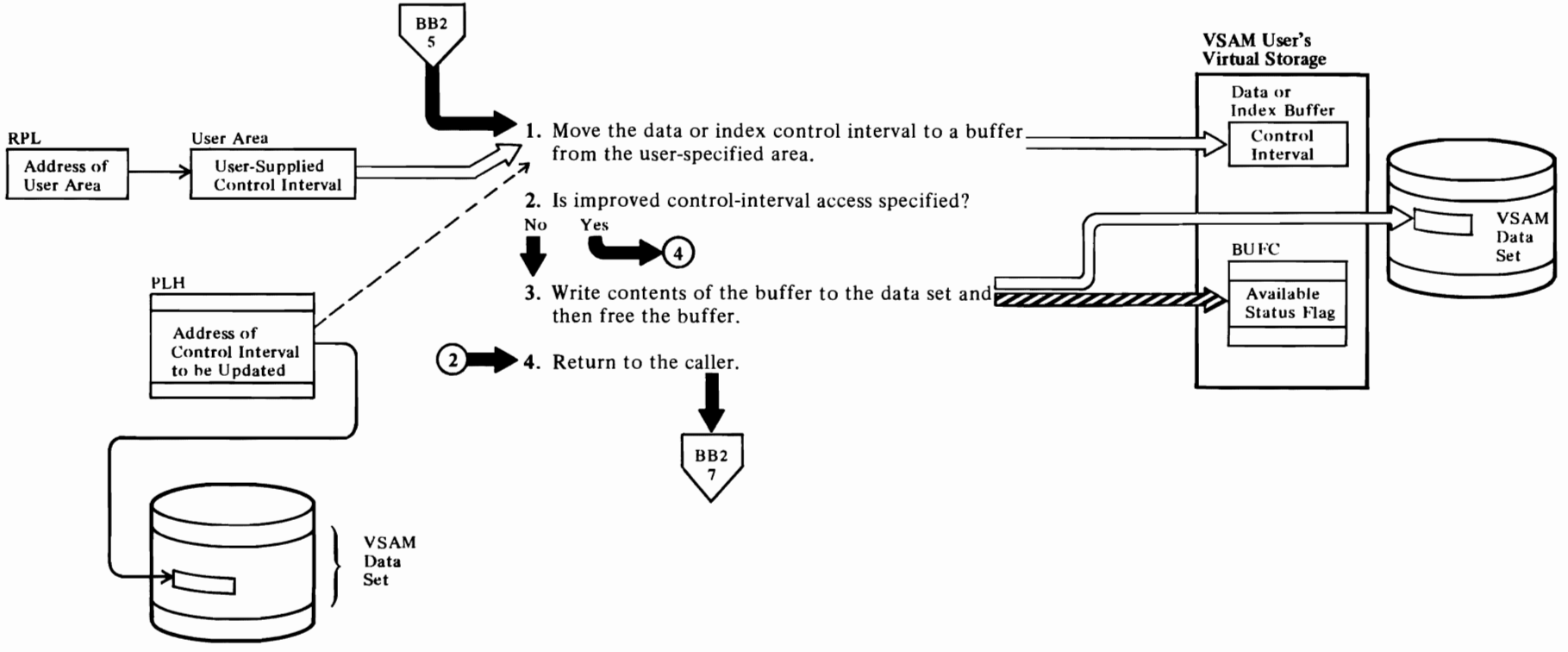
**IDA019R8**

The user-specified control interval is moved into the buffer.

6 **IDA019R8** calls **IDA019RZ (IDAWRBFR)**

# Diagram BN3. Processing by Control Interval

PUT or PUTIX-Update Processing (Update a Control Interval)



## Notes to Diagram BN3

1 **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): IDA019SI**

The request is decoded. A placeholder is obtained.

**IDA019S1** calls **IDA01953**

The control interval specified by the RPL is written.

3 **IDA019R8** calls **IDA019RP (IDATJXIT)**

Before writing the new control interval, journaling is performed if a journal exit routine exists.

**IDA019R8** calls **IDA019RZ (IDAWRBFR)**

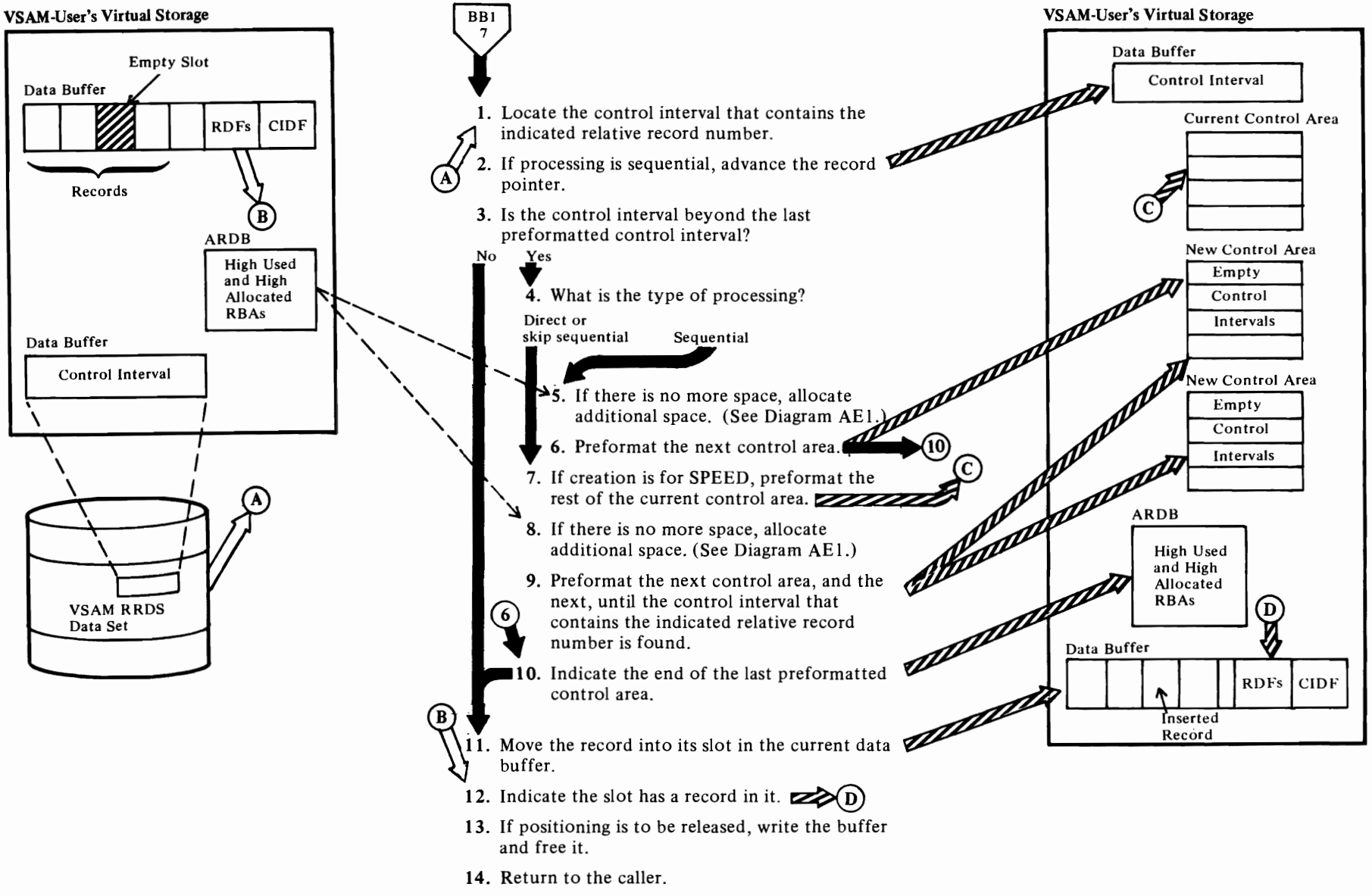
The new control interval is written to the data set.

**IDA019R8** calls **IDA019RZ (IDAFREEB)**

The buffer is released.

# Diagram BO1. Creating or Modifying a Relative Record Data Set

## PUT-Insert Processing



## 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 IDA019RO** calls **IDA019RM (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 AE1.)

**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.

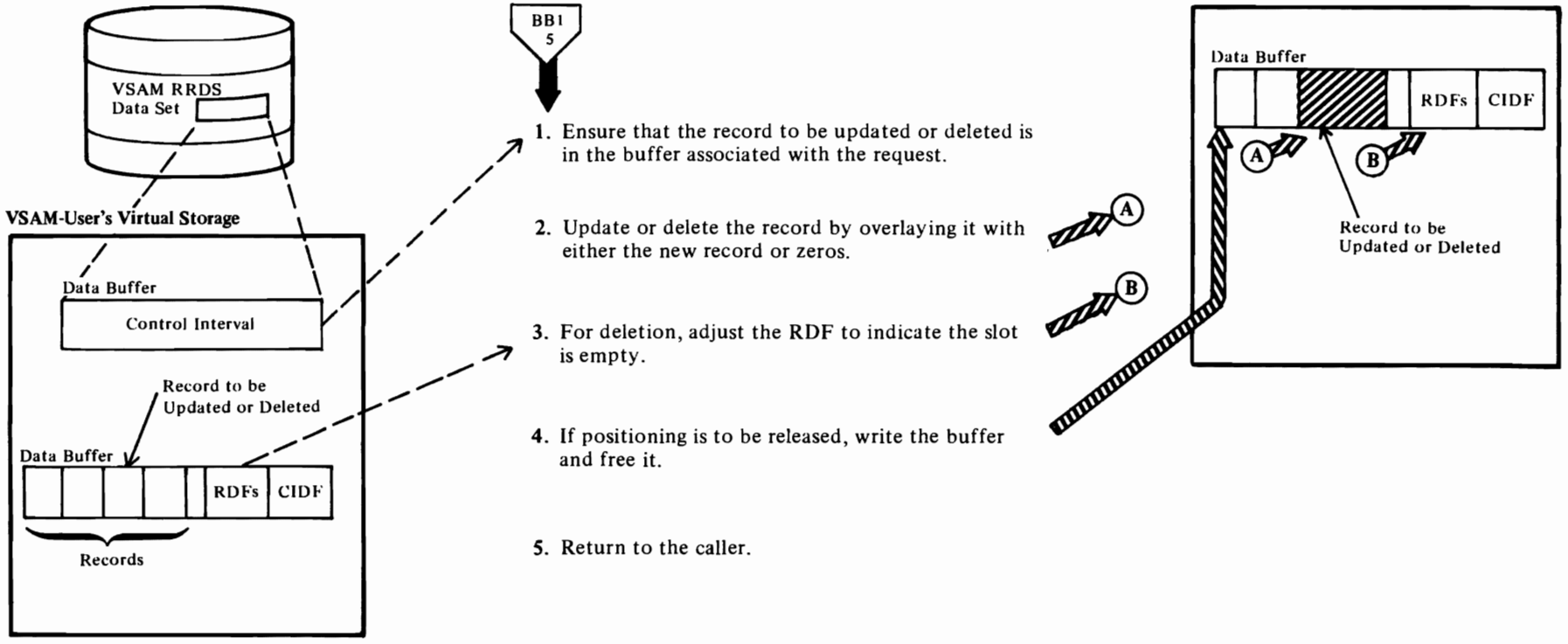
**11 IDA019RQ**

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 PUT-Update or Erase Processing



## Notes for Diagram BO2

1 **IDA019RQ**

A PUT-update or ERASE request must be preceded by a GET-update request.

2 **IDA019RQ**

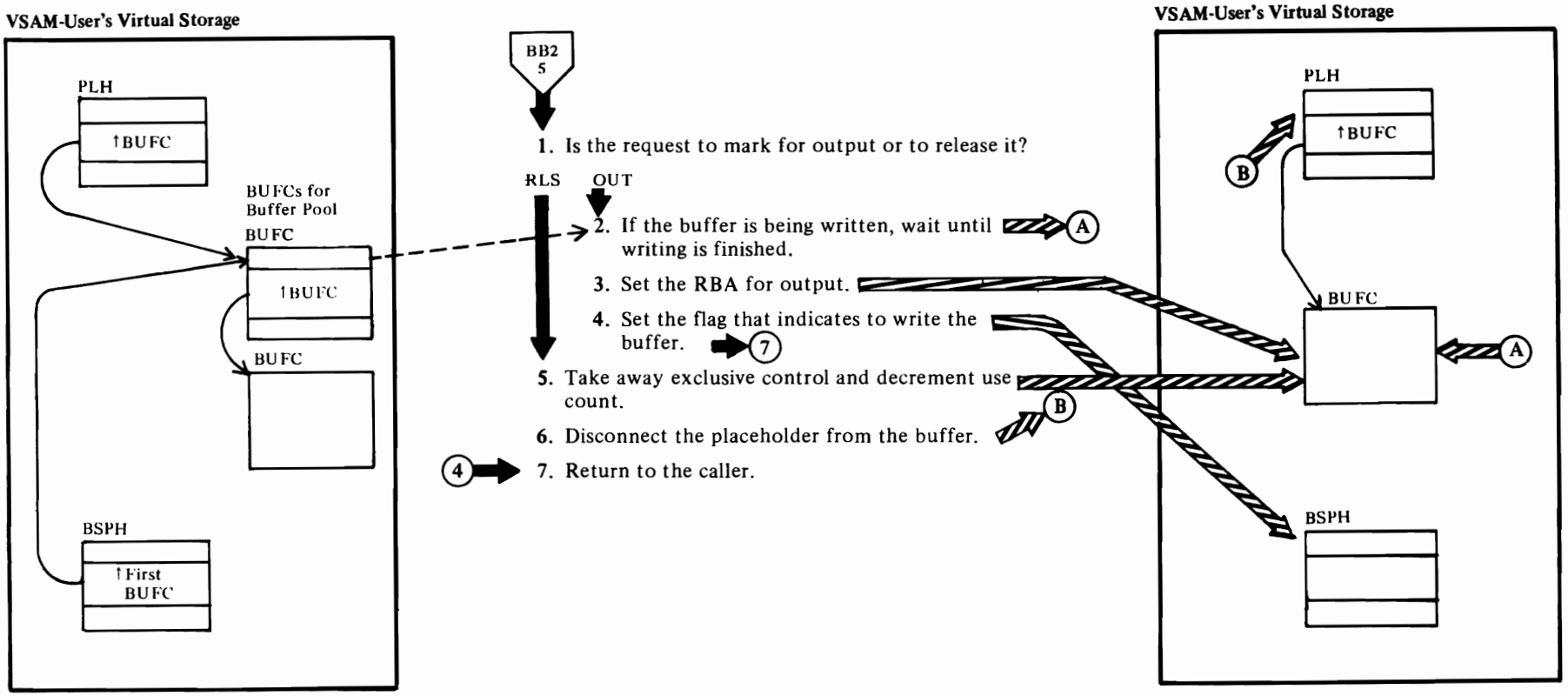
For PUT-update processing, the length of the updated record must be the same as that of the original.

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 Local Shared Resources)



## Notes for Diagram BP1

1 **IDA019RY (MRKBF)**

2 **IDA019RY** calls **IDA019R5 (IDADRQ)**

The request is deferred until the buffer has been written.

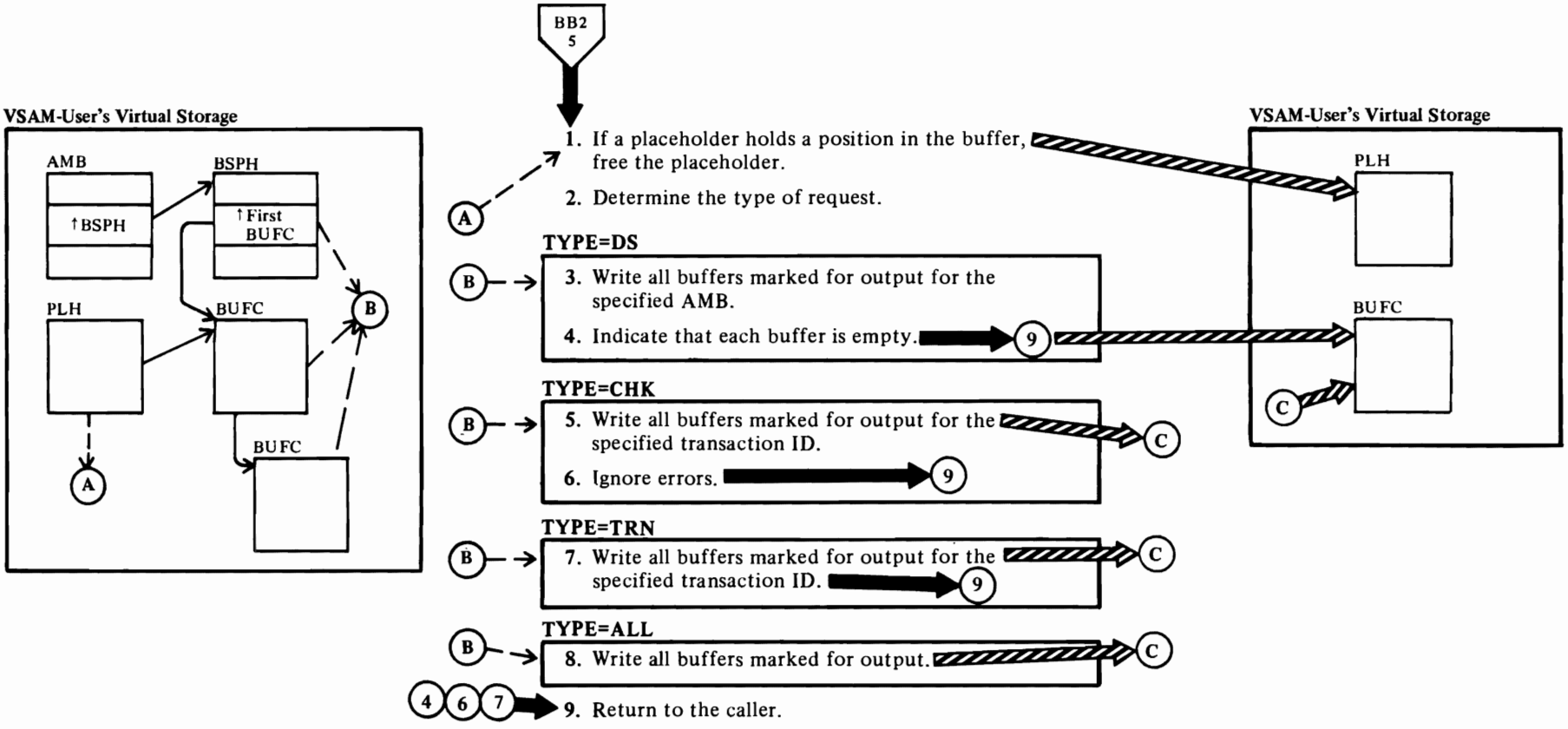
3 **IDA019RY**

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)



## Notes for Diagram BP2

**3** **IDA019RY: WRTBF** calls **IDA019RY (WRBFR)**

WRBFR writes the buffers associated with the BUFCs indicated by the request.

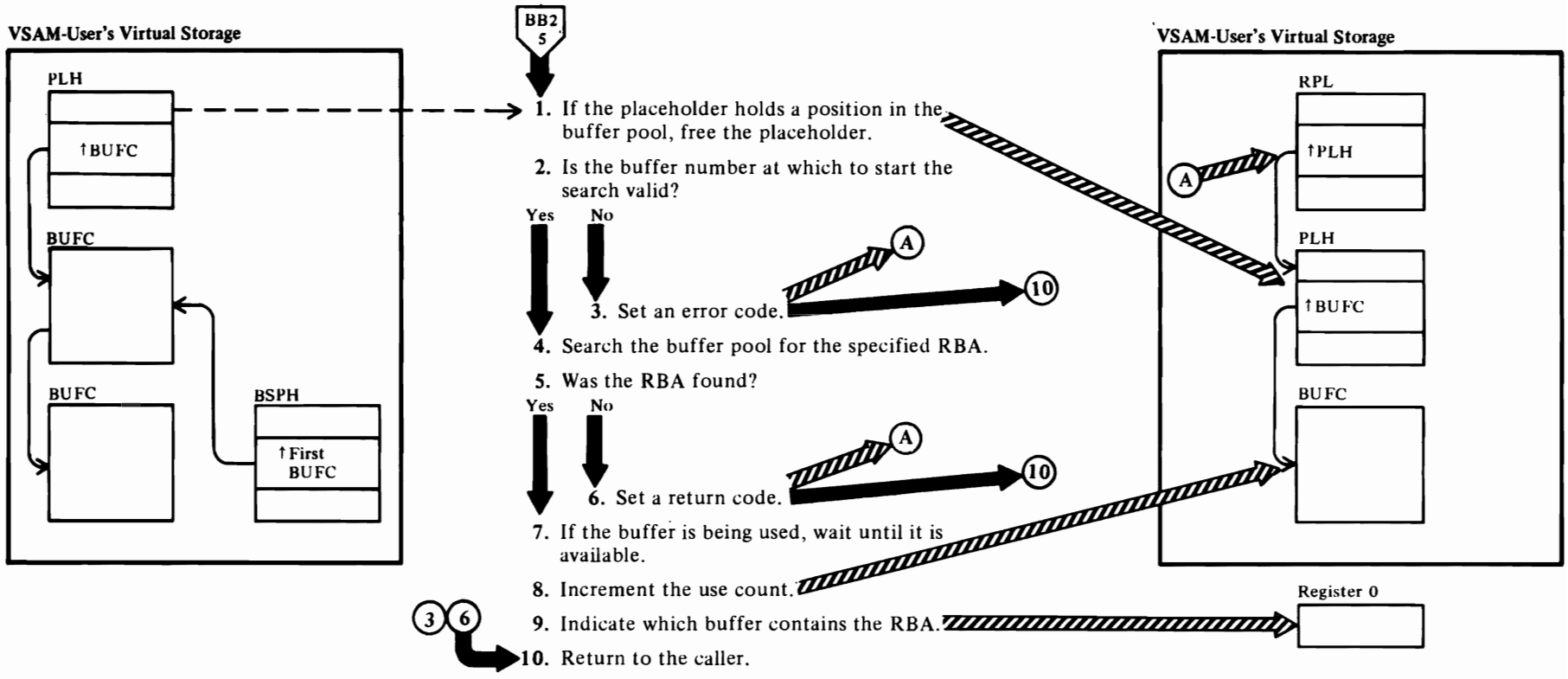
**5** Same as note for step 3.

The user indicates the ID of the transaction in the RPL TRANSID operand.

**7** Same as note for step 3.

**8** Same as note for step 3.

# Diagram BP3. SCHBFR: Searching the Buffer Pool (With Shared Resources)

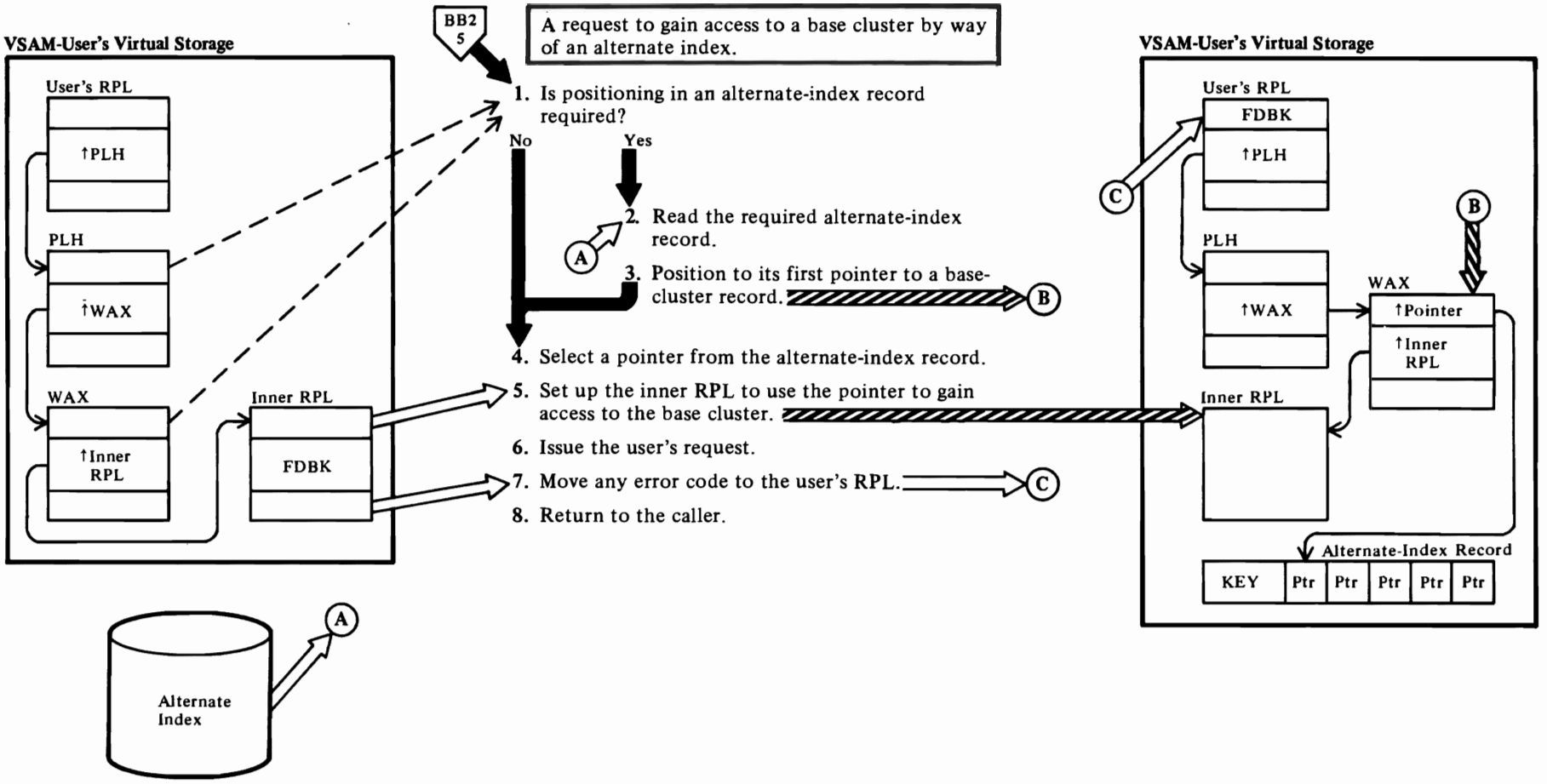


## Notes for Diagram BP3

**7 IDA019RY** calls **IDA019R5 (IDADRQ)**

The request is deferred until the buffer has been processed.

# Diagram BQ1. Processing a Path



## Notes for Diagram BQ1

**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**

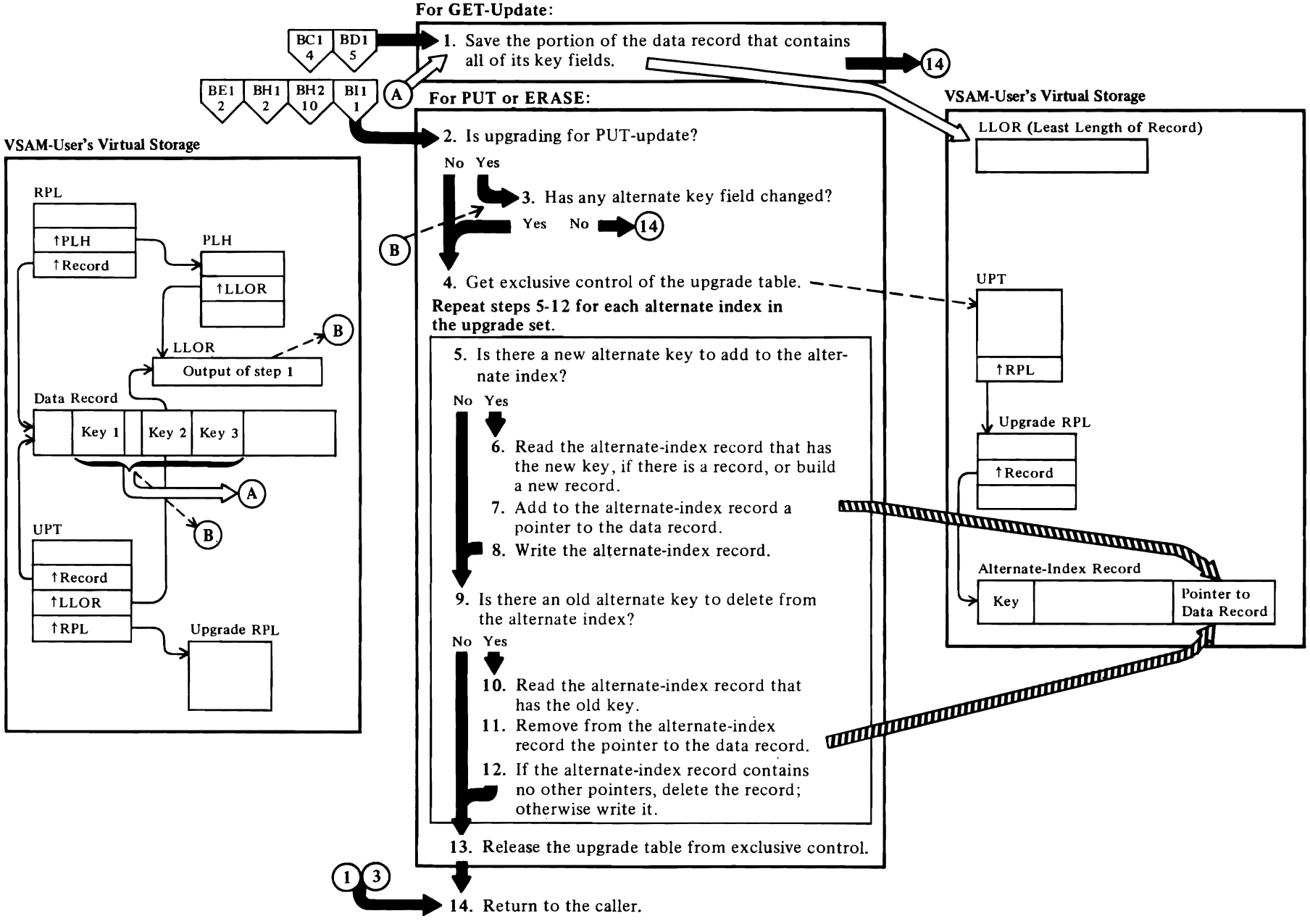
The PLH identifies the alternate-index record positioned at; the WAX indicates the pointer within the alternate-index record positioned at. The alternate-index record contains either prime-key pointers (for a key-sequenced base cluster) or RBA pointers (for an entry-sequenced base cluster).

**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 BR1. Upgrading Alternate Indexes



### Notes for Diagram BR1

**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.

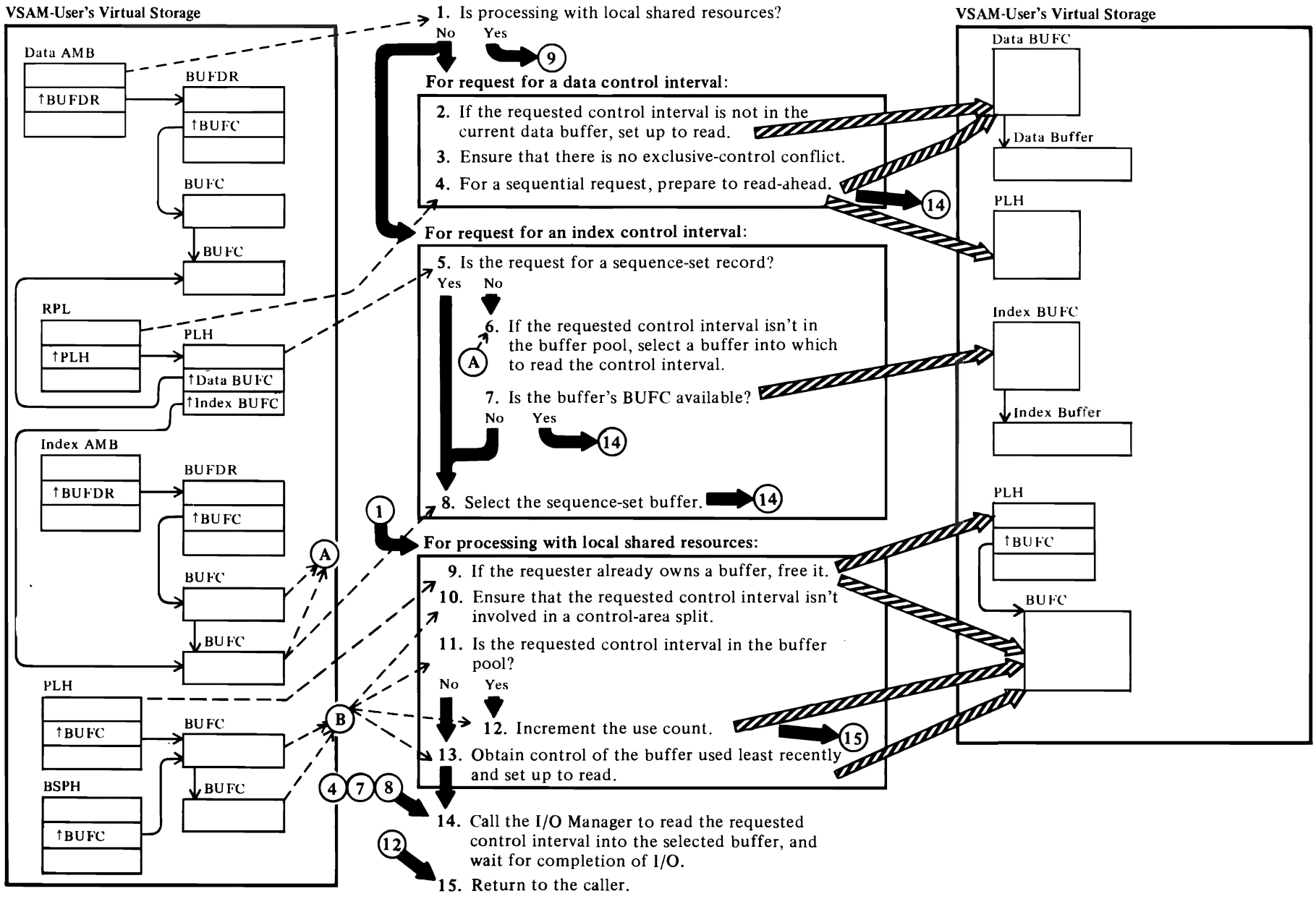
**10 IDA019RU** calls **IDA019R4**

**11 IDA019RU**

**12 IDA019RU** calls **IDA019R4**

**13 IDA019RU**

# Diagram BS1. Buffer Management: Reading a Control Interval into a Buffer



## Notes for Diagram BS1

Buffer Management is called for the processing of almost every Record-Management diagram. See the "Procedure Called-By Directory" for a list of the modules that call IDA0l9RZ.

**1 IDA019RZ (IDAGRB)**

If processing is with shared resources, IDA019RZ calls IDA019RY; if not, it calls IDA019R2.

**2 IDA019R2**

If the requested control interval is in the current data buffer, processing continues at step 15. The read flag in the BUFC is set if:

- The requested control interval is not in the buffer,
- It is in the buffer, but its contents are invalid,
- It is in the buffer, but its exclusive-control level is inappropriate, or
- Share-option 4 is specified.

**3** If exclusive control is required, the requested control interval may not be held in exclusive control or for writing by another string. If it is, an exclusive-control error is indicated: IDA019R2 returns to the caller.

**4** IDA019R2 initializes the REA fields and read flags of the other BUFCs in the chain if the request is for:

- Sequential retrieval,
- Control-area split, or
- Spanned-record retrieval.

**6** If the requested control interval is in the buffer pool, processing continues at step 15. For selecting a buffer for an index-set record, if there are more index buffers than strings, the surplus buffers can be used. The first surplus buffer can be used only for the highest-level index record. Other surplus buffers can be used for other index-set records. The buffer selected is, in this order of priority:

(1) An empty buffer,

(2) A buffer that contains a lower-level index-set record, or

(3) A buffer that contains an index-set record of the same level

**7** The BUFC is not available if its buffer is being used by another string.

**8** The sequence-set buffer is the buffer allocated to the string by Open. It is used for for all requests for a sequence-set record and for requests for an index-set record when there are no surplus buffers.

**9 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.

**12** If data is in the process of being read into the buffer, IDA019RY calls IDA019R5 (IDADRQ) to wait until I/O has finished. If the use count is incremented to more than one and the request is for exclusvie control, a read-exclusive error is indicated.

**13** 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 15.

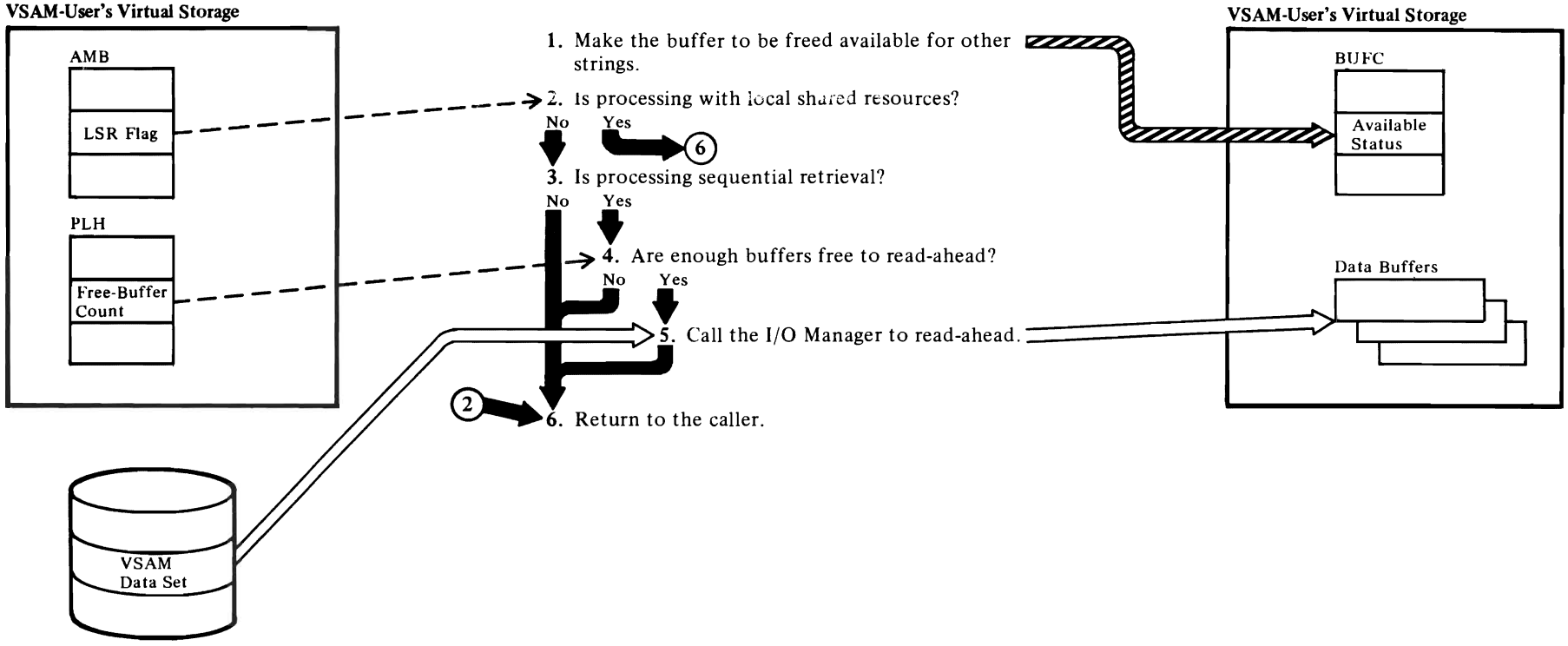
**14 IDA019R2** or **IDA019RY** calls **IDA019R3**

The read is initiated. (See Diagram BT1.)

**IDA019R2** or **IDA019RY** calls **IDA019RZ (IDAWAIT)**

Processing waits for I/O to finish. (See Diagram BS3.)

# Diagram BS2. Buffer Management: Freeing a Buffer



## Notes for Diagram BS2

Many Record-Management routines call Buffer Management. See

the "Procedure Called-By Directory" for a list of the modules that call IDA019RZ.

**1 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 (which the user specifies in the RPL TRANSID operand). 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.

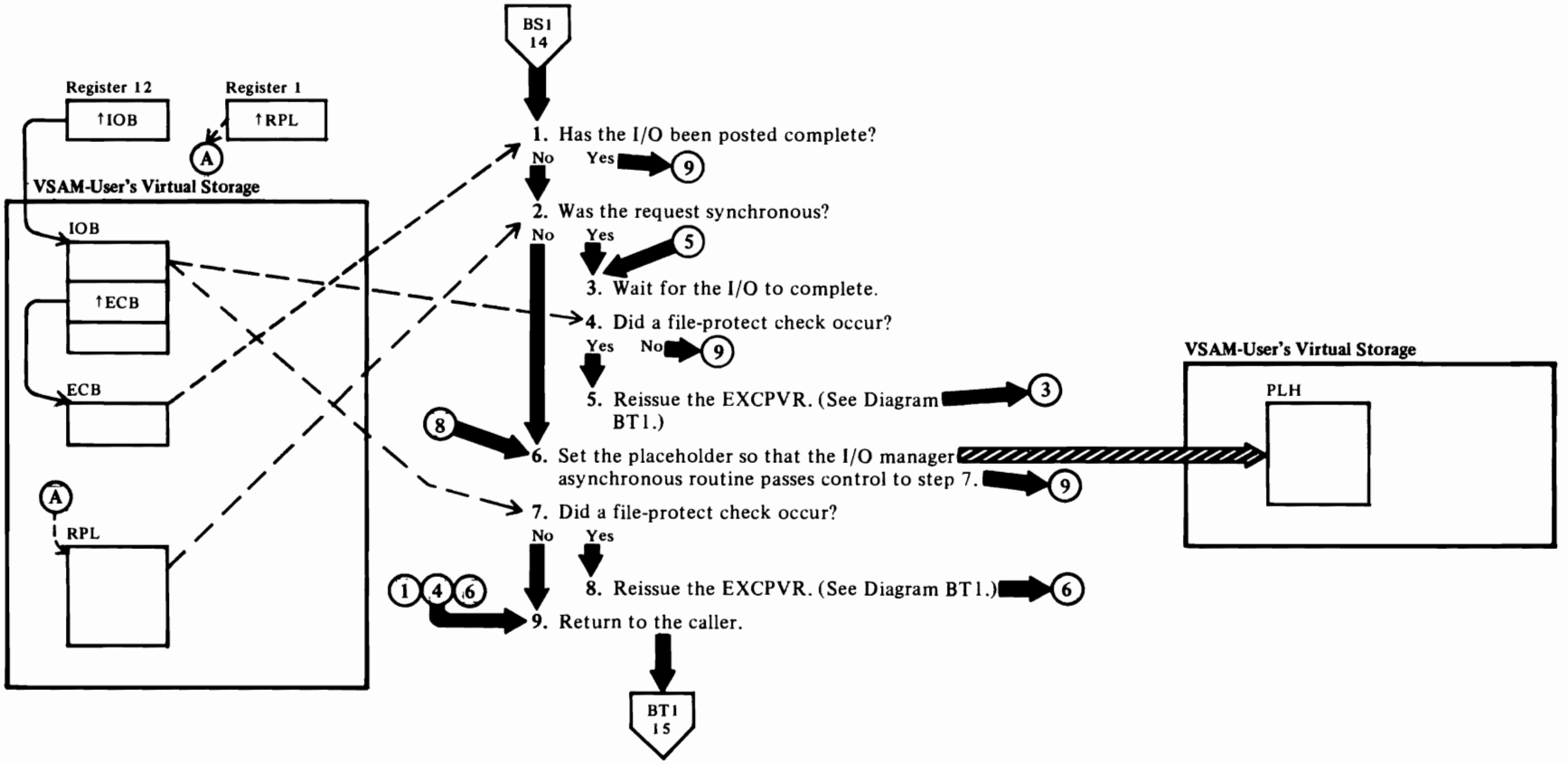
**5 IDA019R2**

IDA019R2 initializes the RBA fields and read flags of the BUFC of each empty buffer if:

- The read threshold has been reached (that is, enough buffers 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.

Read-ahead buffering is begun. (See Diagram BT1.)

# Diagram BS3. Buffer Management: Waiting for I/O Completion



## Notes for Diagram BS3

**1 IDA019RZ: IDAWAIT**

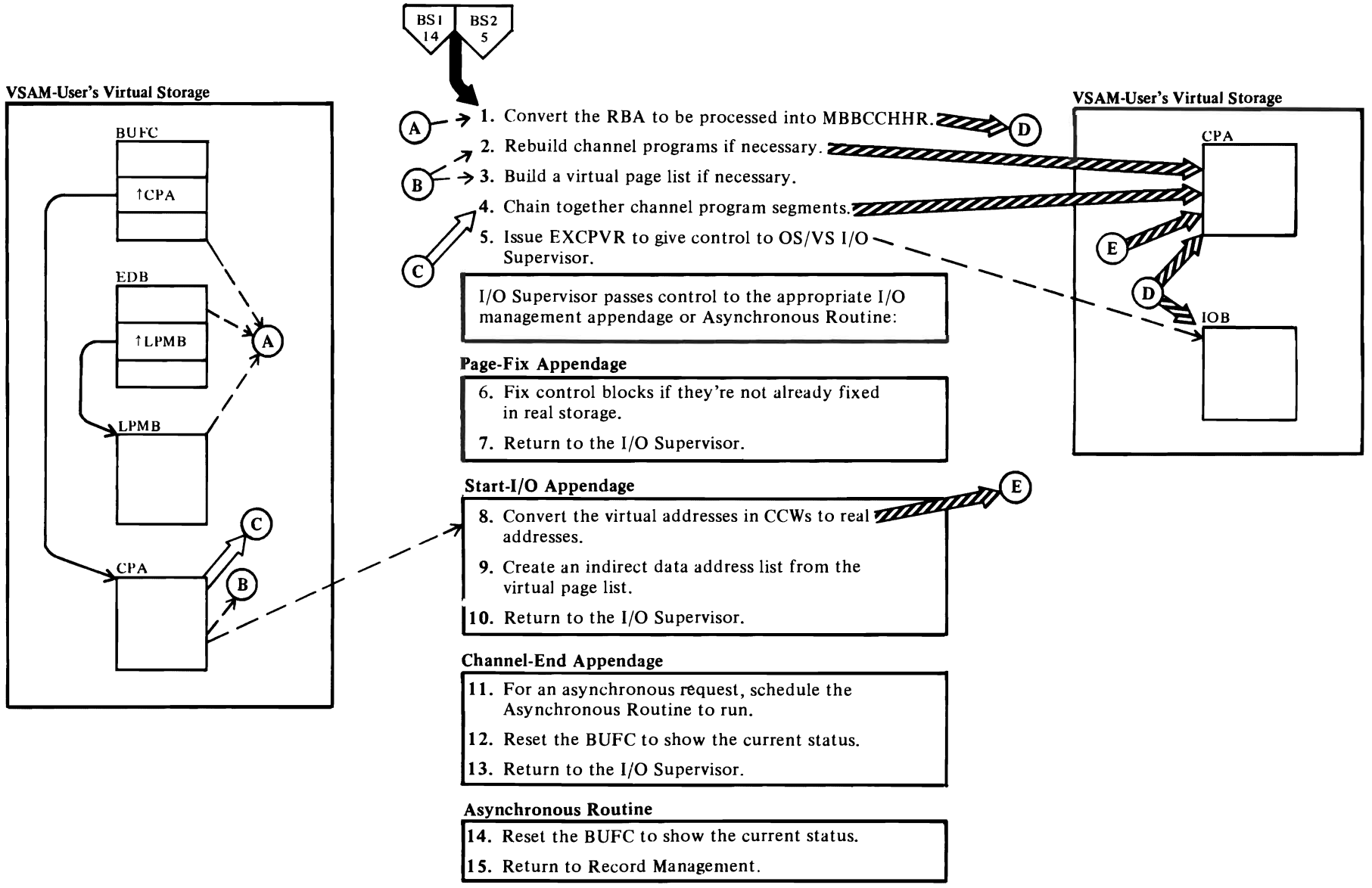
**2** If the RPL specifies synchronous and WAITX, exit to the UPAD routine.

**4** The I/O Manager chains channel program segments together with cylinder seeks if necessary. When a cylinder seek causes a file-protect check, the Abnormal-End Appendage, IDA019R6, resets the starting address of the IOB to point to the CCW that follows the cylinder seek.

**6** The current request is suspended. The Asynchronous Routine eventually resumes the request under an IRB. When it passes control to step 7, return (at step 9) is actually to the Stage-3 Exit Effector.

**8** See note for step 4.

# Diagram BT1. I/O Management



## Notes for Diagram BT1

**1 IDA019R3**

If the RPL specifies synchronous and WAITX, the ECB pointer in the IOB is set to point to the user's ECB. Otherwise, the ECB pointer is set to the VSAM ECB.

**2 IDA019R3**

For processing with shared resources, IDA019R3 calls IDA019SB.

**3 IDA019R3**

The virtual page list contains the virtual address of each block of storage from which to read or into which to write.

**4 IDA019R3**

The OS/VS I/O Supervisor is called by way of SVC 114.

**6 IDA019R9**

The AMB, BUFC, CPA, IOB, and buffers are fixed in real storage for I/O.

**9 IDA019R9**

Each read, write, and write check CCW points to an entry in the indirect data-address list that contains the real address of each storage block of a buffer.

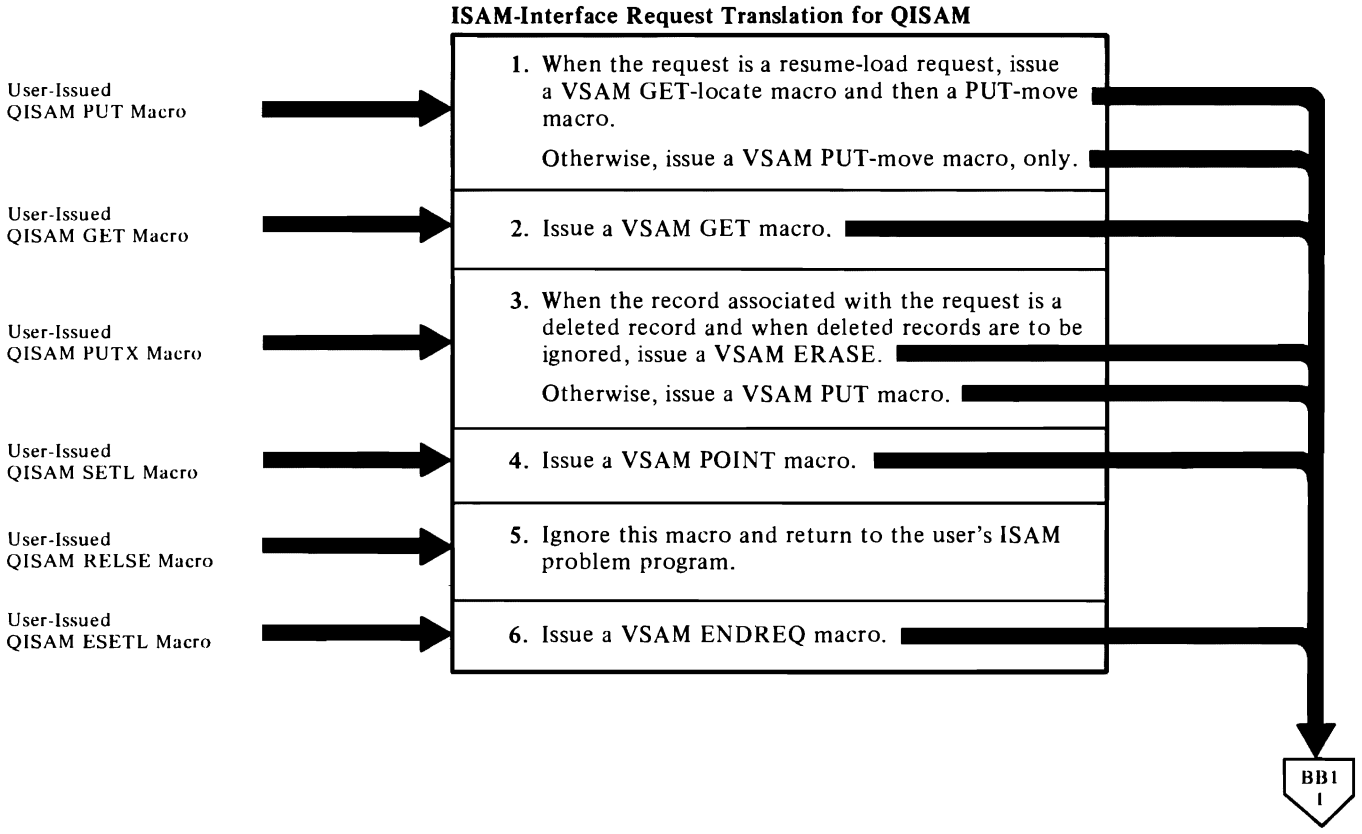
**11 IDA019R6**

For a synchronous request, the IOB is posted when the I/O completes.

**14 IDA019R7**

The return address to Record Management is in the PLH.

# Diagram BU1. ISAM-Interface: Processing a VSAM Data Set



## 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 1 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.

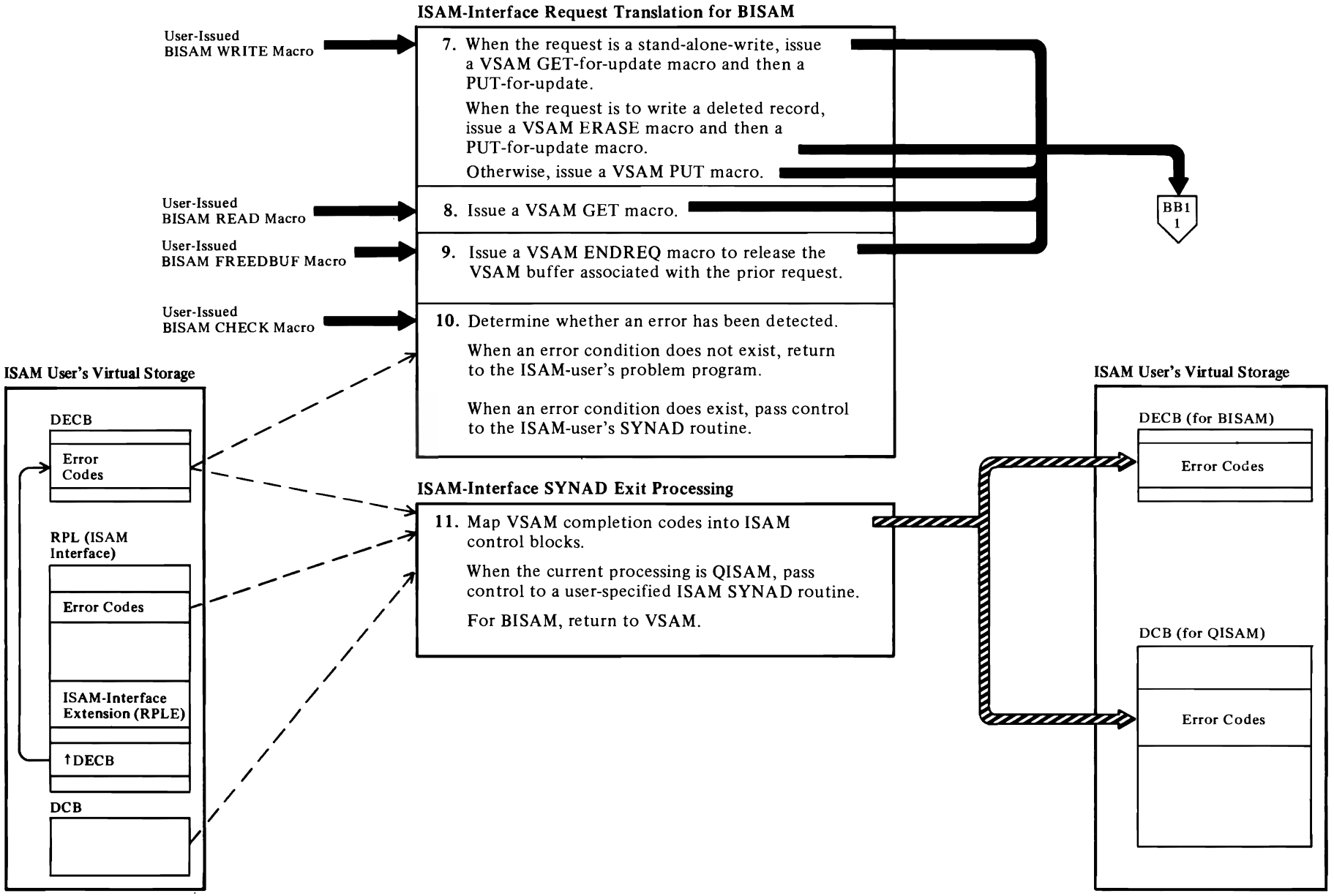
**6 IDAIIPM2: QISAM ESETL Processing**

A VSAM ENDREQ macro instruction 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 recieves 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



## 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.

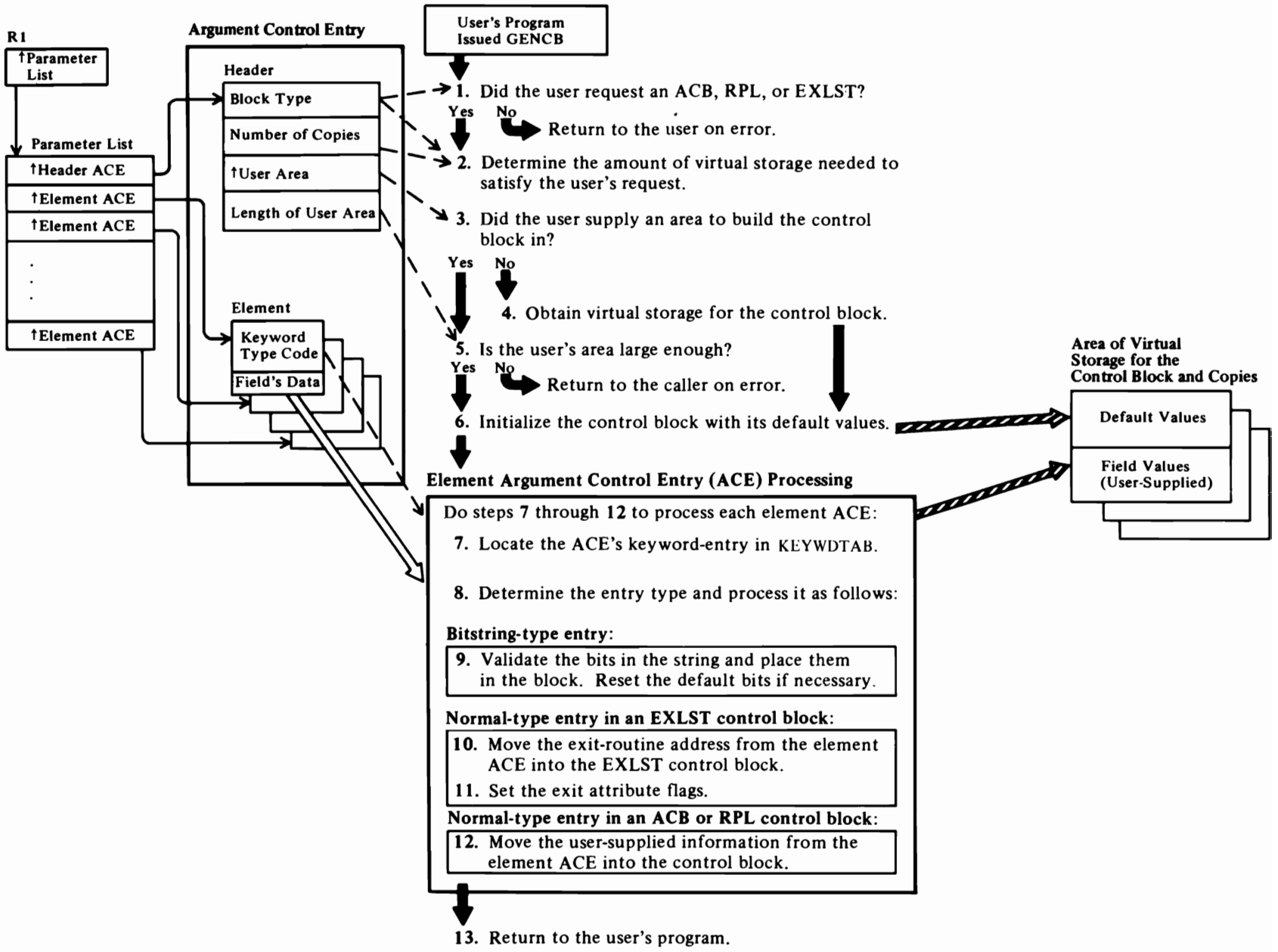
For QISAM processing, the VSAM error codes in the RPL are copied into the DCB, and for BISAM processing, the error codes are copied into the DECB.

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 CA1. GENCB: Build a New Control Block



## Notes for Diagram CA1

**1 IDA019C1**

The GENCB macro instruction is issued to create an ACB, RPL, or EXLST dynamically.

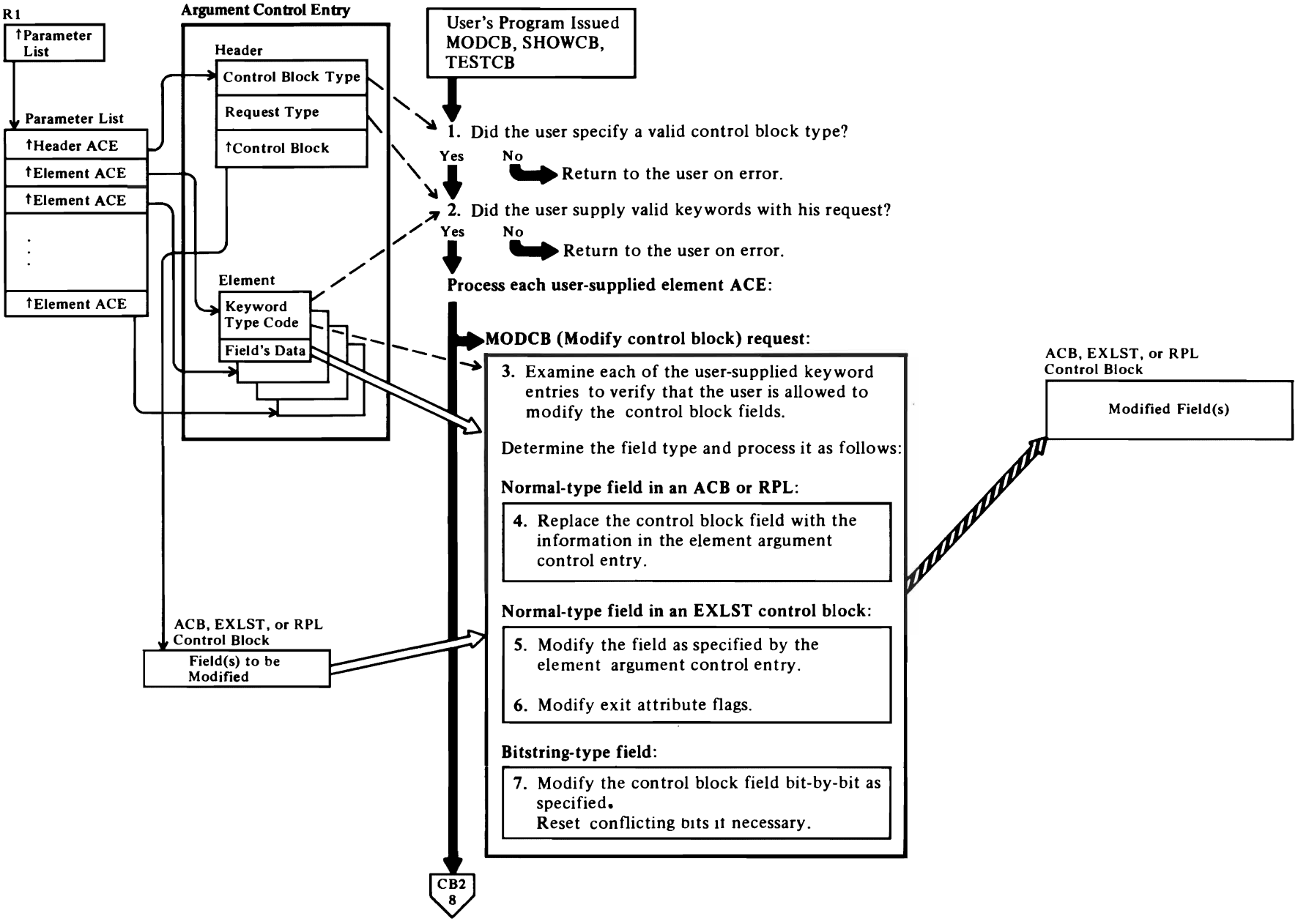
**2-5**

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 instruction 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

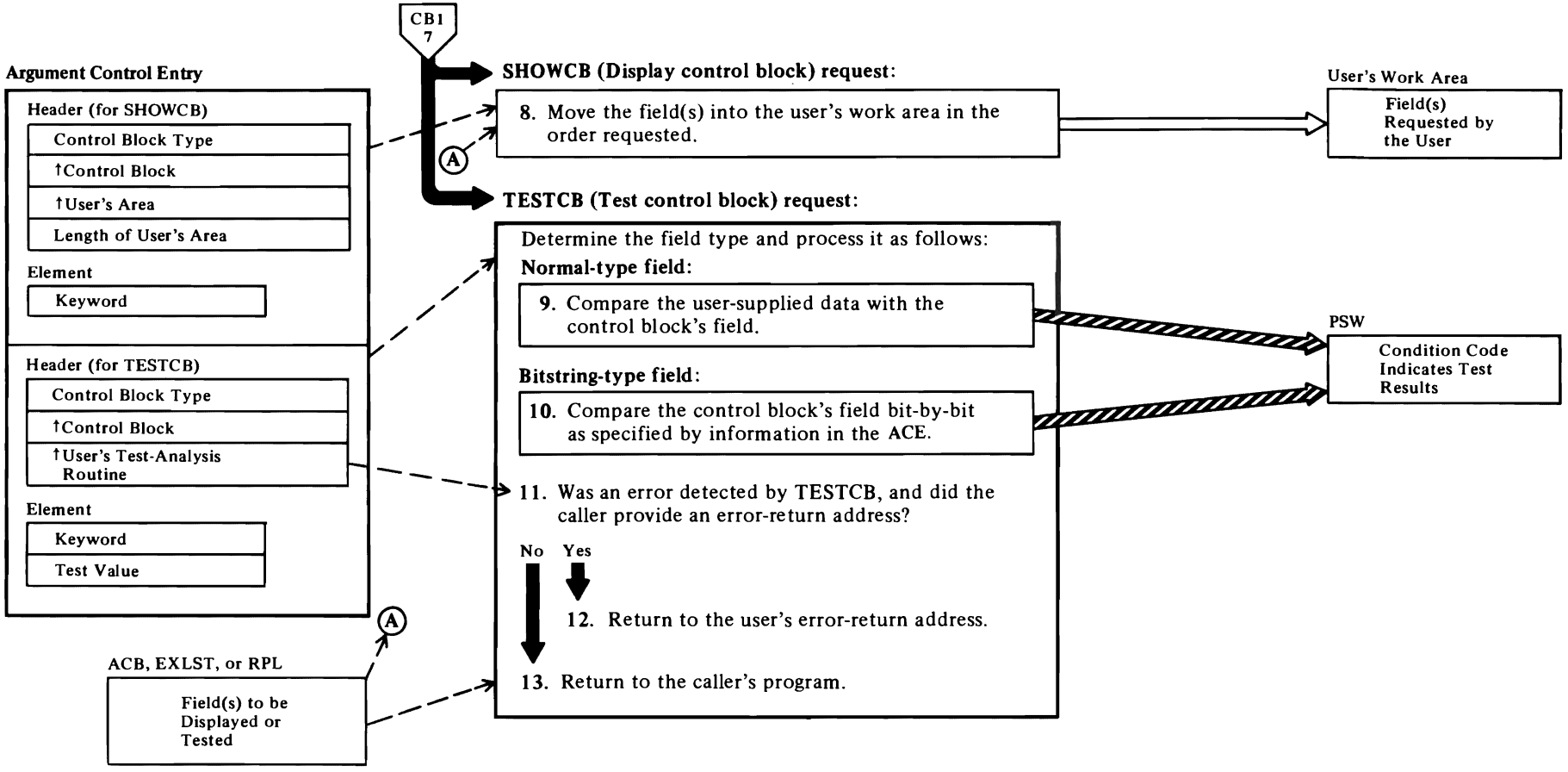


## Notes for Diagram CB1

**1 IDA019C1**

The MODCB, SHOWCB, and TESTCB macro instructions 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



## Notes for Diagram CB2

**4-13**

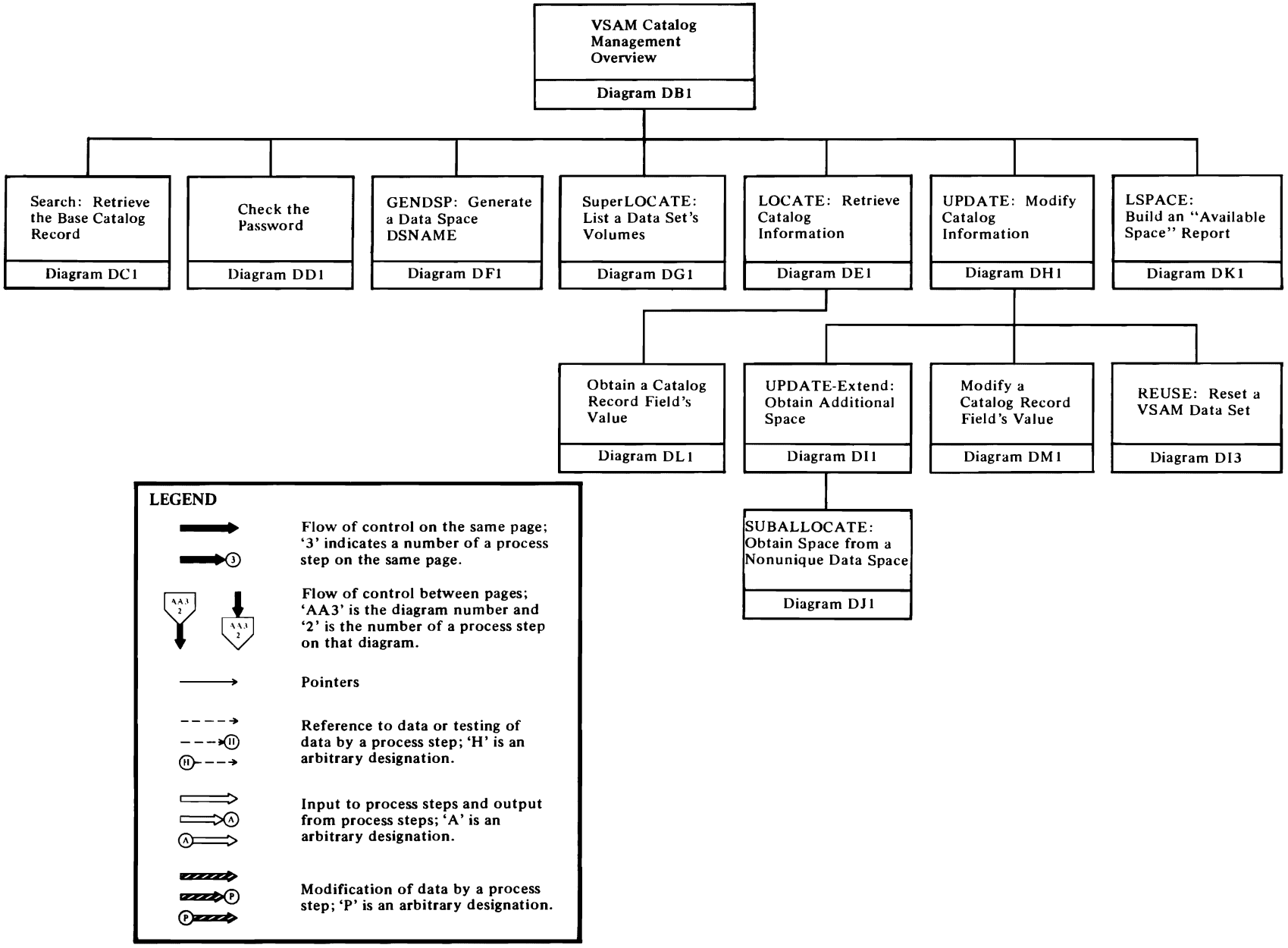
The field attribute table entry contains the length, offset from the beginning of the block, and characteristics of the field in the control block.

Three types of entries are identified in the field attribute table: bitstring, normal, and entries that require a special subroutine to process them.

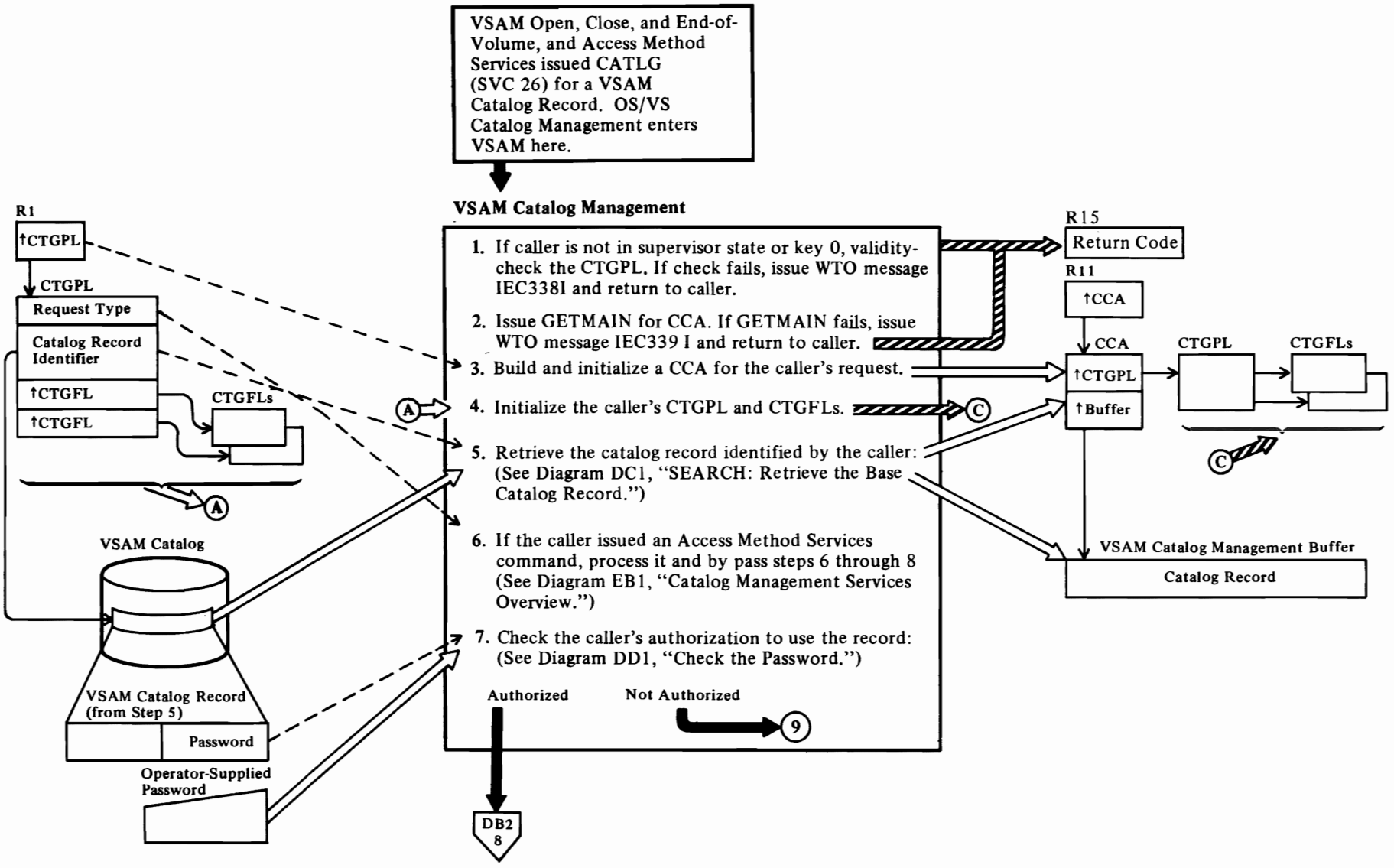
If the entry is a bitstring type, the field attribute table points to a series of bit entries in the bitstring table that are used to modify the control block (MODCB), or are compared to a value supplied by the user (TESTCB).

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. VSAM Catalog Management Table of Contents



# Diagram DB1. VSAM Catalog Management Overview



## Notes for Diagram DB1

VSAM Catalog Management is called by OS/VS Catalog Management when VSAM Open, Close, End-of-volume, and the Access Method Services routines issue the CATLG macro instruction (SVC 26). Register 1 contains the address of the caller's catalog parameter list. The catalog parameter list identifies which catalog record to process and what process to perform.

A user's program can access the VSAM catalog by issuing an Access Method Services utility request. Access Method Services translates the request into an SVC 26 and a catalog parameter list.

The LOCATE command is processed first by the VSAM catalog management routines and then, if the requested information is not in a VSAM catalog, by the OS/VS catalog management routines.

**IGC0002F**

Register 1 contains the address of a catalog parameter list (CTGPL). OS/VS Catalog Management transfers control (XCTL) to VSAM catalog management transient module, IGG0CLA1.

IGG0CLA1 loads IGG0CLC9, if IGG0CLC9 is not already loaded, and calls IGG0CLC9 to process the VSAM catalog management request.

**1 IGG0CLC9: BLDCCA**

A call is made to the task supervisor validity-check routine to verify that the storage passed as a CTGPL is owned by the caller. A condition code of 8 is set in the PSW if the check is successful.

**2 IGG0CLC9: BLDCCA**

Issue a page boundary GETMAIN for CCA and record areas. If return code is not zero, issue "insufficient storage" message. Set return code 8 in register 15 if caller was a SUPERLOCATE request or a translated request. If it was not, set reason and error code and module ID in the CTGPL.

**3 IGG0CLC9: BLDCCA**

The catalog control area (CCA) contains data about catalog records retrieved to process the request. The CCA also contains a register save area that shows the flow of control between catalog management routines used to process the request.

Each time a catalog management routine calls another catalog management routine, the contents of registers 12, 13, and 14 are put in the CCA's register save area. Register 13 contains the address of the next 12-byte register save area in the CCA. Register 12 contains the address of the calling routine. Register 14 contains the return address to the calling routine.

See "Data Areas" for details about the CCA and CTGPL.

See "Diagnostic Aids" for details about the CCA register save area.

**4 IGG0CLAB: IGGPACDV** (calls **IGGPSCNC (IGG0CLAY)**)

The caller's work area and each CTGFL are checked to ensure that it is within the caller's address space.

The CTGFL's field-name value is used to obtain dictionary data that defines the field's characteristics and location within the record.

See "Data Areas" for details about the field name dictionary.

**5 IGG0CLAB: IGGPACDV** (calls **IGGPSCAT (IGG0CLAH)**)

The catalog record is identified by the caller's dsname value, volume serial number, or control interval number.

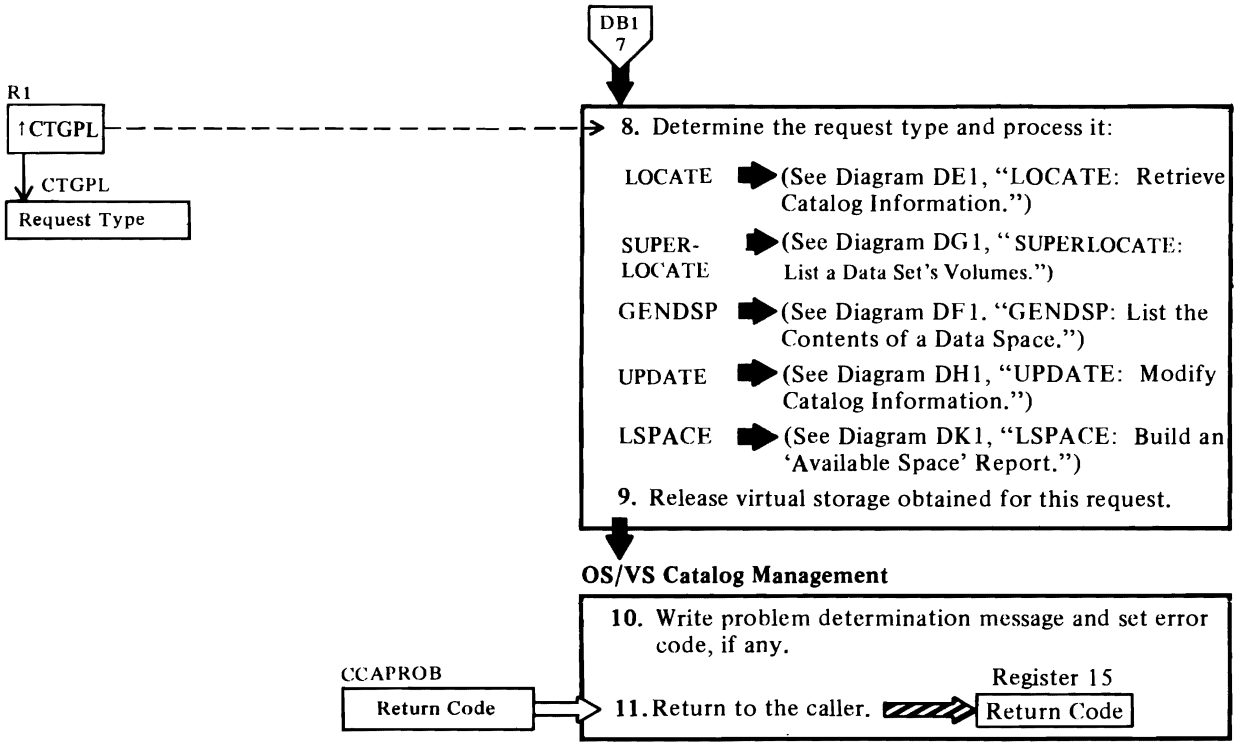
**6 IGG0CLAB: IGGPACDV** (calls **IGGPCDVR (IGG0CLAT)**)

An Access Method Services command is translated into a catalog management services request to define, modify, delete, or list catalog records.

**7 IGG0CLAB: IGGPACDV** (calls **IGGPCKAU (IGG0CLBM)**)

The caller's request type determines the level of password that, when supplied by the operator, allows the VSAM catalog management routines to complete the caller's request.

# Diagram DB2. VSAM Catalog Management Overview



## Notes for Diagram DB2

**8 IGG0CLAB: IGGPACDV** (calls **IGGPSLOC (IGG0CLAM), IGGPGDSP (IGG0CLBJ), IGGPLOC (IGG0CLAZ), IGGPUPD (IGG0CLAV),** or **IGGPLSP (IGG0CLBK))**

**IGGPSLOC:**

A SUPERLOCATE request builds a list of all volumes and units associated with a dsname.

**IGGPGDSP:**

A GENDSP request builds a list of all VSAM data sets in a VSAM data space.

**IGGPLOC:**

A LOCATE request retrieves information from the catalog record.

**IGGPUPD:**

An UPDATE request modifies information in a catalog record. An UPDATE request can also obtain direct-access space for the data set or index identified by the dsname value.

**IGGPLSP:**

A LSPACE request determines the amount of available space on a VSAM direct-access volume, when the volume is described in a VSAM catalog.

**9 IGG0CLC9: IGGPRCLU**

When the VSAM catalog management request is complete, all virtual storage obtained for work areas, control blocks, and the request's CCA is returned to the OS/VS system.

**10 IGG0CLC9: IGGPRCU**

Write problem determination message, if it was a SUPERLOCATE request or a translated request, and set error code in CTGPL.

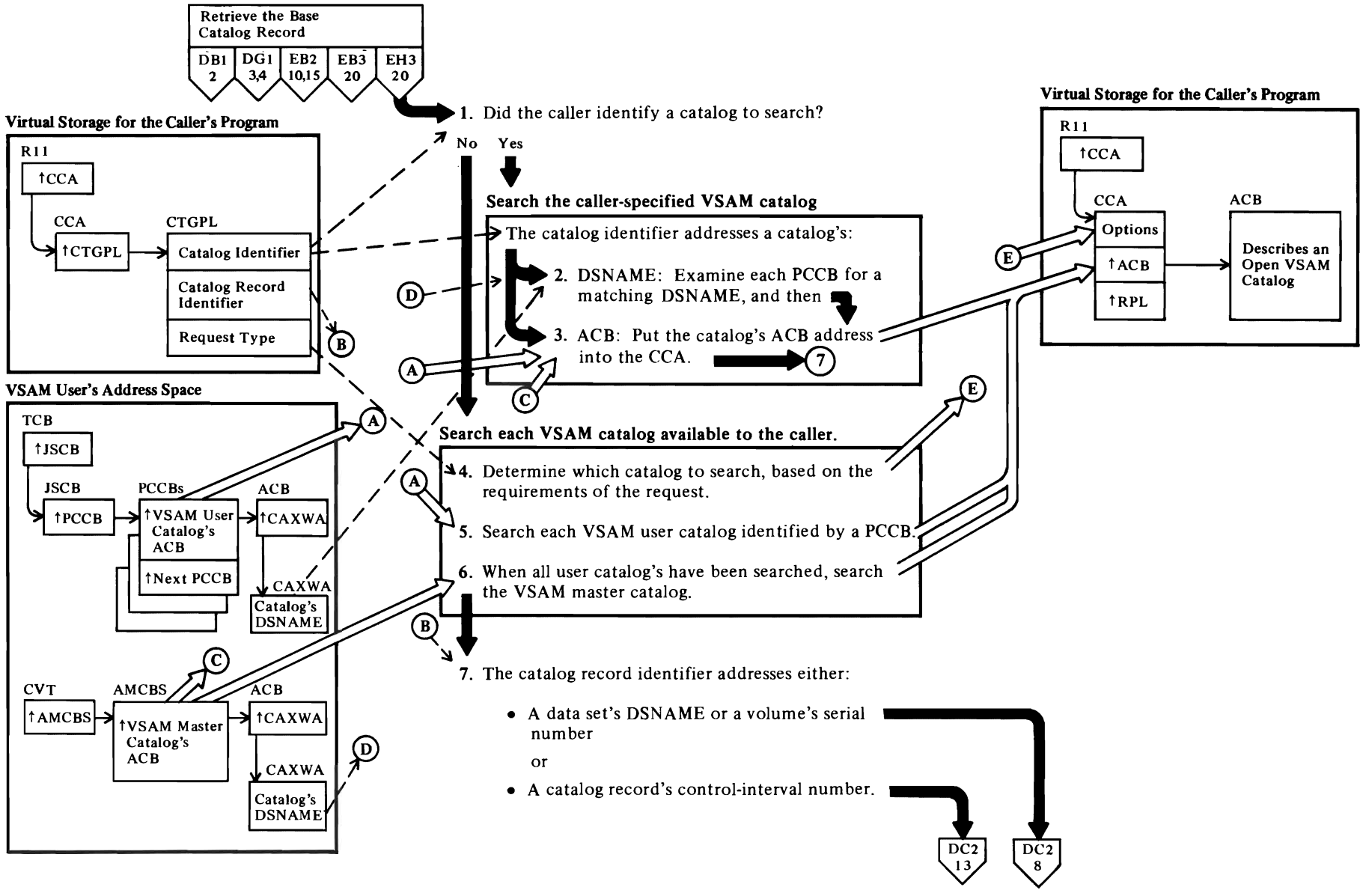
**11**

VSAM catalog management common processing (IGG0CLC9) sets a return code in register 2 and returns to IGG0CLA1.

IGG0CLA1 deletes IGG0CLC9 and transfers control (via XCTL) to IGC0002F. Register 1 contains the 2s complement of register 1's contents when IGG0CLA1 was entered, to indicate to IGC0002F whether or not the CATLG macro instruction (SVC 26) was issued by a VSAM catalog management procedure.

IGC0002F puts the return code (register 2's contents) into register 15 and returns to the caller via the SVC return.

# Diagram DC1. SEARCH: Retrieve the Base Catalog Record



## Notes for Diagram DC1

**1 IGG0CLAH: IGGPSCAT**

The CTGPL's catalog identifier field, set by the caller, can contain the address of a catalog's ACB, the address of a catalog's dsname, or 0.

See "Data Areas" for details about the CCA, ACB, and CTGPL.

**2 IGG0CLAH: IGGPSCA**

The catalog specified by the caller is the only catalog searched. The Catalog Management Services DEFINE routine calls the Search routine to confirm that, when a caller wants to create a VSAM cluster or catalog, the new cluster or catalog dsname isn't duplicated in the catalog. The caller (Catalog Management Services DEFINE routine) expects the "no record found" return code.

If the CTGPL's catalog identifier field contains the address of a catalog dsname, the search routine examines each protected catalog control block (PCCB) for a matching dsname field. Each PCCB contains the address of its catalog's ACB.

If no PCCB contains a matching dsname, the user-supplied catalog dsname refers to either a nonexistent catalog or to an unopened catalog.

See "Data Areas" for details about the CCA and PCCB.

See "Diagnostic Aids" for details about catalog management error codes.

**3 IGG0CLAH: IGGPSCA**

**4 IGG0CLAH: IGGPSCA**

Some user requests, such as DEFINE CATALOG and DELETE CATALOG, require searching the VSAM master catalog and prohibit searching user catalogs, even if they are specified.

If the CTGPL's catalog identifier field contains 0, the VSAM user catalogs specified by the user's JCL JOBCAT and STEPCAT DD statements, and the VSAM master catalog, are searched until either the record is found or there are no more catalogs to search.

See "Data Areas" for details about the CTGPL search options, cluster catalog records, and volume catalog records.

The JSCB contains the address of the first PCCB in the PCCB chain. Each PCCB describes one of the VSAM user catalogs that have been opened to satisfy the user's JCL JOBCAT and STEPCAT DD statements. A PCCB contains the address of a catalog's ACB. The catalog's ACB address is put in the CCA to identify the catalog being searched.

See "Data Areas" for details about the PCCB, ACB, and CCA.

**6 IGG0CLAH: IGGPSCA**

The AMCBS (addressed by the CVT) contains the address of the VSAM master catalog's ACB.

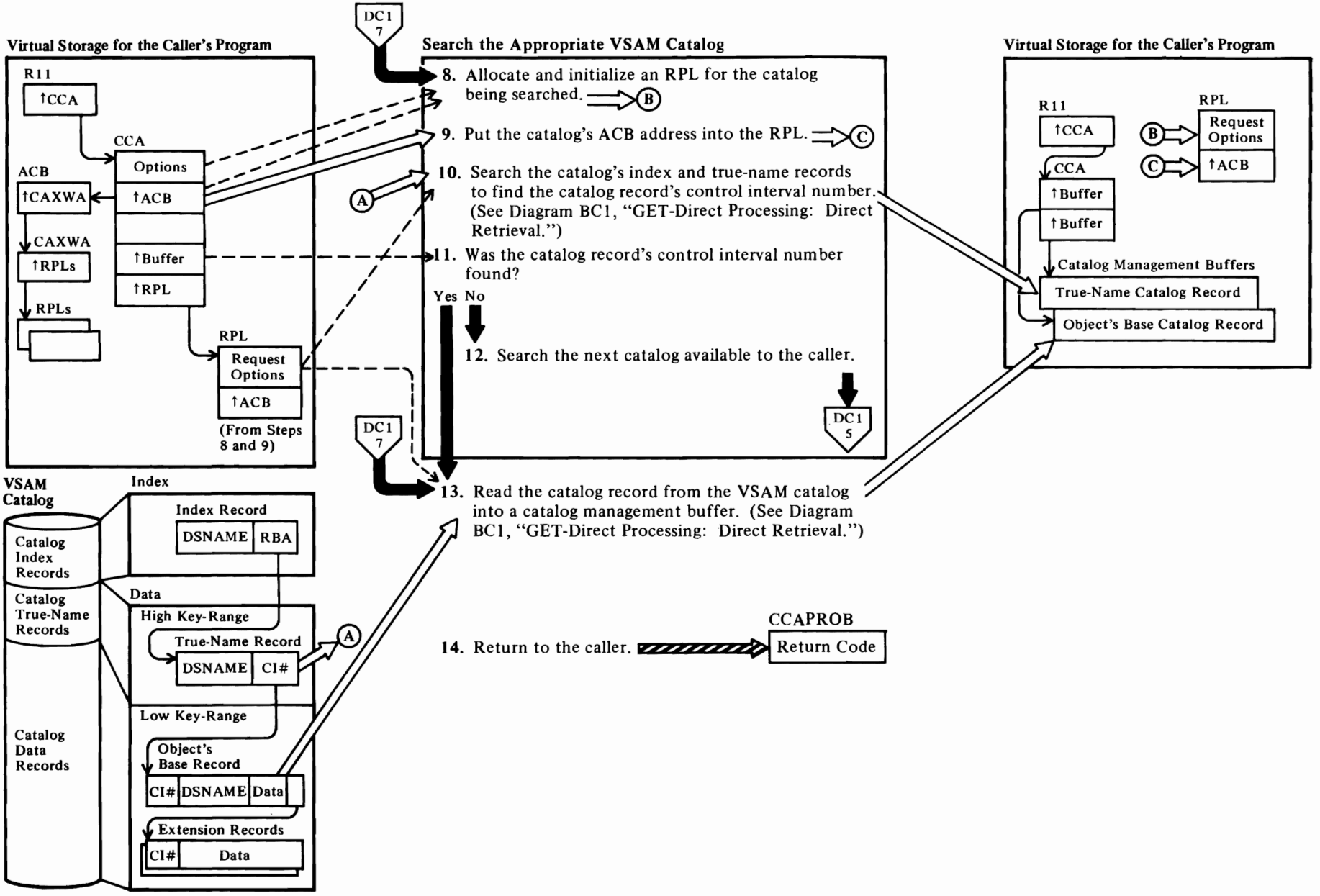
See "Data Areas" for details about the AMCBS and ACB.

See *OS/VS1 System Data Areas* for details about the CVT.

**7 IGG0CLAH: IGGPSCAT**

If the CTGPL's catalog record identifier addresses the record's control interval number, the catalog record can be retrieved without a search of the catalog's index.

# Diagram DC2. SEARCH: Retrieve the Base Catalog Record



## Notes for Diagram DC2

**8 IGG0CLAH: IGGPRPLM**

The search routine assigns one request parameter list (RPL) to the caller. Catalog management routines issue GET and PUT macro instructions to retrieve and write catalog records. Each record-management request (GET, PUT, etc.) needed to satisfy the caller's catalog-management request refers to this RPL. This RPL is initialized for a caller and used as often as necessary to process the caller's catalog-management request. When the caller's catalog-management request is completed, the RPL is assigned to another caller.

**9 IGG0CLAH: IGGPRPLM**

**10 IGG0CLAH: IGGPSCAT** (calls **IGGPGET (IGG0CLBI)**)

The goal of the search is to find the true name record identified by the dsname or the volume serial number. The true name record contains the cluster's dsname or volume serial number and the control interval number of the cluster or volume catalog record.

See "Data Areas" for details about the catalog record.

**12 IGG0CLAH: IGGPSCA**

If the caller supplied a catalog's ACB address or dsname, no further catalog searches are performed. The search routine sets the "no record found" error code in CCACD1 and returns to the caller. If the VSAM master catalog and all VSAM user catalogs available to the user's program have been unsuccessfully searched, the search routine returns to the caller with the same error code.

See "Diagnostic Aids" for details about catalog management error codes.

**13 IGG0CLAH: IGGPSCAT** (calls **IGGPGET (IGG0CLBI)**)

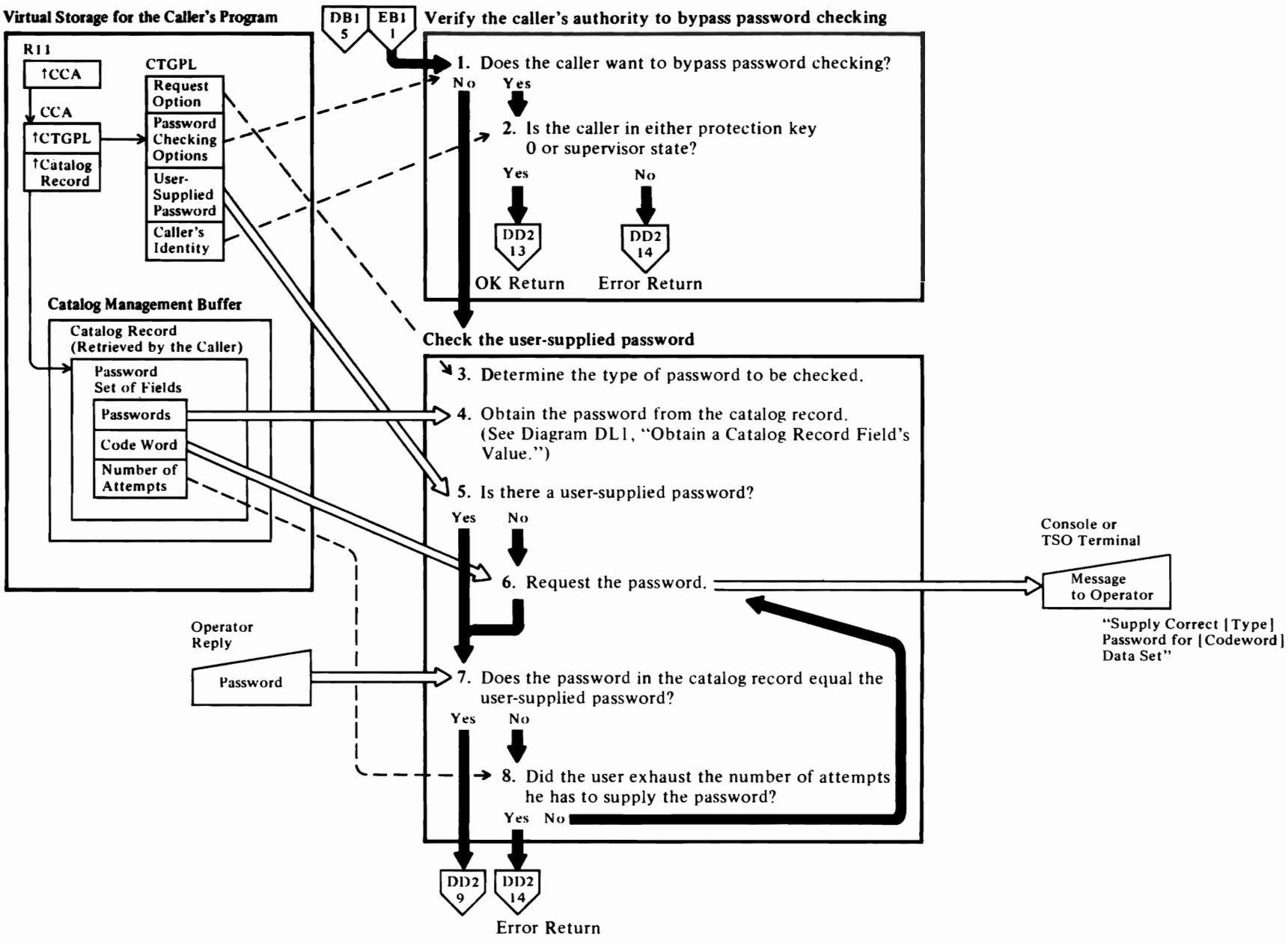
The catalog record is located by its control interval number and read into a catalog management buffer. The buffer's address is put into the CCA.

See "Data Areas" for details about the CCA.

**14 IGG0CLAH: IGGPSCAT**

See "Diagnostic Aids" for details about catalog management return codes and error codes.

# Diagram DD1. Check the Password



## Notes on Diagram DD1

**IDA0192C** and **IGG0CLAB: IGGPACDV** (calls **IGGPCKAU (IGG0CLBM)**)

When the VSAM Open routine (IDA0192C) calls VSAM Catalog Management to retrieve a cluster catalog record, the password checking routine confirms the user's authorization to gain access to the cluster.

**IGG0CLAT: IGGPCDVR** (calls **IGGPCKAU (IGG0CLBM)**)

When an Access Method Services routine calls a catalog management services routine, the password checking routine confirms the user's authorization to gain access to the VSAM catalog or a specific catalog record.

The catalog record containing the password(s) is available in the buffer addressed by the caller's CCA.

See "Data Areas" for password set of fields details.

The type of processing that the user is allowed to do with the data set is determined by the password:

- Master password: The user is allowed to modify passwords and catalog records that describe his data set, and to process his data set's control intervals and records.
- Control-interval password: The user is allowed to process the data set's control intervals as well as its records.
- Update password: The user is allowed to process his data set's records.
- Read-only password: The user is allowed to read, but not to write (add or update), records in his data set.

**1 IGG0CLBM: IGGPCKAU**

If the user's password has been verified during a previous catalog management request, the caller (VSAM Open, or a Catalog Management Services routine) can set the CTGPL's bypass-password-checking flag on.

**2 IGG0CLBM: IGGPCKAU**

Other VSAM catalog management callers, such as the user's program (with Access Method Services commands), and utility programs, are not in protection key 0 or supervisor state. If these programs attempt to bypass password checking, the password checking routine sets an error return code that prevents further VSAM catalog management processing for the caller's program.

**3 IGG0CLB6: IGGPSPSC**

The caller can indicate what type of password is supplied with the CTGPL, but the password checking routine determines the type of password required for the request.

**4 IGG0CLBM: IGGPCKEX**

The password is in the password set-of-fields in the cluster, data set, or index catalog record. The CTGPL can contain a password that the user supplied in a JCL statement.

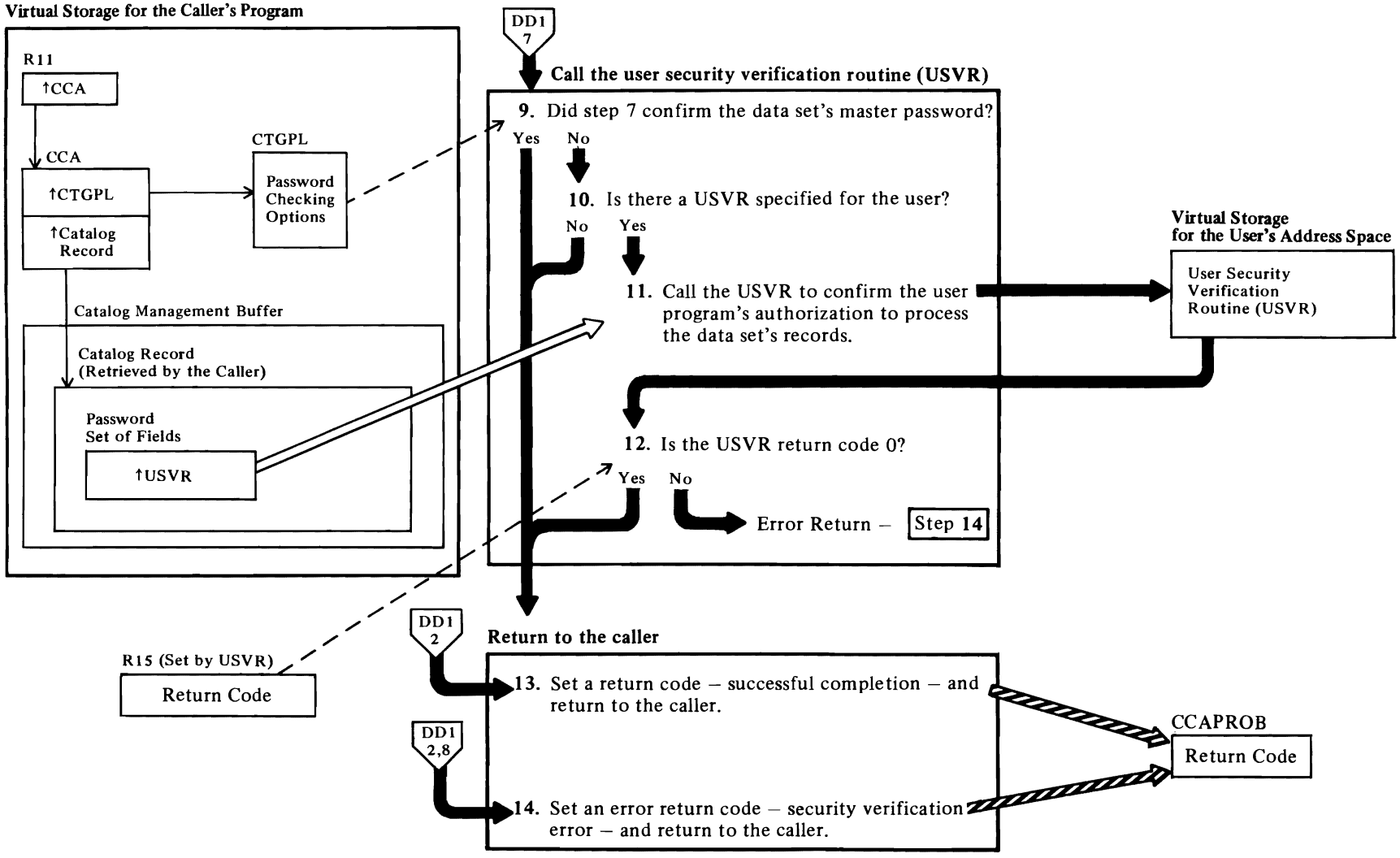
**6 IGG0CLBM: IGGPPWGT**

The console operator, or TSO user, can reply to the VSAM request-for-password message with a password.

**7 IGG0CLBM: IGGPPWVR**

**8 IGG0CLBM: IGGPPWVR**

# Diagram DD2. Check the Password



## Notes for Diagram DD2

**9 IGG0CLB6: IGGPINMD**

If the user supplied the correct master password, the user security verification routine (USVR), if it exists, is bypassed. If a USVR exists, the USVR exit is taken only if the user provided another type of password correctly.

**10 IGG0CLB6: IGGPINMD**

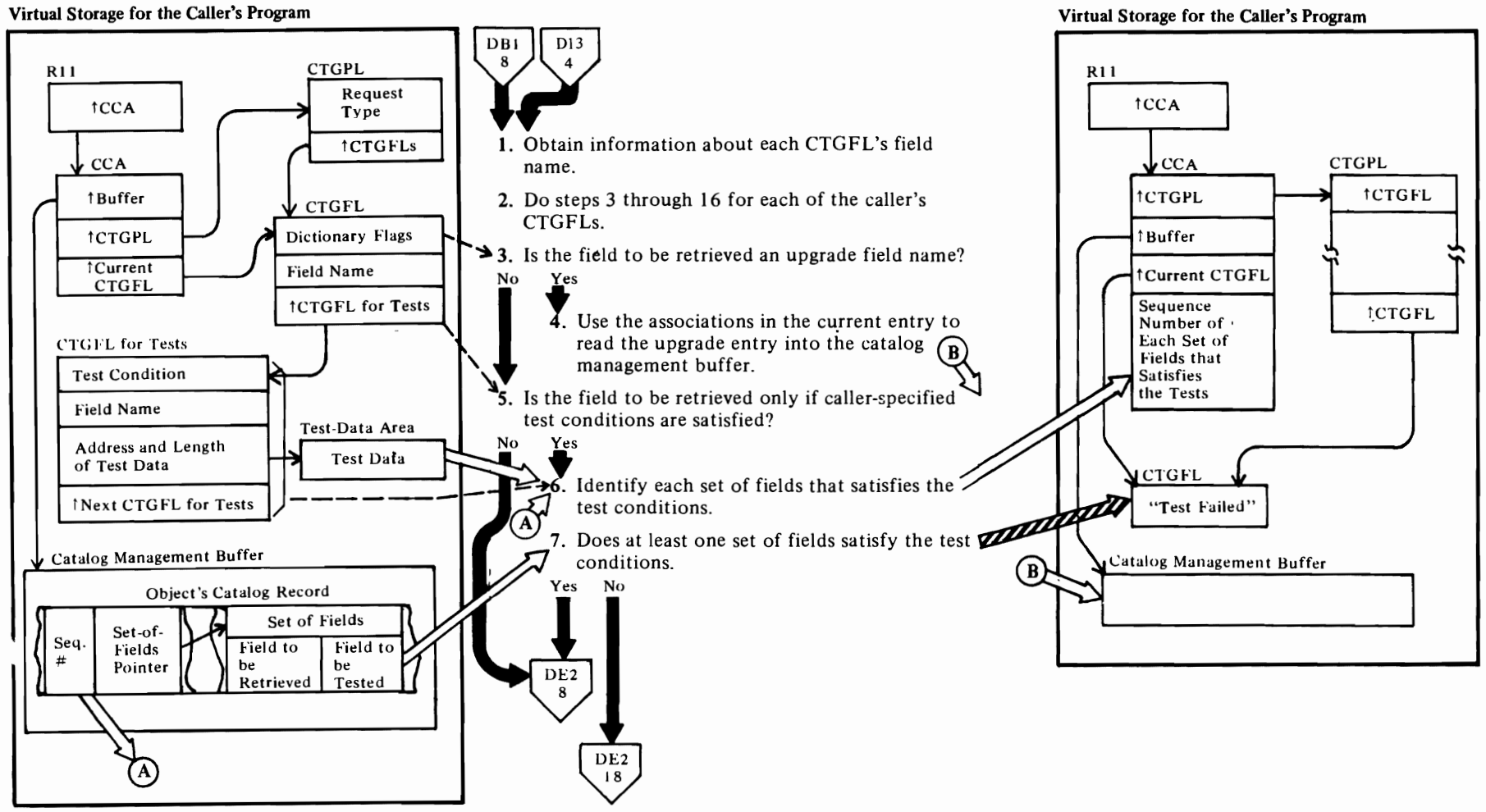
If a user security verification routine exists, its address is in the catalog record's password set-of-fields.

See "Data Areas" for details about the cluster catalog record and the password set-of-fields.

**11 IGG0CLB6: IGGPINMD**

The user security verification routine (USVR) is an installation-supplied routine that confirms a user's authorization to gain access to the data set. The USVR confirms that the user satisfies the installation's security verification criteria.

# Diagram DE1. LOCATE: Retrieve Catalog Information



## Notes for Diagram DE1

**IDA0192C**

The VSAM Open routine issues the CATLG macro instruction (SVC 26) to obtain data set and volume information about the user's data set and index. See Diagram AC1, VSAM Open Processing, for details.

**IDA0557A**

The VSAM end-of-volume routine issues the CATLG macro instruction (SVC 26) to obtain volume information about the extents added to the user's data set. See Diagram AE1, VSAM End-of-Volume Processing, for details.

**IGG0CLAB: IGGPACDV** (calls **IGGPLOC (IGG0CLAZ)**)

When the caller issues a CATLG macro instruction, register 1 points to the caller's catalog parameter list (CTGPL). The CTGPL's request options are decoded and the base catalog record is retrieved for the request. See Diagram DB1, VSAM Catalog Management Overview, for details about initial catalog management processing and request decoding.

**IGG0CLB7: IGGPRUS, IGGPFRWK** (calls **IGGPLOC (IGG0CLAZ)**)

Upon completion of Reuse processing, LOCATE is called to return catalog field information from the reset entry.

1 **IGG0CLAZ: IGGPEXT** (calls **IGGPSCNC (IGG0CLAY)**)

Each CTGFL is initialized with the dictionary entry associated with the CTGFL's field-name value. Calls from within catalog management (as opposed to external calls, such as LOCATE) enter at this point to use the field management retrieval function.

2 **IGG0CLAZ: IGGPSCNF**

Steps 5 through 15 are performed for each of the caller's CTGFLs.

The Locate routine processes each CTGFL associated with the caller's CTGPL and returns as much caller-requested data (in the caller's work area) as the caller's test conditions and work area size permit.

3 **IGG0CLAZ: IGGPSCNF** (calls **IGGPUPGD**)

A caller may request catalog information from an associated upgrade entry by using upgrade field names.

4 **IGG0CLAZ: IGGPUGD**

The upgrade entry may not be in the catalog management buffer. If it is not in the buffer, the associations in the current entry are used to retrieve the upgrade entry.

5 **IGG0CLAZ: IGGPSCNF**

The caller's CTGFL list contains the address of each CTGFL required to satisfy the caller's need for catalog information. Each CTGFL describes one of the catalog record fields to be retrieved. Each CTGFL is completely processed before the next one is started.

**IGG0CLAZ: IGGPSCNF** (calls **IGGPTSTS (IGG0CLBA)**)

A caller might make conditional requests for retrieval of catalog record fields. In this case, two CTGFLs are supplied with the request and processed together. One CTGFL identifies a field to be retrieved and points to a second CTGFL that contains the name of the catalog field to be tested, the test conditions (equal, low, high, etc.), and the address and length of the caller's test data area. The catalog record field identified by the second CTGFL is compared to (tested against) the caller's data. If the comparison satisfies the test conditions, the catalog record field specified by the first CTGFL is retrieved.

6 **IGG0CLBA: IGGPTSTS**

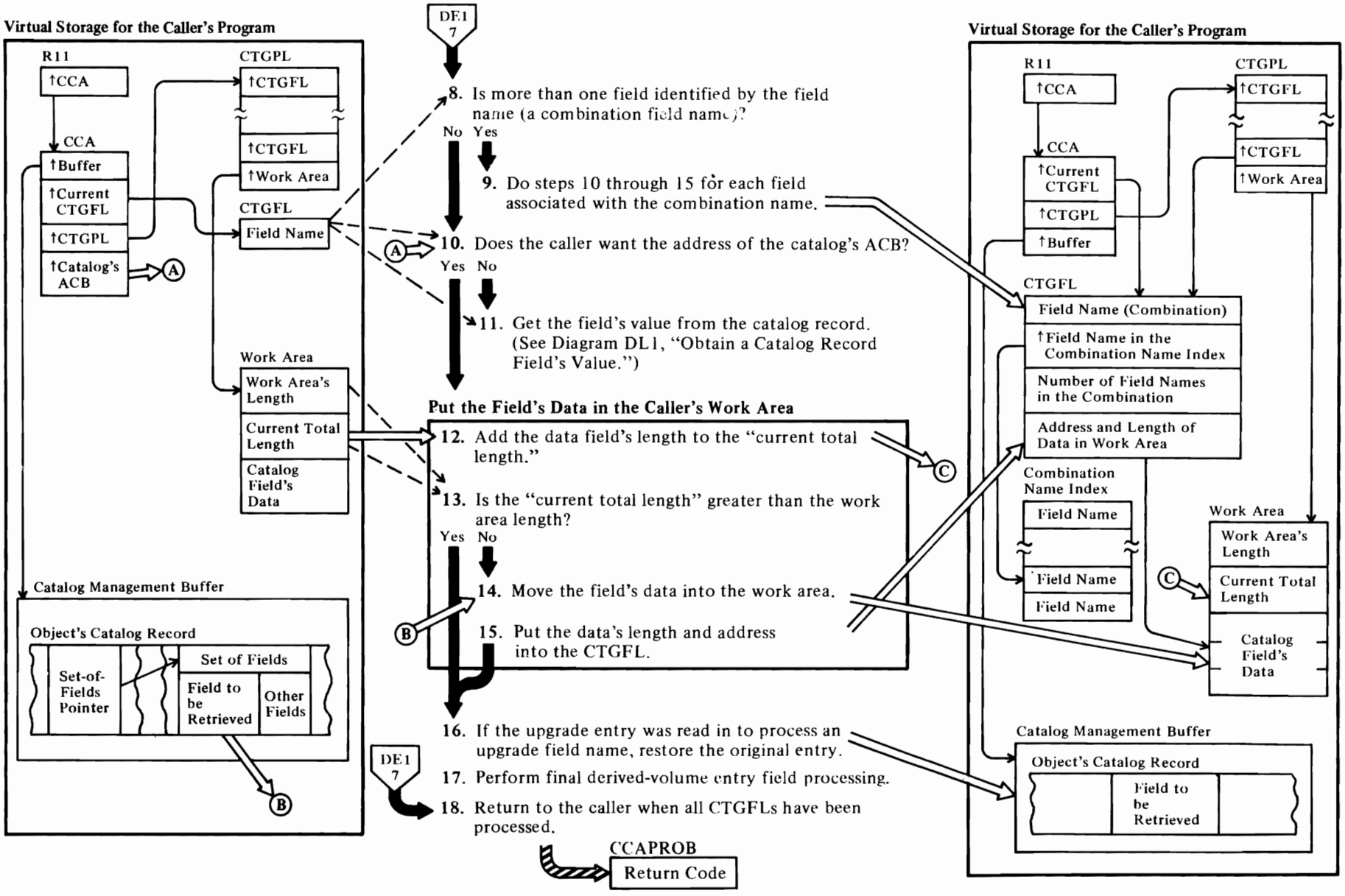
If the caller wants to retrieve a catalog record's header field, the field's data is retrieved if all tests are satisfied.

If the caller wants to retrieve a field from one of the sets of fields that follow the header fields, the field's data is retrieved from each set of fields that satisfies all tests.

See "Data Areas" for details about a catalog record and its sets of fields.

7 The sequence number of each set of fields that satisfies the tests is put in the CCA. After the sets of fields have been tested, the sequence numbers in the CCA are used to identify each set of fields that contain caller-requested data.

# Diagram DE2. LOCATE: Retrieve Catalog Information



## Notes for Diagram DE2

**8 IGG0CLAZ: IGGPLOC2**

A combination name refers to a set of related catalog field names, and is used by the caller instead of a separate CTGFL for each field name.

**9 IGG0CLAZ: IGGPLOC2**

The CTGPL, CTGFL, and catalog control area (CCA) are described in "Data Areas."

The combination name index has an entry for each field name in the combination. The Locate routine processes each field name entry in the combination name index sequentially, starting at the index of the first field name entry for the combination, and ending when the number of entries processed equals the number of field names associated with the combination name.

The test sequence (if any) associated with a combination-name CTGFL is done only once, not once for each field name in the combination.

**10 IGG0CLAZ: IGGPLOC2**

The address of the catalog's ACB is in the CCA. All other catalog record fields that the caller can request are in the catalog record. Each catalog record field is identified by its field name. See "Data Areas" for catalog record field names.

**11 IGG0CLAZ: IGGPLOC2** (calls **IGGPGVAL (IGG0CLBA)**)

Diagram DL1, Obtain A Catalog Record Field's Value, shows how the requested catalog record field (specified by its field name in the CTGFL) is located for the Locate routine.

**12 IGG0CLAZ: IGGPSHIN**

The first two fields in the caller's work area specify the number of bytes the caller allocated to the work area and the number of bytes that contain catalog record field data (the "current total length" exceeds the work area length, the current total length field is updated with the length of the catalog record data, but the data itself is not moved in the caller's work area.

**14 IGG0CLAZ: IGGPSHIN**

The Locate routine puts the beginning address and the length of the catalog field into the CTGFL's field-data entry.

**15 IGG0CLAZ: IGGPSHIN**

The CTGFL's field-data entry contains the beginning address and length of the data in the caller's work area. When control is returned to the caller, the caller can use the field-data entry to locate a specific field's data in the work area.

**16 IGG0CLAZ: IGGPSCNF**

If the field name processed required the upgrade entry to be read in, the original entry is restored before the next CTGFL is processed.

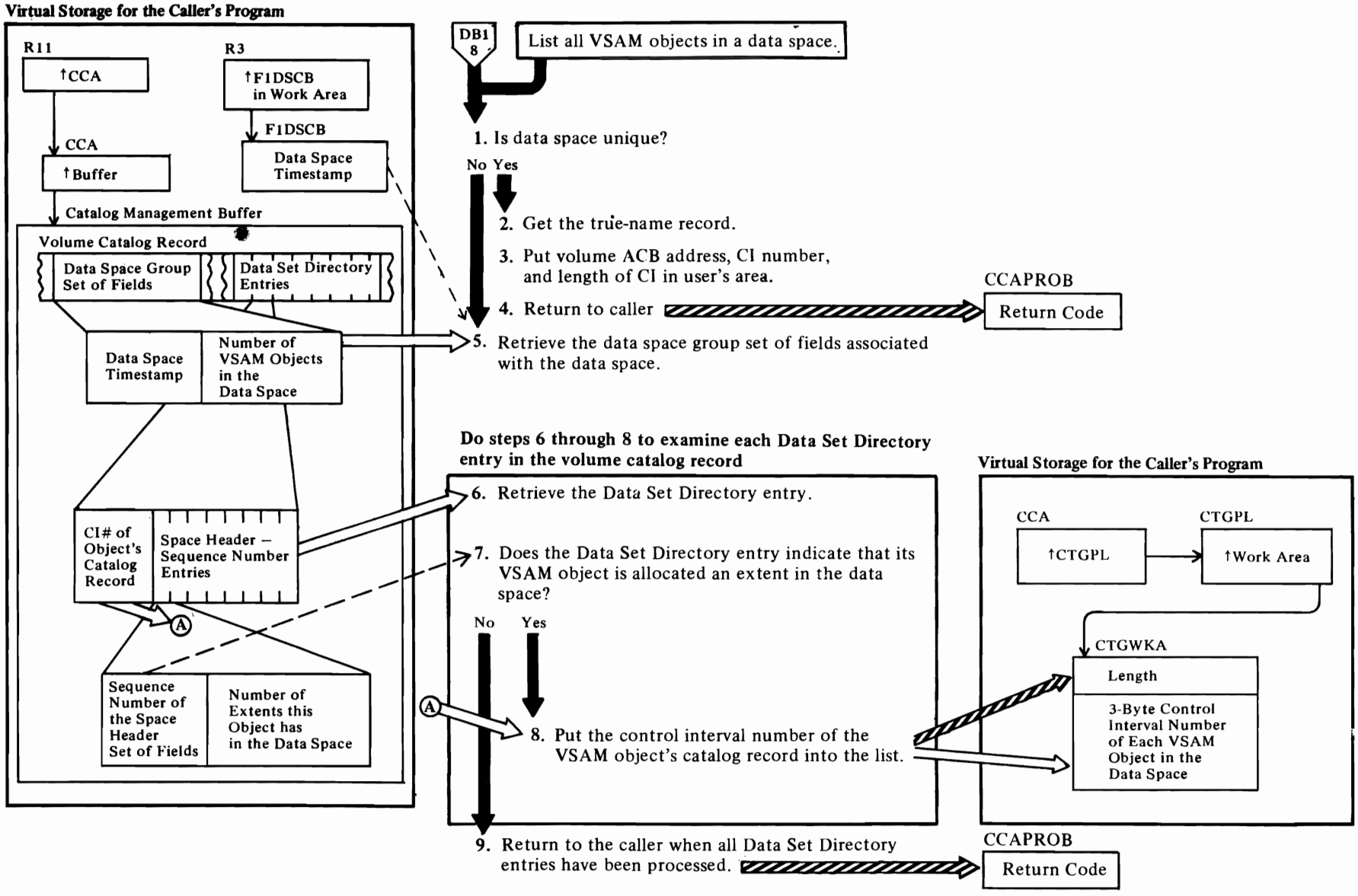
**17 IGG0CLAZ: IGGPEXT**

If this function was requested by an internal catalog management function, final derived-volume processing must be done. This processing consists of generating certain volume entry fields from the catalog information returned in the user's work area.

**18 IGG0CLAZ: IGGPLOC**

See "Diagnostic Aids" for details about catalog management return codes and error codes.

# Diagram DF1. GENDSP: List the Contents of a Data Space



## Notes for Diagram DF1

The caller (an OS/VS Utilities program or OS/VS Open) specifies the GENDSP option of LOCATE to obtain the control interval number of the catalog record of each object (cluster, data set, index, and catalog) that is contained in a VSAM data space identified by a DSNAME value (from the format 1 (identifier) DSCB).

**1 IGG0CLBJ: IGGPGDSP**

The user-provided workarea is tested to ensure that the minimum size has been provided. The GENDSP routine tests the first seven characters of the data space name to determine whether the data space is unique. A data space name beginning with "Z999999..." is a nonunique data space.

**2 IGG0CLBJ: IGGPGUDS**

The true-name catalog record associates the data space name with the control interval number of the catalog record that describes the data space.

**3 IGG0CLBJ: IGGPGUDS**

The fixed length of the control interval number area and the control interval number are put into the user-provided workarea.

**5 IGG0CLBJ: IGGPGDSP**

The CCA, a CPL, and two FPLs are set up to extract the data space group set of fields for the appropriate data space. The DSCB timestamp value is calculated from the data space name and used as the test value.

**6 IGG0CLBJ: IGGPGDSP**

Three FPLs are set up and the Data Set Directory entry is extracted.

**7 IGG0CLBJ: IGGPGDSP**

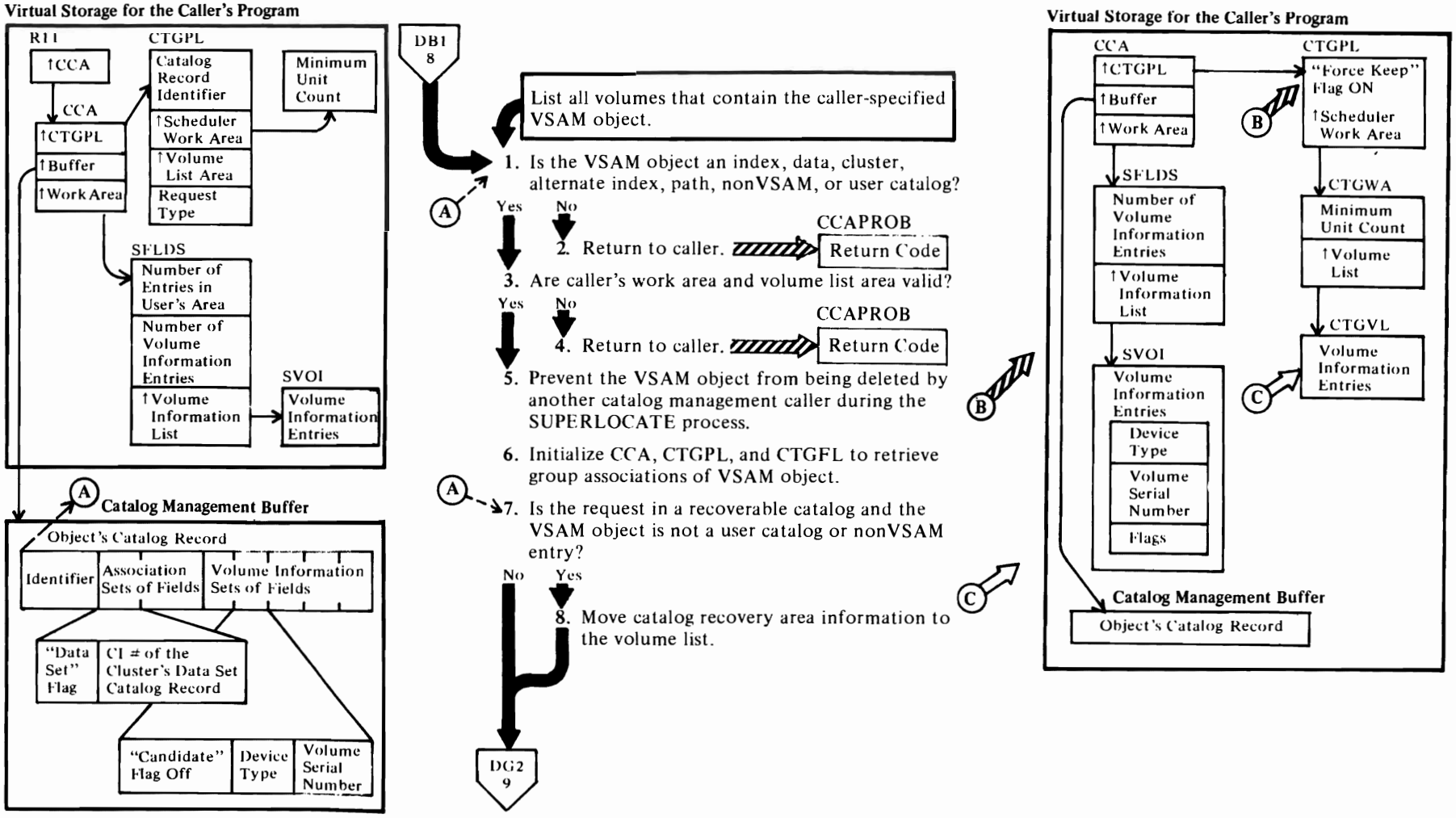
Scan the Data Set Directory set of fields to find a data space sequence number match.

**8 IGG0CLBJ: IGGPGDSP**

When a sequence number match is found, the volume ACB address, the length of the control interval number area, and the control interval number put into the caller's workarea.

See "Diagnostic Aids" for details about catalog management return codes and error codes.

# Diagram DG1. SUPERLOCATE: List a Data Set's Volumes



## Notes for Diagram DG1

The caller (the OS/VS Scheduler) specifies the SUPERLOCATE option of LOCATE to obtain a list of volume serial numbers (and device types) for a VSAM data set's volumes. The caller identifies the data set with its dsname value.

**1 IGG0CLAM: IGGPSLOC**

If the VSAM object is a cluster, the cluster's data volumes are described in the cluster's data set catalog record.If the cluster is key-sequenced, the cluster's index volumes are described in the cluster's index catalog record.

See "Data Areas" for details about cluster, data set, and index catalog records and their sets of fields.

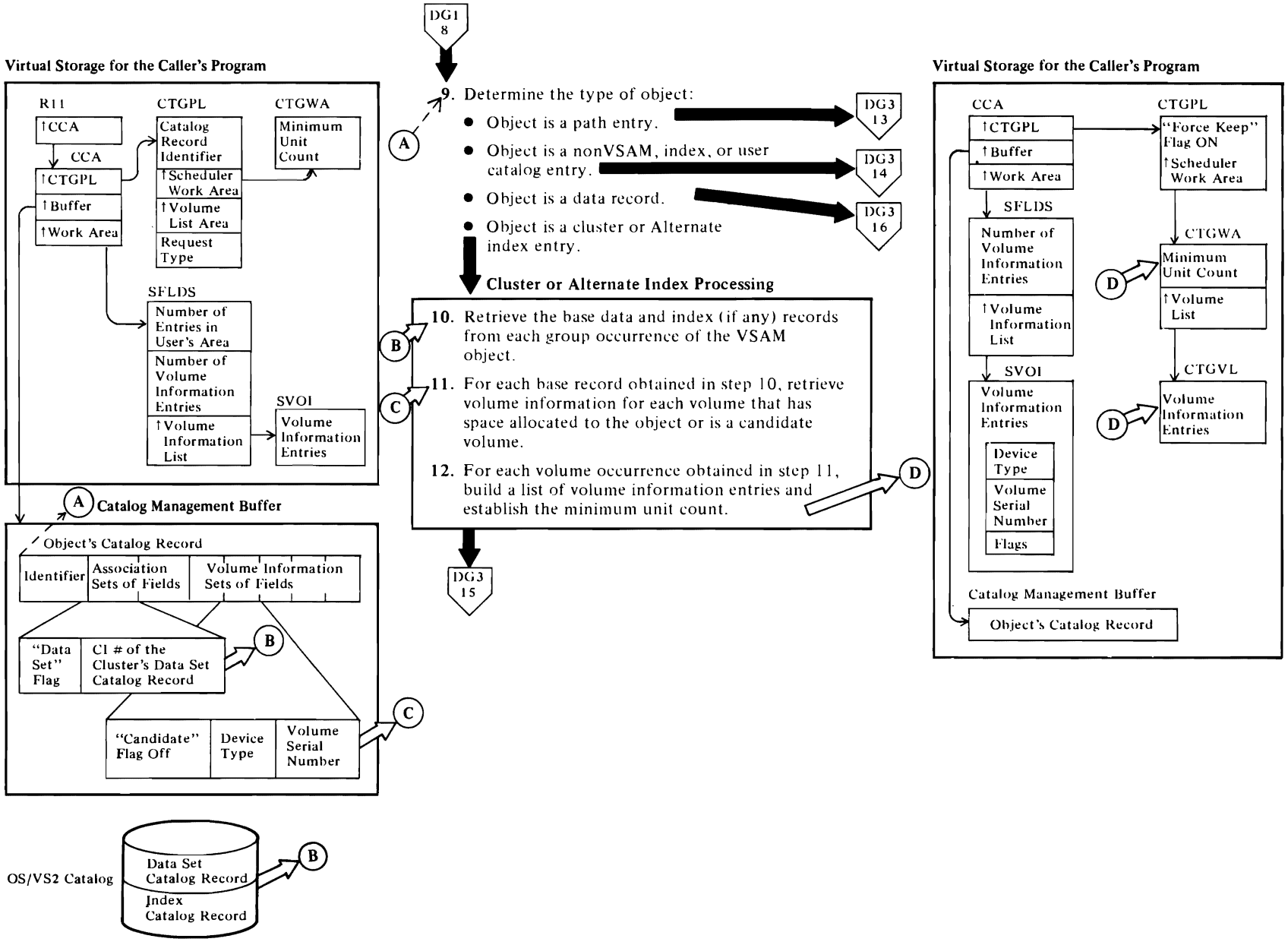
**3 IGG0CLAM: IGGPSLIN** and **IGGPDBVC**

**5 IGG0CLAM: IGGPSLIN**

**6 IGG0CLAM: IGGPSLIN** and **IGGPSLEI**

**7 IGG0CLAA: IGGPSLEN** and **IGGPSLIV**

# Diagram DG2. SUPERLOCATE: List a Data Set's Volumes



## Notes for Diagram DG2

**9 IGG0CLAA: IGGPSLEN**

**10 IGG0CLAA: IGGPSLCG**

If the VSAM object is a base cluster data record, call IGGPSLY (IGG0CLAA) to obtain upgrade associations, if any.

**11 IGG0CLAM: IGGPSLEL**

**12 IGG0CLAA: IGGPSLIV**

The volume list pointed to by CTGWAVL has the following format:

- The volume list contains no duplicate volume serial numbers.
- The volumes are divided by whether they are within the minimum unit count or outside it. Minimum unit count is the minimum number of direct-access devices required to mount the object's volumes. Volumes must be contiguous by device type. Device types within the minimum unit count are not ordered in any particular sequence nor are they related to the device types outside the count.
- Volumes within the minimum unit count will each be assigned an individual unit by the Scheduler. If volumes that do not have units already assigned exist outside the minimum unit count, the last unit assigned to a volume of the same device type within the minimum unit count will be made nonshareable. If this is not possible, an additional nonshareable unit will be assigned.

The volume list pointed to by CTGWAVL has the following content:

- All volumes in a given entry are placed into the volume list, regardless of whether they have allocated space.
- The volume information returned varies according to the entry type specified by the Superlocate request and whether the volume is within the minimum unit count or outside it, as follows:

  **Entry types C, D, G, I, and R**

  Within the minimum unit count, the CRA volume for the particular entry is returned.

  **Data entry (D):**

  Within the minimum unit count, every volume in the upgrade set is returned. Each volume in the data entry that has aunique device type within the data entry and is either the first with allocated space (prime or overflow) or, if no volumes have allocated space, is the first candidate volume is returned.

  Outside the minimum unit count, all others in the data entry are returned.

  **Index entry(I):**

  Within the minimum unit count, each volume in the index entry that has a unique device type within the index entry and also is either the first with allocated space (prime or overflow) or, if there are no volumes with allocated space, is the first candidate volume is returned. If sequence set is with data, the same volume may appear as both a prime and a candidate volume.

  Outside the minimum unit count, all others in the index entry are returned.

  **NonVSAM entry(A):**

  Within the minimum unit count, each volume in the nonVSAM entry that has a unique device type within the nonVSAM entry is returned. Every nonVSAM volume occurrence is marked as prime.

  Outside the minimum unit count, all others in the nonVSAM entry are returned.

  **User catalog entry (U):**

  Within the minimum unit count, each volume in the user catalog entry that has a unique device type within the entry is returned. Every user catalog volume occurrence is marked as prime.

  Outside the minimum unit count, all others in the user catalog entry are returned.

  **Base cluster entry (C):**

  Within the minimum unit count, every volume that does not have sequence set with data is returned. Otherwise, same as data entry.

  Outside the minimum unit count, all others in the data entry are returned.

  **Alternate index entry (G):**

  Same as the base cluster entry, except that there is never an upgrade set.

  **Alias path entry (R):**

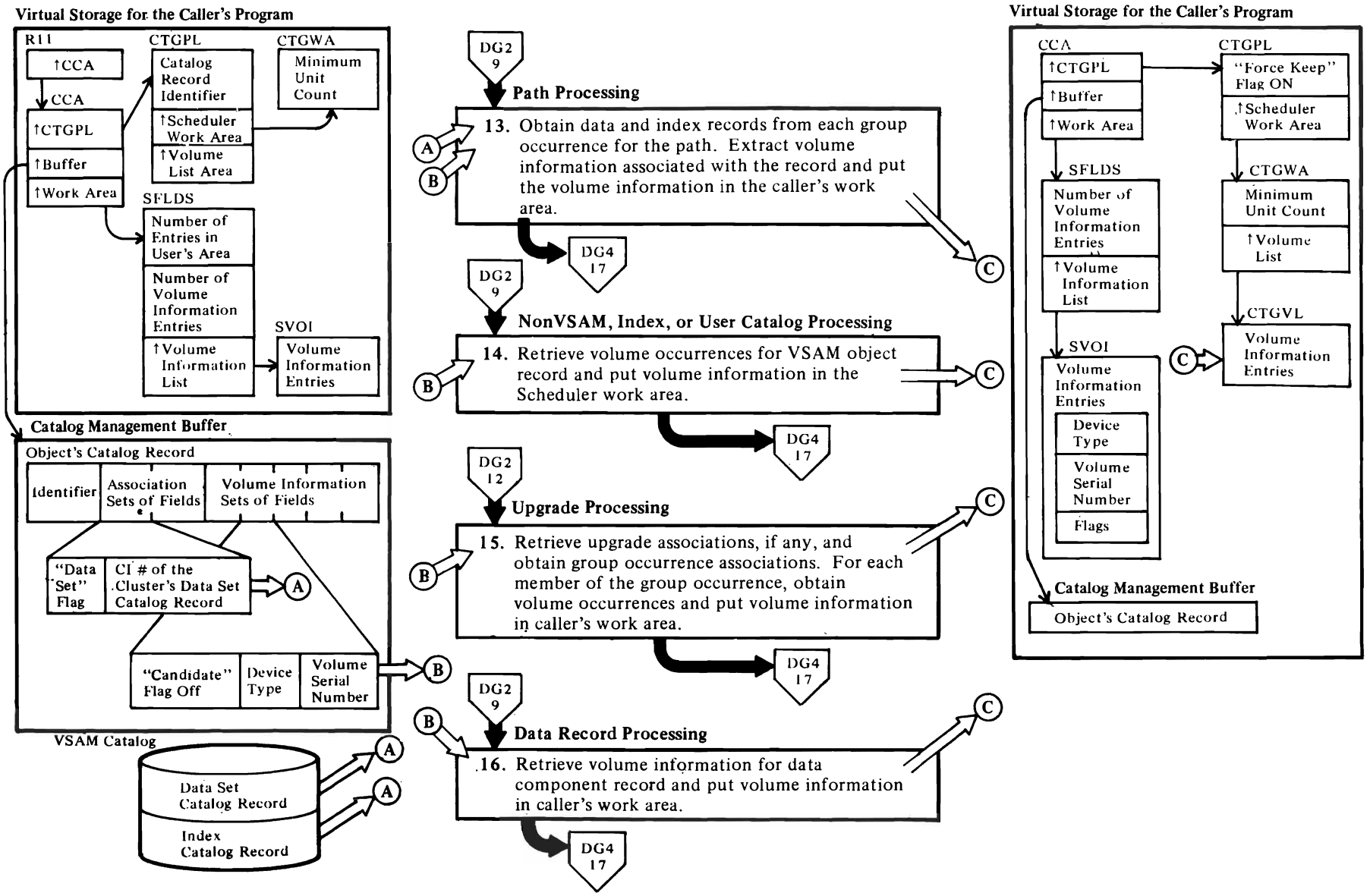
  Same as the base cluster entry, except that the upgrade set inclusion depends on the UPDATE/NOUPDATE flag in the path entry.

  **Normal path entry (R):**

  Within the minimum unit count, every volume of the alternate index under this path is returned. Otherwise, same as the base cluster entry, except that the upgrade set inclusion depends on the UPDATE/NOUPDATE flag in the path entry.

  Outside the minimum unit count, all others in the data entry are returned.

# Diagram DG3. SUPERLOCATE: List a Data Set's Volume



## Notes for Diagram DG3

**13 IGG0CLAA: IGGPSLR, IGGPSLIV,** and **IGGPSLY**

For each record obtained, IGGPSLEL extracts the volume information associated with the record. Then IGGPSLIV inserts the information into the caller's work area. When the base cluster data record has been retrieved, IGGPSLY obtains any upgrade associations related to the record.

**14 IGG0CLAA: IGGPSLIV**
**IGG0CLAM: IGGPSLEL**

IGGPSLIV inserts volume information into the caller's work area.

**15 IGG0CLAA: IGGPSLY** and **IGGPSLIV**
**IGG0CLAN: IGGPSLEL**

IGGPSLEL obtains the volume occurrences; IGGPSLIV inserts the volume information into the caller's work area.

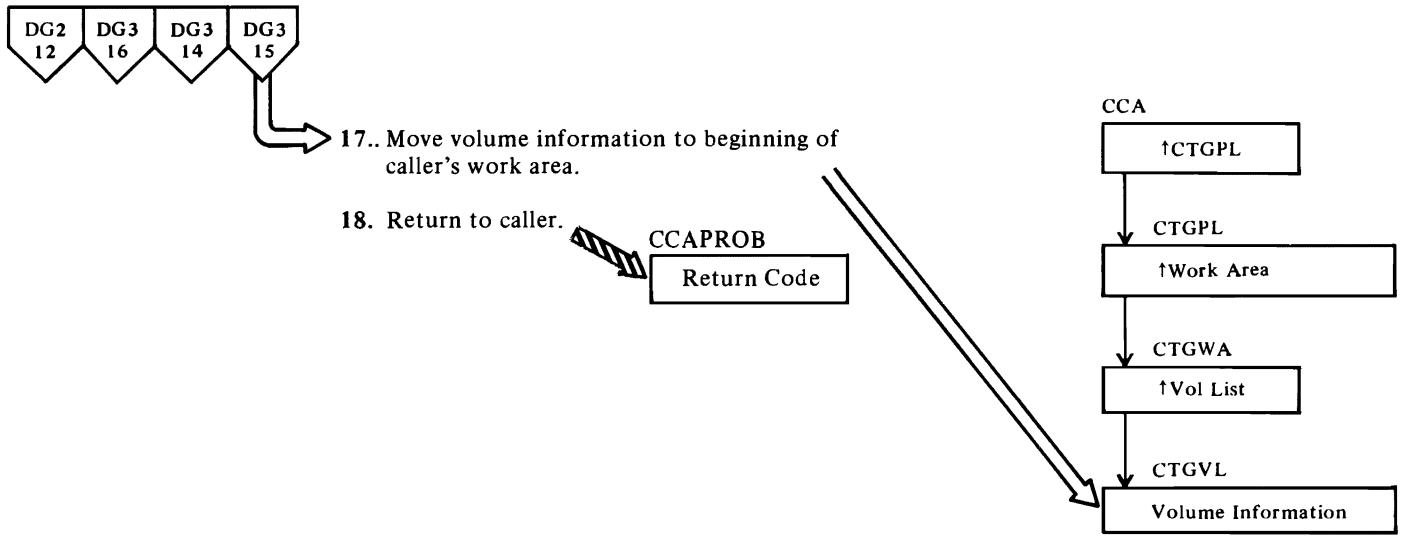
**16 IGG0CLAA: IGGPSLEN**

If the VSAM object is a base cluster data record, call IGGPSLY (IGG0CLAA) to obtain upgrade associations, if any.

**IGG0CLAM: IGGPSLEL** and **IGGPSLIV**

IGGPSLIV inserts volume information into the caller's work area.

# Diagram DG4. SUPERLOCATE: List a Data Set's Volumes



## Notes For Diagram DG4

**IGG0CLAA: IGGPSLEN and IGGPSLIV**

**17** IGGPSLIV builds the volume list from the end of the work area to the beginning, thus allowing the sorting of entries by device type both within and outside the minimum unit count.

**18 IGG0CLAM: IGGPSLOC**

If an error is detected, the procedure detecting the error returns control immediately to the calling procedure. IGGPSLOC returns to the caller of SUPERLOCATE.

For additional information about topics related to SUPERLOCATE processing, see:

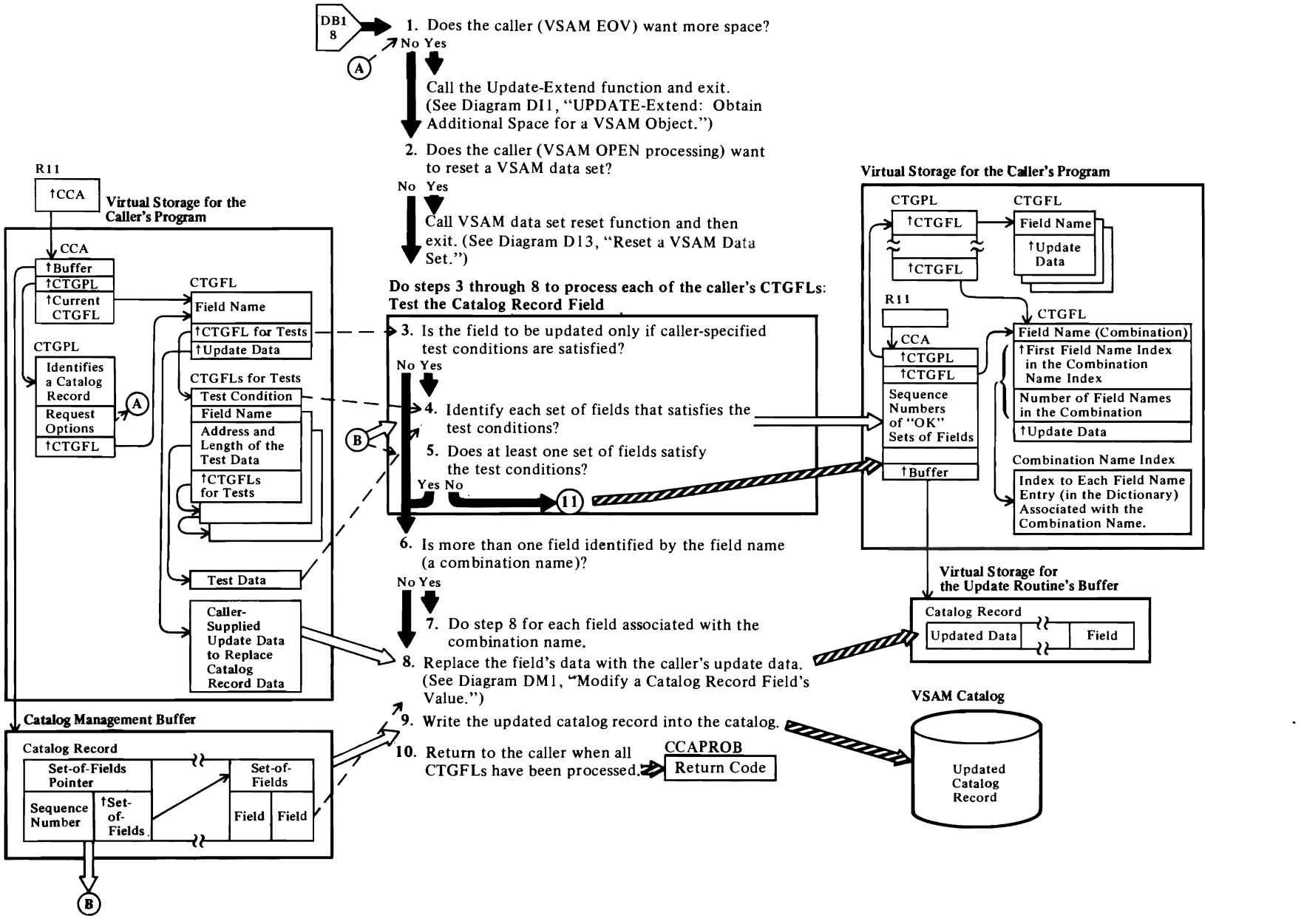
"Data Areas:"

Catalog record descriptions and formats

"Diagnostic Aids:"

Catalog management return codes

# Diagram DH1. UPDATE: Modify Catalog Information



## Notes for Diagram DH1

**IDA0192A**

The VSAM Open routine uses VSAM catalog management to reset a VSAM data set.

**IDA0200T**

The VSAM Close routine uses VSAM catalog management to modify the data set and index statistics maintained in the catalog record's copy of the AMDSB.

**IDA0557A**

The VSAM EOV routine uses VSAM catalog management to obtain more space for a data set.

**IGG0CLAB: IGGPACDV** (calls **IGGPUPD (IGG0CLAV)**)

When the caller issues the CATLG macro instruction, register 1 points to the caller's catalog parameter list (CTGPL).The CTGPL request options are decoded and the base catalog record is retrieved for the request. See Diagram DB1, VSAM Catalog Management Overview, for a description of initial catalog management processing and request decoding.

1 **IGG0CLAV: IGGPUPD** (calls **IGGPUPDE (IGG0CLBB)**)

If more space is required for the data set, the UPDATE—Extend routine processes the caller's update request, and returns to IGGPACDV in IGG0CLAB.

**IGG0CLAV: IGGPSFPL**

Steps 2 through 7 are performed to update each of the catalog record fields identified by the caller's CTGFLs.

2 **IGG0CLAV: IGGPUPD** (calls **IGGPRUS (IGG0CLB7)**)

If a VSAM data set must be reset, the UPDATE-REUSE routine processes the caller's request and returns to IGGPACDV (IGG0CLAB).

3 **IGG0CLAV: IGGPSFPL**

The caller's CTGFL list contains the address of each CTGFL needed to satisfy the caller's updating requirements. Each field parameter list (CTGFL) describes one of the catalog record fields to be updated. Each CTGFL is completely processed before the next one is started.

**IGG0CLAV: IGGPSFPL** (calls **IGGPTSTS (IGG0CLBA)**)

Sometimes the caller wants to update a field only if another field's value, when compared to the caller's test value, satisfies the caller's test conditions. If so, the caller builds a CTGFL that contains the name of the catalog field to be tested, the test conditions (equal, high, low, etc.), and the address and length of the caller's test value. If a CTGFL contains the address of another CTGFL, the second CTGFL describes a catalog record field that is to be compared to the caller's data. If the comparison satisfies the test conditions, the catalog record field specified by the first CTGFL is updated with the caller's data.

4 **IGG0CLBA: IGGPTSTS** and **IGGPTCMP**

If the caller wants to update a catalog record's header field, the field's data is updated with the caller's data if all tests are satisfied.

If the caller wants to update a field from one of the sets of fields that follow the header field, the field's data is updated with the caller's data for each set of fields' field that satisfies all tests. The set of fields that contains the field to be updated can also be identified by it's sequence number.

See "Data Areas" for details about the catalog record and its sets-of-fields.

The sequence number of each set of fields that satisfies the tests is put in the CCA.When all sets of fields have been tested, the sequence numbers are used to identify each set of fields that contains caller-requested data.

6 **IGG0CLAX: IGGPALT2**

A combination name refers to a set of related catalog field names, and is used by the caller instead of a separate CTGFL for each field name.

7 **IGG0CLAX: IGGPALT2**

The CTGPL, CTGFL, and catalog control area (CCA) are described in "Data Areas."

The CCA's combination name index has an entry for each field name in the combination. The Update routine processes each field name entry in the combination name index sequentially, starting with the index of the first field name entry for the combination, and ending when the number of entries processed equals the number of field names associated with the combination name.

The combination name's CTGFL contains the beginning address and the total length of the group of update data fields in the caller's work area.

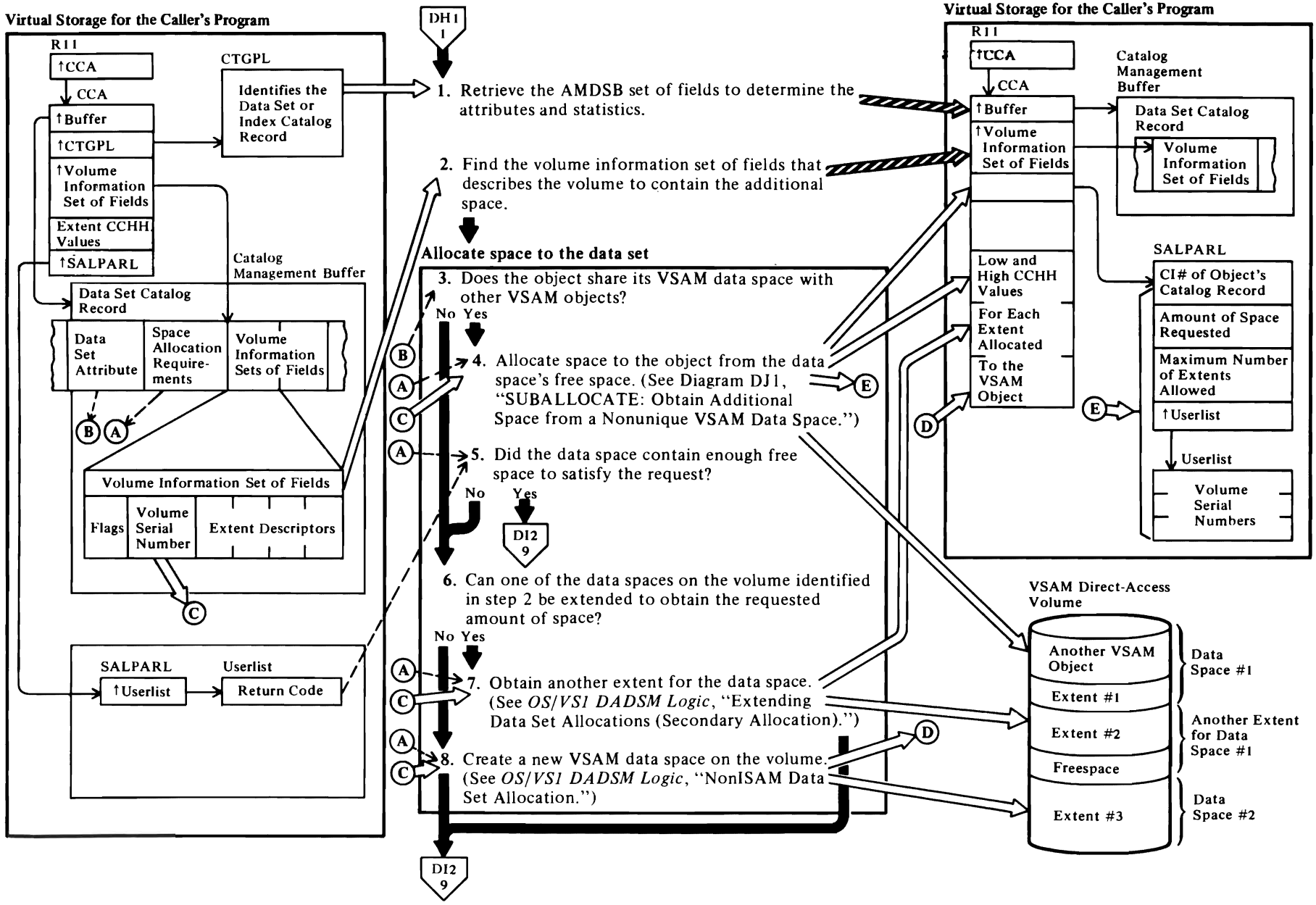
The test sequence (if any) associated with a combination-name CTGFL is done only once, not once for each field name in the combination.

9 **IGG0CLAV: IGGPSFPL** (calls **IGGPPREC (IGG0CLAW)**)

When the catalog record is updated (in a buffer in the Update routine's virtual storage) the update routine sets the "must write" flag on to indicate that the buffer must be written from virtual storage into the catalog before the buffer can be made available to contain another catalog record. When the caller's update request is finished, or when the Update routine needs the buffer to process another catalog record associated with the request, the Update routine calls IGGPPUPC or IGGPPAD (IGG0CLAG) to write the catalog record from the buffer into the VSAM catalog (on a direct-access storage device).

See "Diagnostic Aids" for details about catalog management return codes and error codes.

# Diagram DI1. UPDATE-Extend: Obtain Additional Space for a VSAM Object



## Notes for Diagram DI1

The UPDATE-Extend routine is called whenever a VSAM object (cluster, data set, index, or catalog) needs more space to store it's records.

The VSAM end of volume routine calls the catalog-management UPDATE routine, and an amount of space, based on the object's direct-access space allocation requirements, is allocated from one of the following sources:

- A shared VSAM data space that has enough free space to satisfy the allocation requirements
- A shared VSAM data space, extended to satisfy the allocation requirements from the free space on the object's currently mounted volume
- A new VSAM data space, created to satisfy the allocation requirements, the object's currently mounted volume.

**1 IGG0CLBB: IGGPUPDE** (calls **IGGPINIT (IGG0CLBC)**)

**2 IGG0CLBB: IGGPUPDE** (calls **IGGPSVOL (IGG0CLBC)**)

The volume information set of fields is identified by volume serial and key ranges, if this is a key-range data set. See "Data Areas" for details about the volume information set of fields.

**3 IGG0CLBB: IGGPUALL**

A shared (nonunique) VSAM data space contains all or parts of two or more VSAM objects. A unique VSAM data space contains all or part of only one VSAM object, and is not allowed to contain records of another object.

**4 IGG0CLBB: IGGPCSAL** (calls **IGGPSALL (IGG0CLAR)**)

If the object shares its data space with other VSAM data sets or indexes, there might be enough free space in one of the data spaces on the volume to satisfy the object's direct-access space allocation requirements.

**5 IGG0CLBB: IGGPUPDE**

If there is not enough free space, another extent is obtained for one of the volume's data spaces, or a new data space is created.

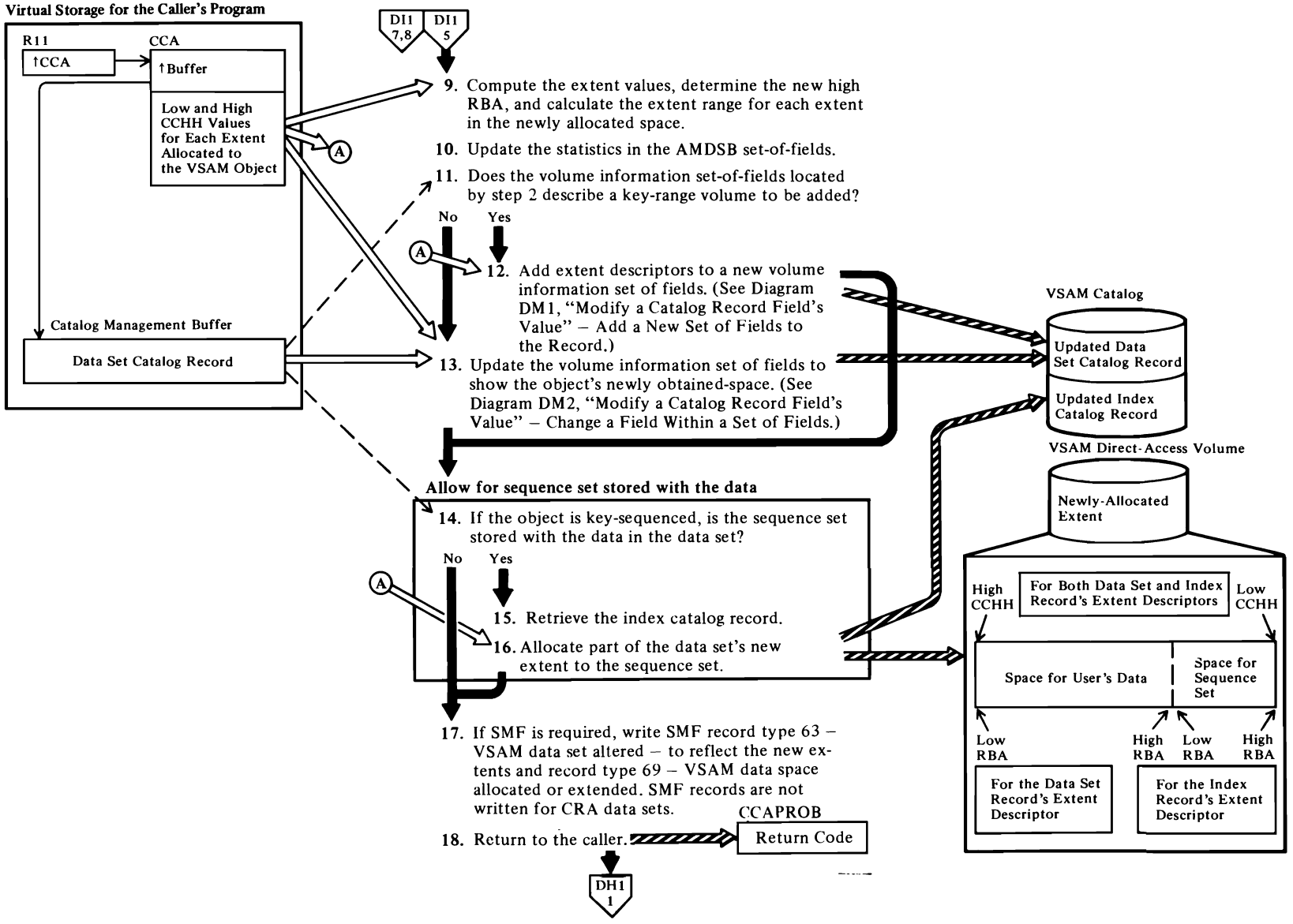
**6 IGG0CLBB: IGGPUPDE**

If any data space on the volume has less than 12 extents, the data space can be extended.

**8 IGG0CLBB: IGGPUPDE**

If a new extent was obtained for one of the volume's data spaces or a new data space was created, this space is suballocated to the object (nonunique) or given directly to the object (unique). For recoverable catalogs, the format-4 timestamp field is updated on the physical volume.

# Diagram DI2. UPDATE-Extend: Obtain Additional Space for a VSAM Object



## Notes for Diagram DI2

**11 IGG0CLBB: IGGPMVOL**

If a key range data set obtains space from a candidate volume, the second, third, fourth,... key range that obtains space from that candidate volume for the first time will require a new volume information set of fields. Note that each key range or each different volume is described by an individual volume information set of fields.

**12 IGG0CLBB: IGGPMVOL**

The object's catalog record contains a volume information set of fields to describe the object's space on each volume that contains a part of the object. If the object's newly obtained extent is in a key range on a new volume, the UPDATE-Extend routine may build a volume information set of fields to describe the new volume and extent. Otherwise, an existing volume information set of fields is updated with the high-allocated RBA and extent information in the form:

SS CCHH CCHH DDDD DDDD

where:

- SS identifies the VSAM data space.
- CCHH are the low and high cylinder and track addresses.
- DDDD are the low and high RBAs.

See "Data Areas" for details about the object's catalog record, and its for volume information set of fields.

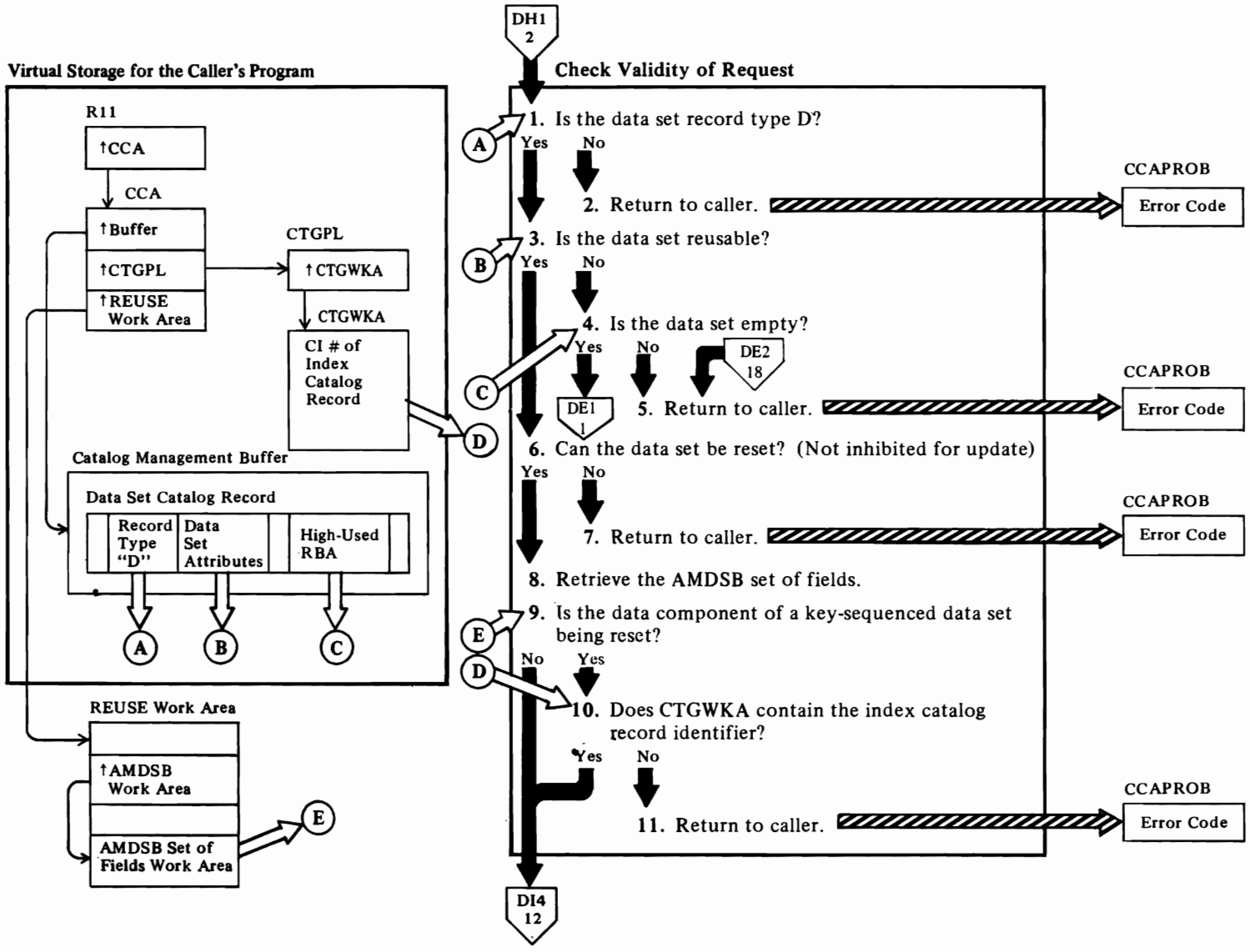
**16 IGG0CLBB: IGGPSSWD**

The low and high CCHH addresses (in the index catalog-record's volume information set of fields) are those of the extent obtained for the data set. The low and high RBA values are for the sequence set.

**18**

See "Diagnostic Aids" for details about catalog management return codes and error codes.

# Diagram DI3. REUSE: Reset a VSAM Data Set



## Notes for Diagram DI3

REUSABLE is an attribute that may be assigned to a VSAM data set or an alternate index. This attribute allows the data set to have its high-used RBA set to zero at open time, if the user specifies the RESET option. The indicator for REUSE is retained in the attributes field of data and index records of a key-sequenced data set and an alternate index and in the data record of an entry-sequenced and relative record data set. Reusable data sets may be multi-volumed and they must be suballocated only. That is, reusable data sets may not be unique. Also, reusable data sets cannot have key ranges and they are restricted to a maximum of 16 physical extents per volume. If a base cluster is defined as reusable, it may not have alternate indexes associated with it; however, it is permissible to define reusable alternate indexes that are related to a nonreusable base cluster.

**1** **IGG0CLAV: IGGPUPD** (calls **IGGPRUS (IGG0CLB7)**)

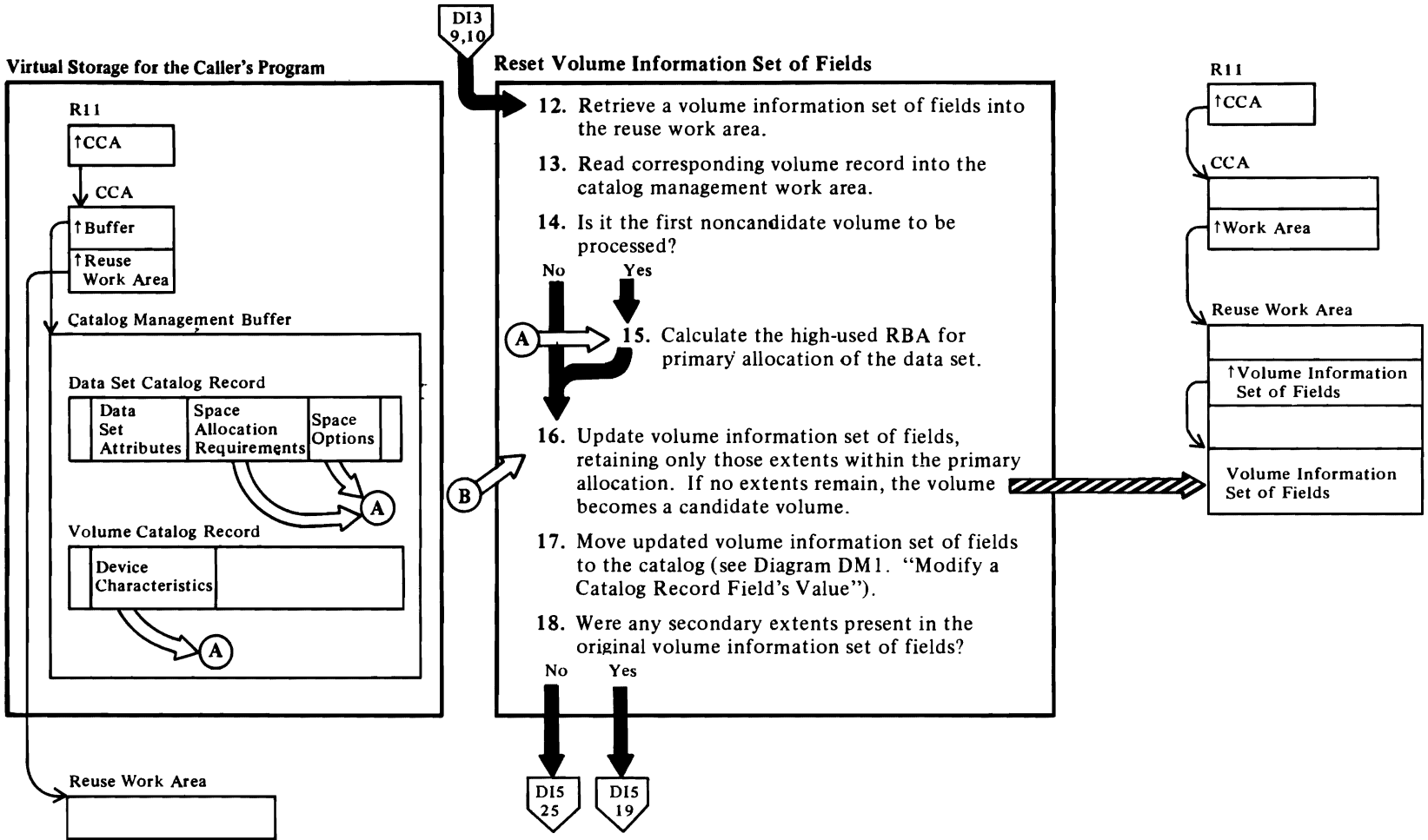
Initially, the data set catalog record is available in a catalog management buffer.

**4** If the data set is not reusable but is empty, the caller's LOCATE request is processed. No error codes are returned.

**8** **IGG0CLB7: IGGPRUS** (calls **IGGPEXT (IGG0CLAZ)**)

The reuse work area is set up by IGGPRUS and appropriate pointers are initialized. The reuse work area includes CTGPL and CTGFLs required by Modify and Extract Logic.

# Diagram DI4. REUSE: Reset a VSAM Data Set



## Notes for Diagram DI4

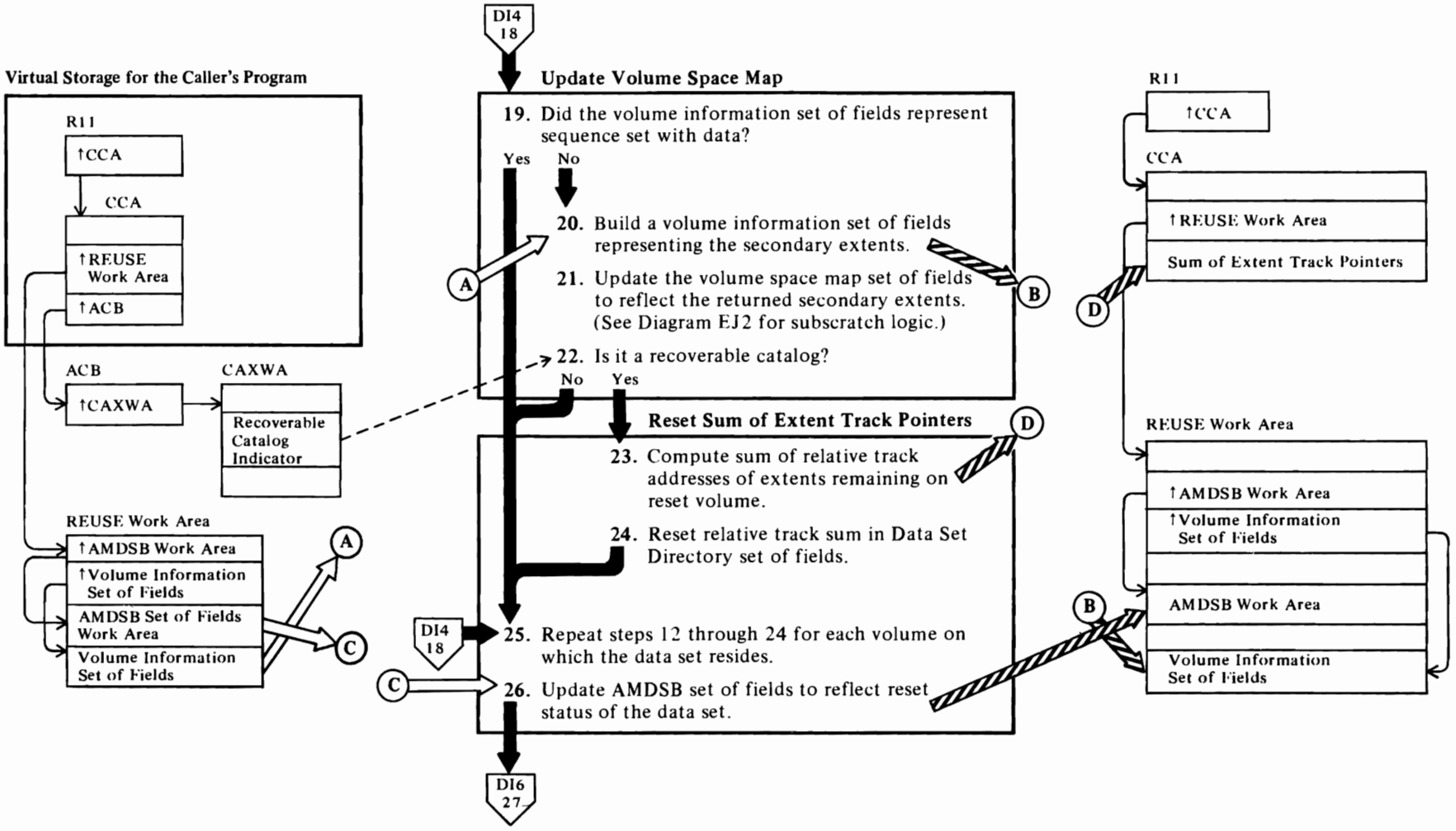
**12 IGG0CLB7: IGGPRUS** (calls **IGGPEXT (IGG0CLAZ)**)

The reuse work area provides space for a volume information set of fields that contains a maximum of 16 extent descriptors.

**13 IGG0CLB7: IGGPRUS** (calls **IGGPGET (IGG0CLBI)**)

**17 IGG0CLB7: IGGPRUS** (calls **IGGPMOD (IGG0CLAV)**)

# Diagram DI5. REUSE: Reset a VSAM Data Set



## Notes for Diagram DI5

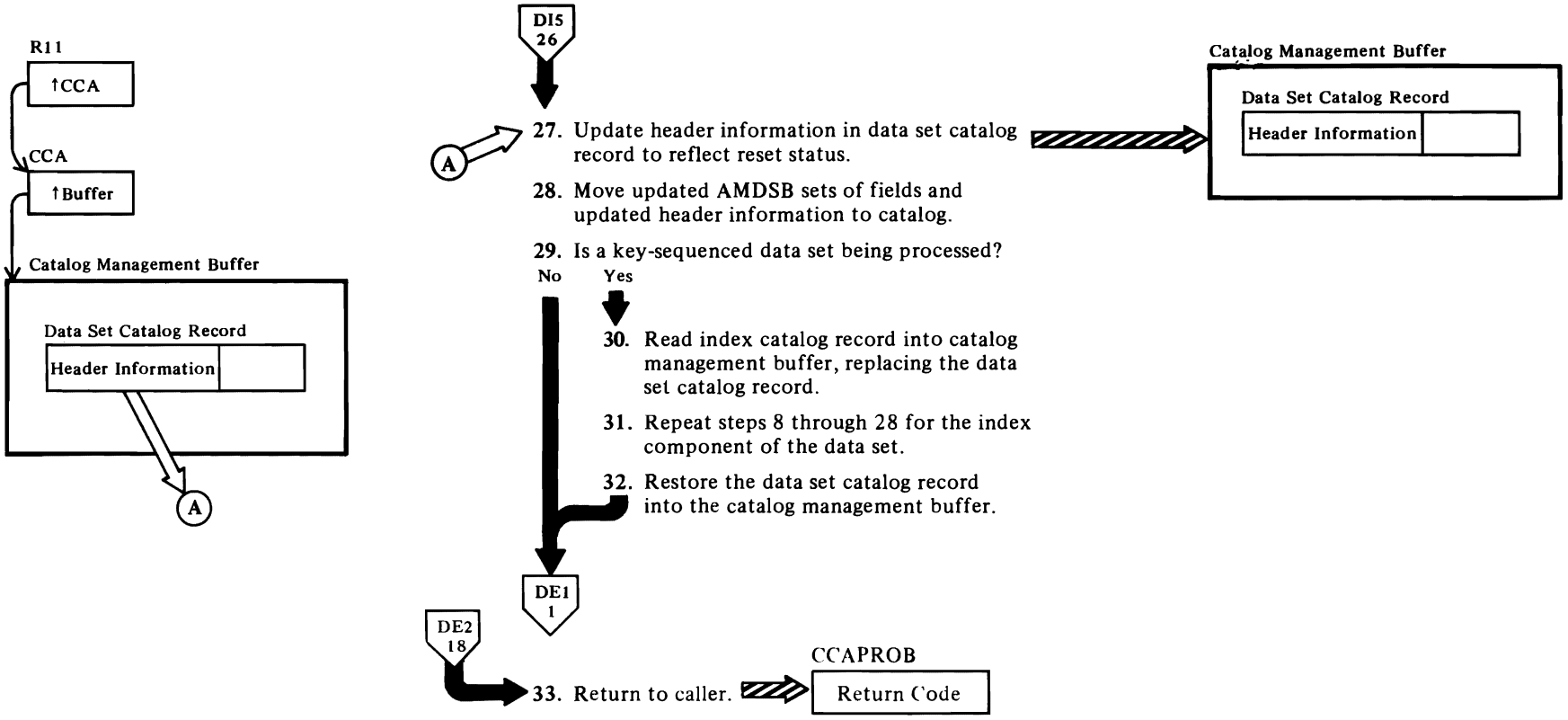
**19** The flag field in the volume information set of fields indicates whether sequence set is with data. If so, no resetting of the space map is necessary, since the space map will have been set already from the corresponding data set volume information set of fields. This is also true for the relative track sum in the case of a recoverable catalog.

**21** **IGG0CLB7: IGGPRUS** (calls **IGGPSSCR (IGG0CLBF))**

**23** **IGG0CLB7: IGGPRUS** (calls **IGGPTNXO (IGG0CLBI))**

If the reset volume is a candidate volume, the computed sum in the CCA will be zero.

# Diagram DI6. REUSE: Reset a VSAM Data Set



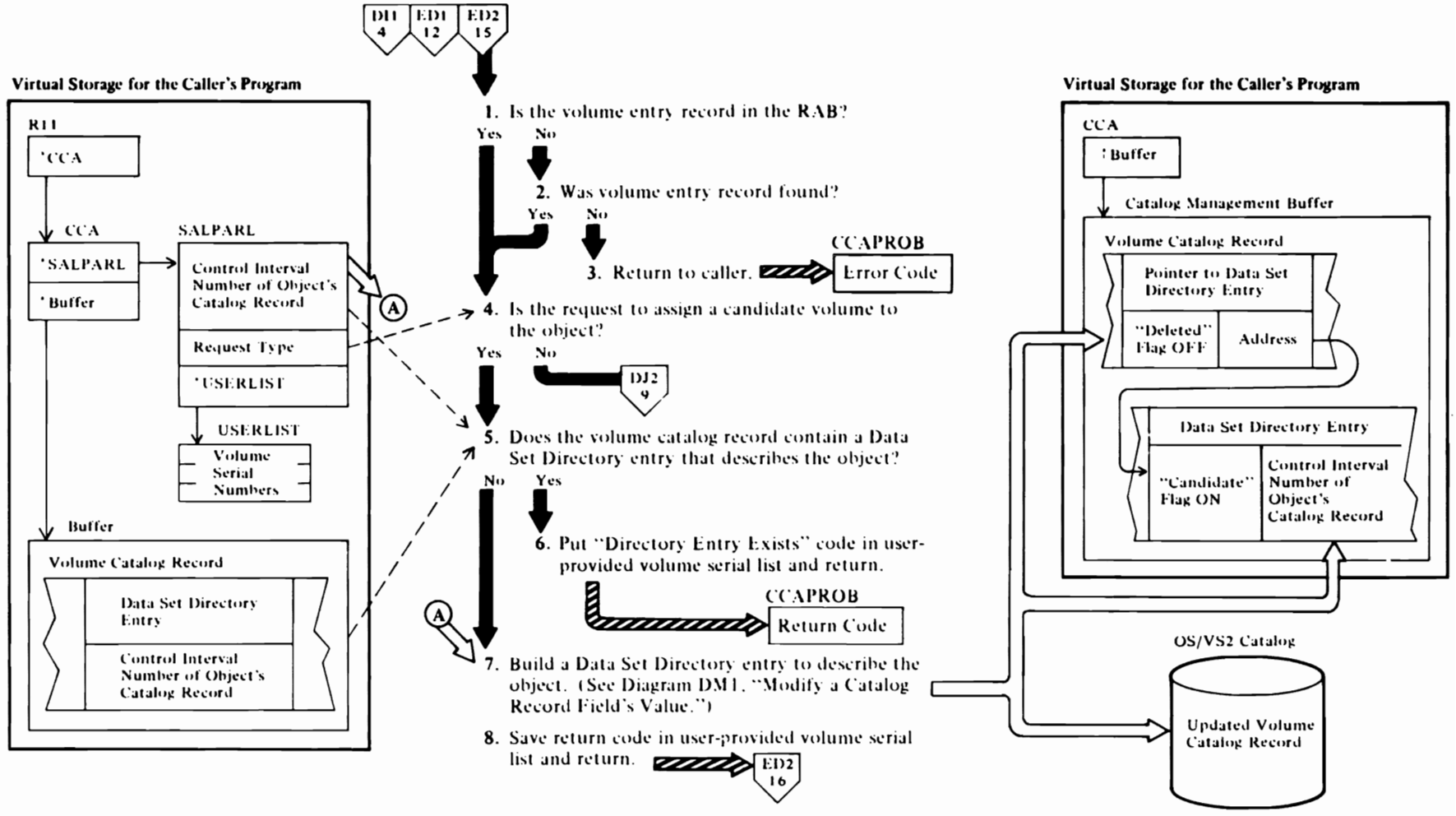
## Notes for Diagram DI6

**28 IGG0CLB7: IGGPRUS** (calls **IGGPMOD (IGG0CLAV)**)

**30 IGG0CLB7: IGGPRUS** (calls **IGGPGET (IGG0CLBI)**)

**32 IGG0CLB7: IGGPRUS** (calls **IGGPGET (IGG0CLBI)**)

# Diagram DJ1. SUBALLOCATE: Obtain Additional Space from a Nonunique-VSAM Data Space



## Notes for Diagram DJ1

The Suballocate routine is called to assign a candidate volume to a VSAM object (cluster, data set, index, or catalog) and to assign available space to a VSAM object from one of the data spaces on the caller-specified volume. The caller, either the UPDATE-Extend routine (see Diagram DI1) or the DEFINE CLUSTER routine (see Diagram ED1), builds a list of volume serial numbers to identify each volume to be assigned to the object as a candidate volume. If the caller requests space allocated to the object, the list contains one volume serial number.

**1 IGG0CLAR: IGGPSALL**

The volume entry record may already exist in the RAB, having been put there by the caller of suballocate.

**2 ICC0CLAR: ICCPSALL**

If the volume entry record is not in the RAB, a call is made to IGGPGET (BI) to get the record.

**4 IGG0CLAR: IGGPSALL**

If the request is to assign available space to an object from a specified volume, IGGPSALL calls IGGPSALS (AU).

**5 IGG0CLAR: IGGPSALL**

If the volume catalog record already contains a data set directory entry set of fields, the volume either is already assigned to the VSAM object as a candidate volume or has some of its space allocated to the VSAM object.

**6 IGG0CLAR: IGGPSALL**

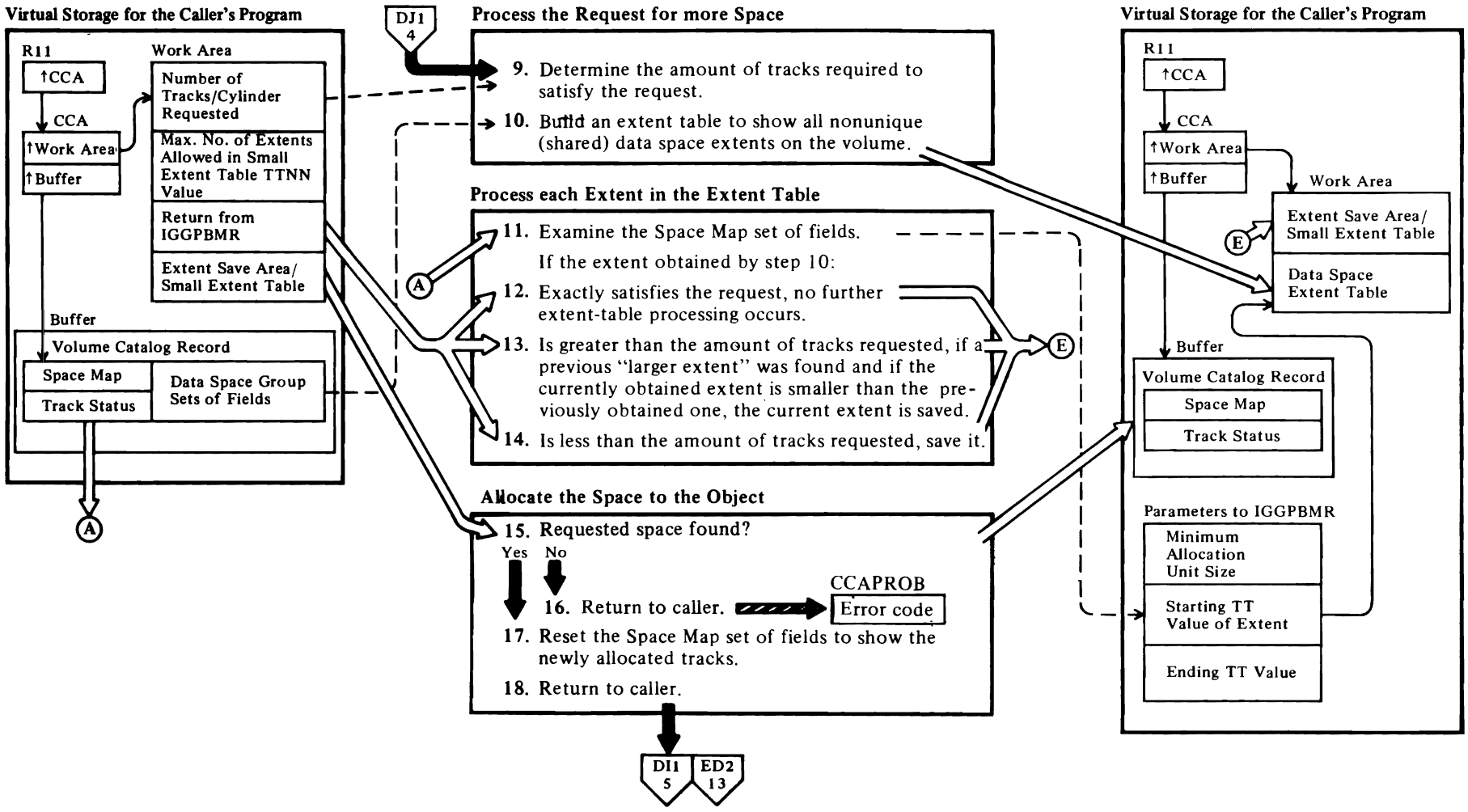
If a Directory Entry already exists for the data set, the return code is set in the user-provided volume serial number list.

**7 IGG0CLAR: IGGPSALL**

IGGPSALL calls IGGPISCJ and IGGPMOD to add the new data set directory entry to the volume catalog record.

See "Data Areas" for details about the data set directory entry set of fields.

# Diagram DJ2. SUBALLOCATE: Obtain Additional Space from a Nonunique VSAM Data Space



## Notes for Diagram DJ2

**9 IGG0CLAU: IGGPSALS**

If the amount of space requested is a number of cylinders, convert it to a number of tracks.

**10 IGG0CLAU: IGGPSALS** (calls **IGGPEXT (IGG0CLAZ)**)

The extent table is built by retrieving each extent descriptor (from each data space group set of fields) that might contain enough free space to satisfy the request's minimum allocation requirement (the number of tracks in one control area).

All extents of shared data spaces are described in the table until either there are no more extents to describe or the table is full.If the table is full, step 10 is repeated when steps 11 through 14 are completed, until the extents of all shared data spaces have been examined.

**11 IGG0CLAU: IGGPEDS** (calls **IGGPBMR (IGG0CLBR)**)

Each extent descriptor in the extent table is in the form:

S# TT NN

where:

- S# is the sequence number of the data space's extent.
- TT is the extent's starting track number.
- NN is the number of tracks in the extent.

The extent descriptors are processed beginning with the lowest TT value in the table, then the next lowest, etc., until all extent descriptors have been processed.

IGGPBMR examines each extent to find an amount of contiguous unallocated tracks at least as large as the request's minimum allocation unit. IGGPBMR examines the Space Map set of fields, starting at bit position (track indicator) TT and ending at bit position (track indicator) TT+NN-1 (usually the extent's track boundaries). If IGGPBMR finds a large enough amount of unallocated tracks, it returns to IGGPEDS with the beginning track number (TT) and the number of tracks (NN). If the data space's extent might contain another amount of unallocated tracks at least as large as the request's minimum allocation unit, IGGPEDS calls IGGPBMR again to examine the rest of the data space's extent.

**12 IGG0CLAU: IGGPEDS**

If the extent returned by IGGPBMR is the exact number of tracks required to satisfy the caller's request, no further extent table processing is done. Larger or smaller extents obtained from previous extent-table entries are ignored.

**13 IGG0CLAU: IGGPEDS**

If the extent returned by IGGPBMR is larger than the amount of tracks required to satisfy the request, the extent is saved if either:

- No other "larger-than-requested-amount" extent has been returned yet, or
- The current extent is smaller than a previously obtained "larger-than-requested-amount" extent.

MONDAY AUG 23 - merged TNL with base (parts 1 and 2) and sent to printq.

In either case, only one "larger-than-requested-amount" extent value is saved. The "small extent table" (built in step 13) is ignored and no longer used.

**14 IGG0CLAU: IGGPEDS**

If the extent returned by IGGPBMR is smaller than the amount of tracks required to satisfy the request, its TTNN value is adjusted so that TT is on a cylinder boundary. If NN is now at least as large as the request's minimum allocation unit (number of tracks for one control area), the extent is saved in the "small extent table" if:

- The table has fewer than five entries (or a caller-specified maximum less than five), or
- The table is full and the current extent's NN value is greater than the table's smallest extent's NN value. The current extent replaces the table's smallest extent.

In either case, the extent is not put in the "small extent table" if it is too small (adjusted NN is less than the minimum allocation unit) or if a "larger-than-requested-amount" extent already exists (see step 12).

If, after all data spaces have been examined, the total of the NN values in the "small extent table" is less than the amount required to satisfy the request, no space is allocated to the object.

**17 IGG0CLAU: IGGPSALS** (calls **IGGPBMR (IGG0CLBR)**)

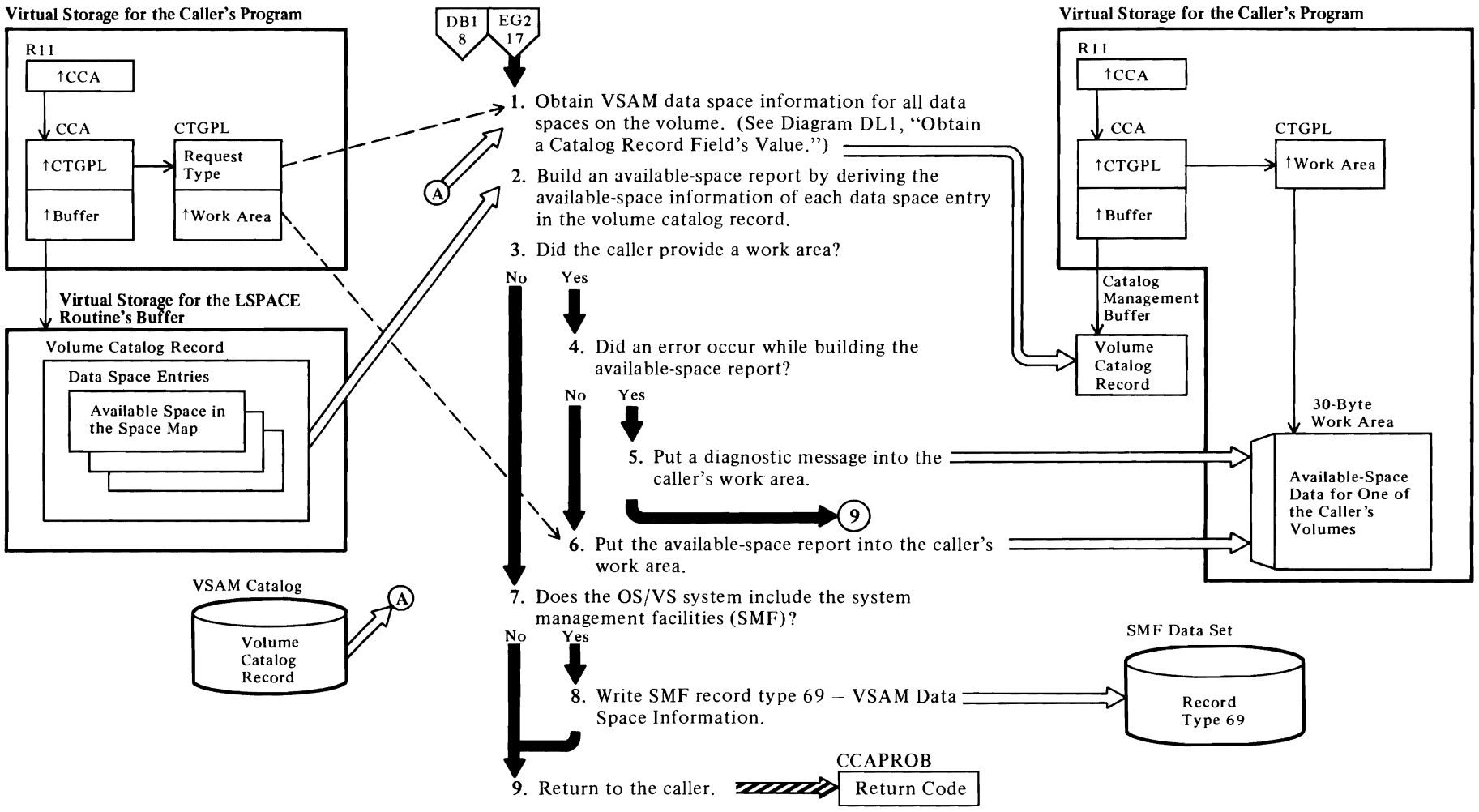
If the selected extent is larger than or equal to the amount of space requested, IGGPBMR adjusts the Space Map set of fields starting at bit position (track indicator) TT, turning off NN bits (NN is the exact number of tracks required to satisfy the request).

If the space is allocated to an object from a number of extents, the "small extent table" is sorted so that the largest NN value is first, the smallest last. IGGPBMR then adjusts the Space Map set of fields for each TTNN value in the "small extent table," until the amount of allocated tracks equals the amount of tracks requested.

**18 IGG0CLAU: IGGPSALS**

IGGPSALS returns the sequence number of the data space's extent, starting track number, and number of tracks for each extent obtained for the request. The caller uses this information to build extent descriptor entries in the VSAM object's volume information set of fields.

# Diagram DK1. LSPACE: Build an "Available Space" Report



## Notes for Diagram DK1

**1 IGG0CLBK: IGGPLSP** and **IGGPLDCE**

The volume catalog record describes each VSAM data space, and its free space, on the volume.

See "Data Areas" for details about the volume catalog record.

**2 IGG0CLBK: IGGPLDCS, IGGPLSMS, IGGPLDAS**

Each data space entry derived from the volume catalog record has a field that describes the available space in the data space. The LSPACE routine analyzes each data space entry and calculates the amount of available space in cylinders and tracks. It also records the number of cylinders and tracks in the longest continuous amount of available space.

**5 IGG0CLBK: IGGPLEMP**

A diagnostic message describing the error which occurred during the building of the available-space report is placed in the caller's work area.

**6 IGG0CLBK: IGGPLDCE, IGGPLSMP**

If the caller provides a 30-byte work area, the available-space report is put into the work area in the form:

SPACE-CCCC,TTTT,AAAA/cccc,tttt

where:

- CCCC is the number of cylinders of free space in all VSAM data spaces on the volume.
- TTTT is the number of tracks in addition to the number of cylinders of free space in all VSAM data spaces on the volume.
- AAAA is the number of extents of free space in all VSAM data spaces on the volume.
- cccc is the largest number of contiguous freespace ce cylinders on the volume.
- tttt is the number of contiguous free-space tracks in addition to the cccc value.

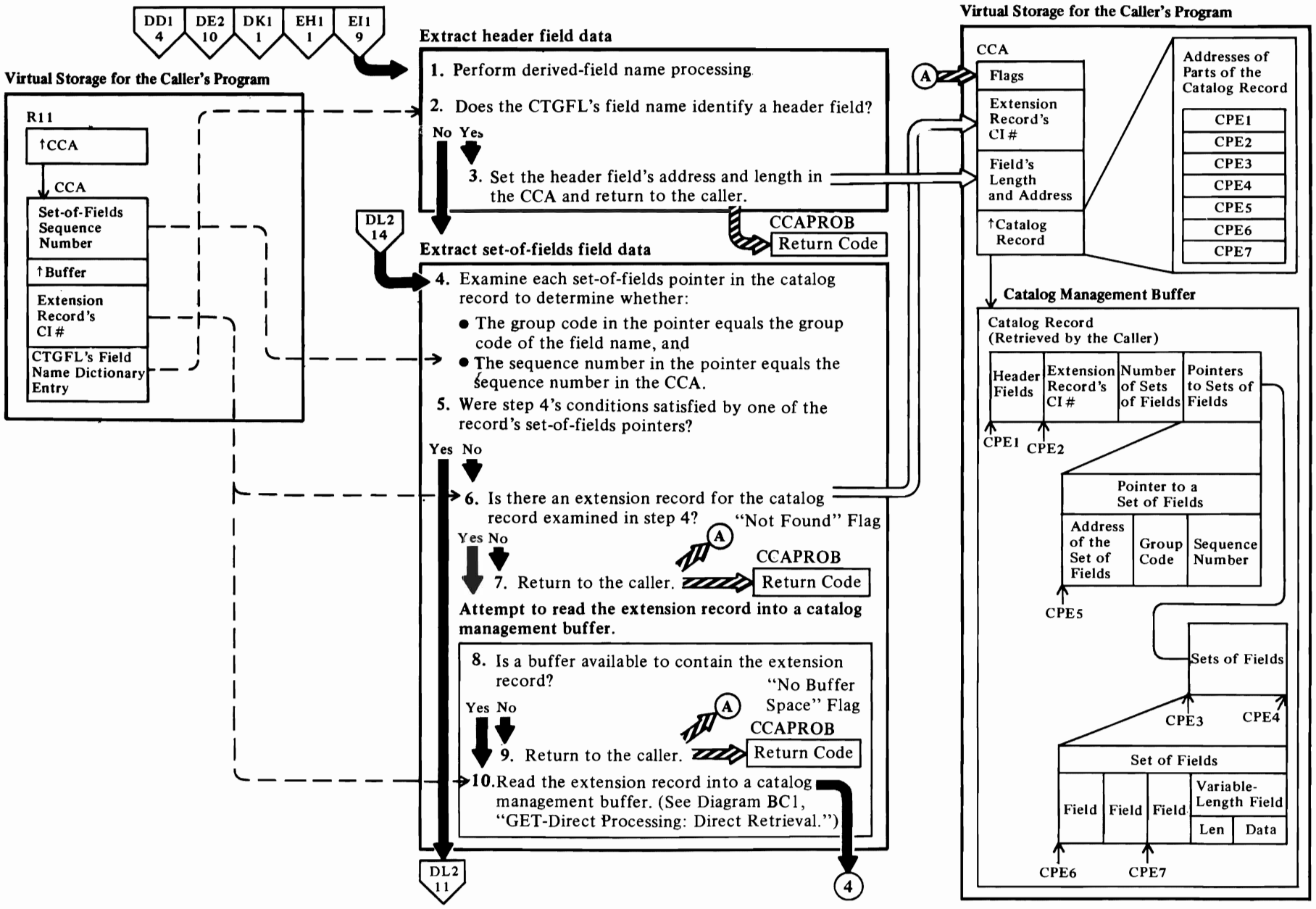
**8 IGG0CLBK: IGGPLSP** (calls **IGGPSMFL (IGG0CLBV)**)

See *OS/VS System Management Facilities (SMF)* for SMF record type 69 details.

**9 IGG0CLBK: IGGPLSP**

See "Diagnostic Aids" for details about catalog management return codes and error codes.

# Diagram DL1. Obtain a Catalog Record Field's Value



## Notes for Diagram DL1

The Get-Field-Value routine is called by other catalog field management routines to obtain the location and length of a field in a catalog record. The record is in virtual storage in a catalog management buffer. The following results could occur:

- The field is entirely contained in the record in the buffer, and the field's address and length are set in the caller's CCA.
- The field is partially contained in the record in the buffer, and the field's address and partial length are set in the caller's CCA. The CCA also has the "not complete" flag on and contains the control-interval number of the catalog record's extension, which contains more of the field.
- The field is not retrieved because it doesn't exist in the caller-specified set of fields, or because there are no more sets of field in the record, or because no buffer space is available to contain an extension record.

**1 IGG0CLBS: IGGPXVAL**

If the field name is derived, the field information to be returned does not exist in the physical catalog record but must be generated from the catalog fields, possibly in different catalog records. Derived field names exist only in the volume entry.

**2 IGG0CLBA: IGGPLVAL**

The field-name dictionary is a read-only catalog management table. The catalog field name dictionary contains an entry for each type of catalog record field, based on it's field name. The caller puts the dictionary entry identified by the CTGFL's field name into the CCA before calling the get-field-value routine.

Header fields are identified by a type-code of 0 (in their dictionary entry). A nonzero type-code identifies a set of fields that contains the field identified by the CTGFL's field name.

See "Data Areas" for details about the catalog field names and dictionary entry format.

**3 IGG0CLBA: IGGPLVAL**

Header fields are fixed-length and located at a fixed displacement from the beginning of the catalog record.

The field's address is obtained by adding the displacement (in the CTGFL's dictionary entry) to the beginning address of the record (the CCA's CCACPE1 value). The field's length is part of CTGFL's dictionary entry.

If the field name identifies a header field, and the field is variable length, the field's address is obtained by using the sequence number in the CTGFL's dictionary entry, which indicates that the field is the first, second, etc. variable-length field.

See "Data Areas" for details about the catalog record header fields and the catalog communications area (CCA).

**4 IGG0CLBA: IGGPLVAL**

The set of fields pointer (GOP) is used to locate a set of fields. The GOPs are grouped together by type code. Within each group of GOPs, the pointers are ordered by sequence number.

See "Data Areas" for details about the catalog record and set of fields pointer.

**5 IGG0CLBA: IGGPLVAL**

If the caller-specified set of fields pointer identified by it's sequence number is found, its displacement field and flags field specify the location of its set of fields as:

- the number of bytes from the beginning of the record's sets of fields (the CCA's CCACPE3 value plus the set-of-fields pointer's displacement field value), or
- the control interval number of the extension record that contains the set of fields. The extension record contains a set-of-fields pointer that specifies the set of field's location as a number of bytes from the beginning of the record's sets of fields.

See "Data Areas" for details about the catalog record, extension record, and set of fields pointer.

**6 IGG0CLBA: IGGPGVAL**

**7 IGG0CLBA: IGGPGVAL**

See "Diagnostic Aids" for details about catalog management return codes and error codes.

**8 IGG0CLBA: IGGPGREC**

Each catalog record (in a catalog management buffer) is identified by a record area block (RAB) within the CCA. The RAB contains flags that indicate whether or not the buffer can be used to contain another record. If the RAB's "must write" flag is on, the buffer cannot be used for another record until its contents have been written into the catalog.

See "Data Areas" for details about the CCA.

**IGG0CLBA: IGGPGVAL**

Each catalog control area (CCA) contains six record area blocks. Each catalog management request can use a maximum of five buffers. If all buffers are filled and cannot be released, the get-field-value routine sets the CCA's "no buffer space" flag on.

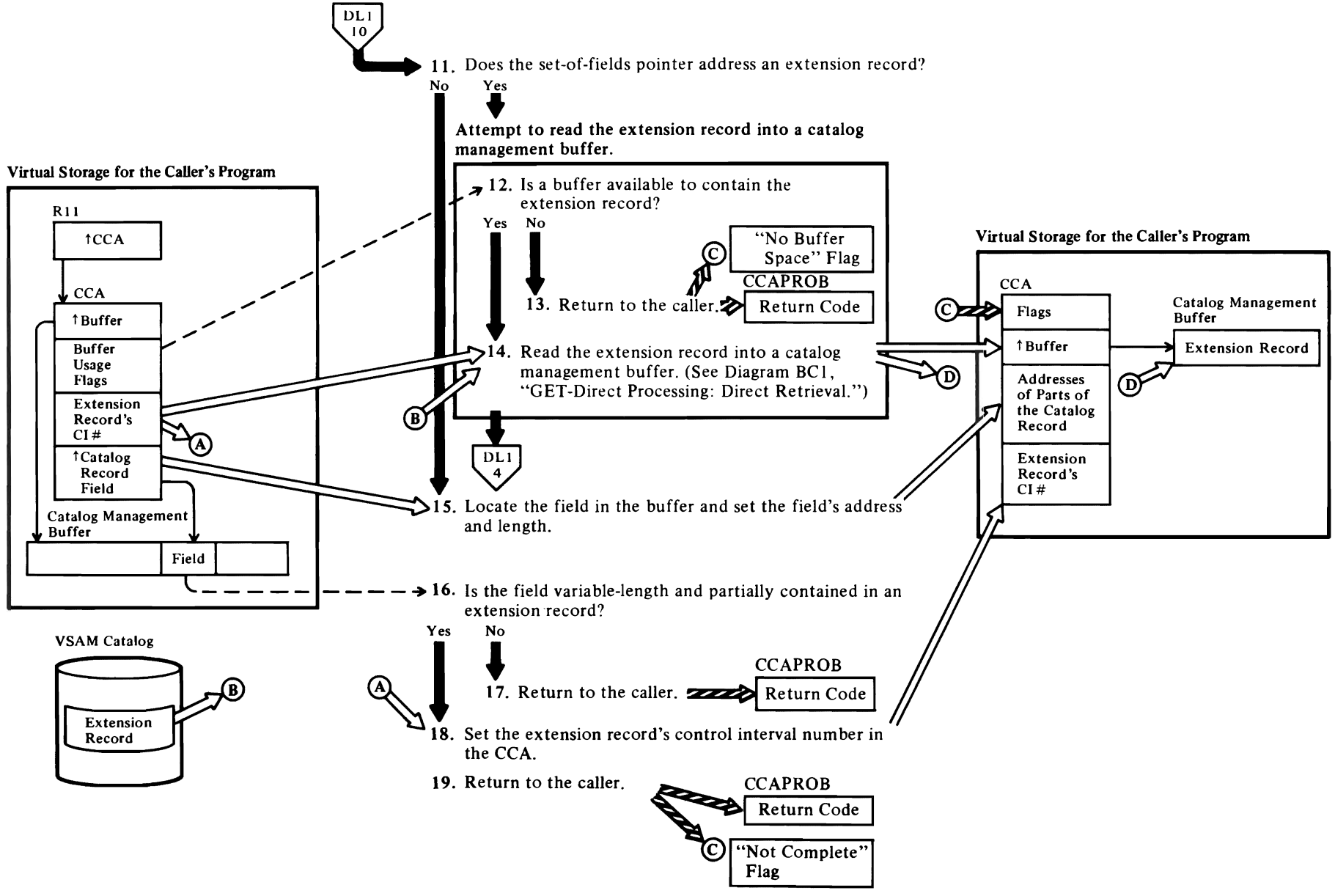
**9 IGG0CLBA: IGGPGVAL**

See "Diagnostic Aids" for details about catalog management return codes and error codes.

**10 IGG0CLBA: IGGPGREC**

The CCA's "not found" flag is set off before returning to step 3 to examine the extension record's set of fields pointers.

# Diagram DL2. Obtain a Catalog Record Field's Value



## Notes for Diagram DL2

**11 IGG0CLBA: IGGPLVAL**

If the set of fields pointer contains the control interval number of an extension record, the set of fields is in that extension record.

**12 IGG0CLBA: IGGPGVAL**

**IGG0CLBA: IGGPGREC**

Each catalog record (in a catalog management buffer) is identified by a record area block (RAB) within the CCA. The RAB contains flags that indicate whether or not the buffer can be used to contain another record. If the RAB's "must write" flag is on, the buffer cannot be used for another record until its contents have been written into the catalog.

See "Data Areas" for details about the CCA.

**IGG0CLBA: IGGPGVAL**

Each catalog control area (CCA) contains six record area blocks. Each catalog management request can use a maximum of five buffers. If all buffers are filled and cannot be released, the get-field-value routine sets the CCA's "no buffer space" flag on.

**13 IGG0CLBA: IGGPGVAL**

See "Diagnostic Aids" for details about catalog management return codes and error codes.

**14 IGG0CLBA: IGGPGREC**

The CCA's "not found" flag is set off before returning to step 3 to examine the extension record's set of fields pointers.

**15 IGG0CLBA: IGGPLVAL**

The field's length is obtained from the CTGFL's dictionary entry (for a fixed-length field) or the first 2 bytes of the field (length bytes of a variable-length field). The field's address is the sum of the address of the set of fields and the displacement in the CTGFL (for a fixed-length field); or is first, second, etc. (as indicated in the CTGFL) for a variable-length field.

**16 IGG0CLBA: IGGPLVAL**

A variable-length field might be partially contained in an extension record. If so, the field's length is greater than the number of bytes remaining in the record.

**17 IGG0CLBA: IGGPLVAL**

See "Diagnostic Aids" for details about catalog management return codes and error codes.

**18 IGG0CLBA: IGGPLVAL**

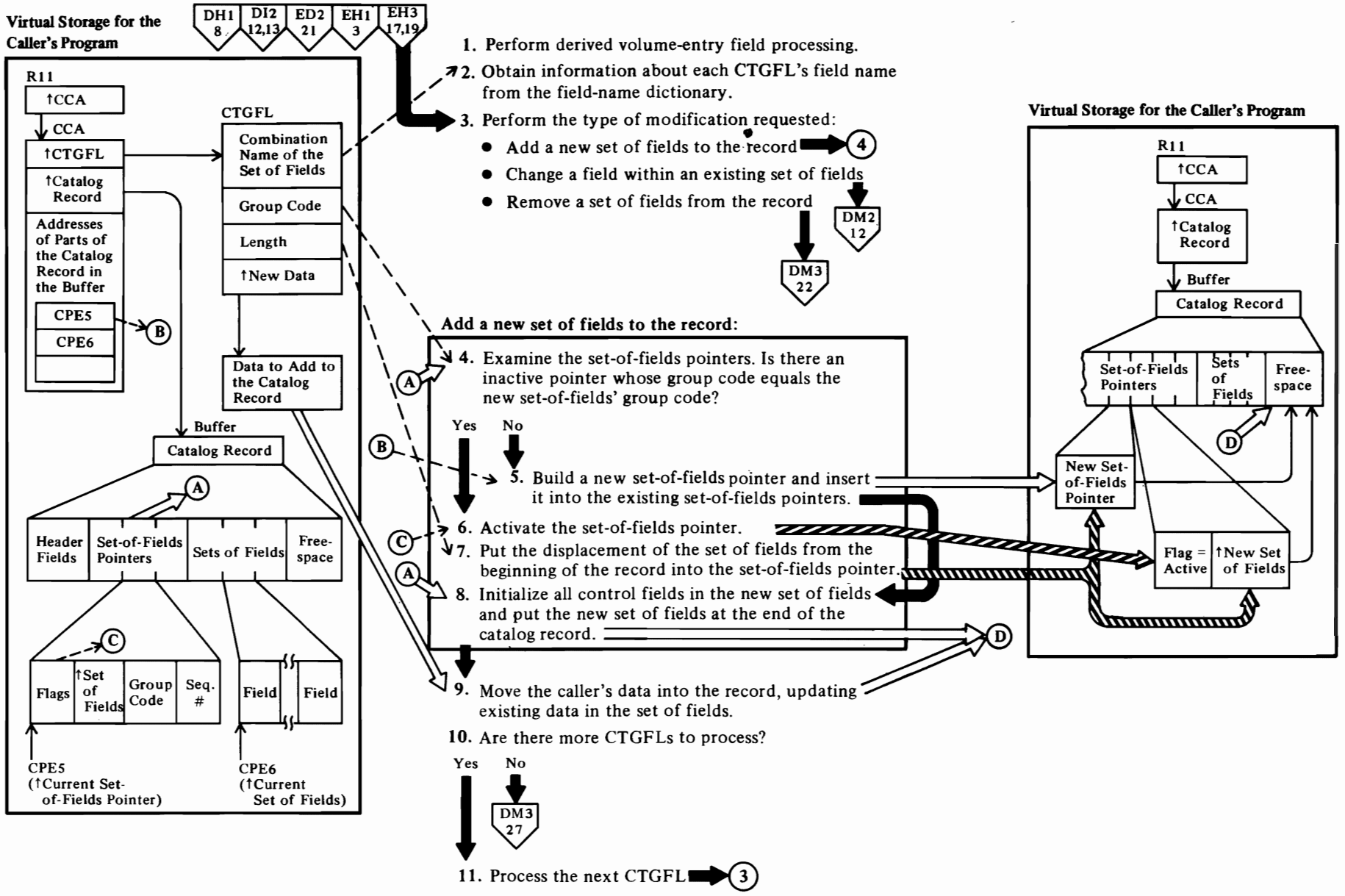
The caller's information requirements might be satisfied with the part of the field that is currently available. If not, the caller (a catalog management routine) returns to the get-field-value routine to obtain the next part of the field from the extension record.

**19 IGG0CLBA: IGGPLVAL**

The caller can move that part of the field currently in the buffer into a work area. If the rest of the field is required, the caller can return to the get-field-value routine to retrieve the extension record.

See "Diagnostic Aids" for details about catalog management return codes and error codes.

# Diagram DM1. Modify a Catalog Record Field's Value



## Notes for Diagram DM1

**1 IGG0CLAV: IGGPMOD** (calls **IGGPXMOD (IGG0CLBT)**)

For newly-created volume entries, deleted set-of-fields pointers are inserted for the bit map set of fields to ensure that the base volume entry (V) contains the bit map set-of-fields pointers. This bit map set of fields is dynamically added when the first data space set of fields is added.

**2 IGG0CLAV: IGGPMOD** (calls **IGGPSCNC (IGG0CLAY)**)

Each CTGFL is initialized with the dictionary entry associated with the CTGFL's field-name value.

**3 IGG0CLAV: IGGPMOD** (calls **IGGPSFPL (IGG0CLAV)**)

The field parameter list (CTGFL) contains the field's name, type code, length, and displacement from the beginning of the record or set of fields in the case of a fixed-length field. For a variable-length field, contains the field's name, type code, and sequence number. If the field exists, it is either a header field (group code = 0) or a field within a set of fields. If the caller supplied modifying data and test conditions, the field is being altered. If the caller supplied modifying data and no test conditons, a set of fields is to be added to the record. If the caller identified a set of fields combination field-name but didn't supply modifying data, the set of fields is being deleted.

**4 IGG0CLAV: IGGPSFPL** (calls **IGGPXDGO (IGG0CLBT)** which in turn calls **IGGPADGO (IGG0CLAW)**)

Every new set of fields is examined by the derived-field processing routine (IGGPXDGO) before being passed on to the normal add field processing routine (IGGPADGO). The derived-field processing routine ensures that certain volume set of fields are never added, added in a different format, or cause dynamic addition of a different set of fields (i.e., bit map set of fields).

**5 IGG0CLAV: IGGPAGOP**

If a new set of fields pointer is built, it is put into the catalog record at the end of its group of set-of-fields pointers. The set-of-fields pointers are grouped by type code with the codes in sequence number order.

If the new set of fields pointer causes the catalog record to overflow, an extension record is obtained from the catalog's free control intervals. All sets of fields in the original record are put into the extension record. The set-of-fields pointer's displacement value (in the original record) is replaced with the control interval number of the extension record. In addition, a set-of-fields pointer is built and put into the extension record for each set of fields in the extension record. The set-of-fields pointer in the extension record contains the displacement from the beginning of the record to its set of fields.

If the new set of fields pointer causes the catalog record to overflow and the catalog record contains only set-of-fields pointers, an extension record is obtained to contain the new set-of-fields pointer. The original record's extension field contains the control interval number of the extension record.

**6 IGG0CLAW: IGGPAGOP** (calls **IGGPIGOP (IGG0CLAW)**)

The Modify roùtine activates the set of fields pointer by setting it's "inactive" flag off.

See "Data Areas" for details about the catalog record and set of fields pointer.

**7 IGG0CLAW: IGGPADGO**

See "Data Areas" for details about the set of fields pointer.

**8 IGG0CLAW: IGGPADGO** (calls **IGGPMVGD (IGG0CLAW)**)

The new set of fields might contains fixed-length fields and variable-length fields.

If the new set of fields causes the record to overflow, an extension record is obtained to contain the new set of fields.

See "Data Areas" for details about the catalog record and its sets of fields.

**9 IGG0CLAW: IGGPADGO** (calls **IGGPMVGO (IGG0CLAW)**)

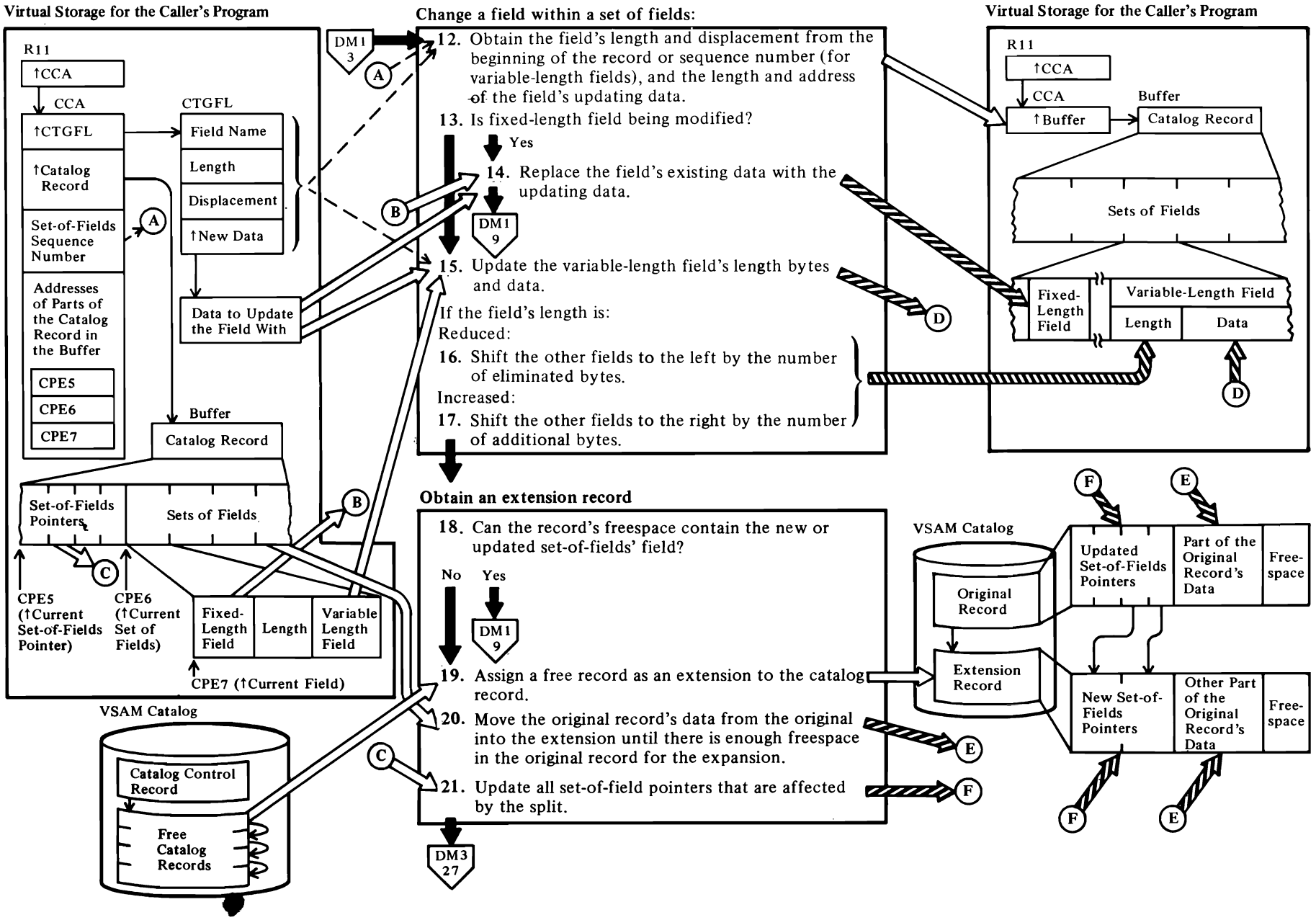
Replace the initial field values (from step 6) with the caller-supplied values addressed by the CCA.

**10 IGG0CLAV: IGGPSFPL**

If there are no more CTGFLs to process, calls IGGPPREC to write each updated catalog record into the catalog.

**11 IGG0CLAV: IGGPSFPL**

# Diagram DM2. Modify a Catalog Record Field's Value



## Notes for Diagram DM2

**12 IGG0CLAV: IGGPSFPL** (calls **IGGPXLT2 (IGG0CLBT)** which in turn calls **IGGPALT2 (IGG0CLAX))**

Every field to be updated is examined by the derived-field processing routine (IGGPXLT2) before being passed on to the normal update field processing routine (IGGPALT2). The derived-field processing routine ensures that certain volume entry set of fields are never altered, primarily because the altered fields do not physically exist in the catalog records.

**IGG0CLAX: IGGPALT2** (calls **IGGPGVAL (IGGCLBA))**

The CCA's CCACPE7 field contains the field's address. The CTGFL contains the address and length of the data to update the field with.

**13 IGG0CLAX: IGGPALT2**

The CTGFL flags field (from the catalog field name directory) specifies field type.

See "Data Areas" for details about the CTGFL.

**14 IGG0CLAX: IGGPALT2**

The CTGFL contains the length and address of the updating data. The data is in the caller's work area.

**15 IGG0CLAX: IGGPMVAR**

The CTGFL flags field (from the catalog field name directory) specifies field type. If the length of the data to update the field with (in the CTGFL) isn't equal to the field's length (in the CCA), the variable-length field's length is either increased or decreased, causing a corresponding reduction or increase in the catalog record's amount of free space.

The variable-length field's length bytes are replaced with the length of the data to update the field with (in the CTGFL).

**16 IGG0CLAX: IGGPSHNK**

The eliminated bytes at the end of the record are added to the record's free space.

**17 IGG0CLAX: IGGPEXPD**

The additional bytes are obtained by reducing the record's free space.

If the increased length causes the catalog record to overflow, an extension record is obtained. The original record's data is split so that part remains in the original record and part is moved into the extension record. Each associated set of fields pointer is updated to show the new position of its set of fields.

**19 IGG0CLAX: IGGPALT2** (calls **IGGPAOCI (IGG0CLAG))**

The catalog control record (CCR) contains the control-interval number of a free control interval. Catalog management allocates the free control interval to the original catalog record as an extension record. The control-interval number of the next free control interval is put into the CCR, and the CCR's free control interval count is decreased by 1.

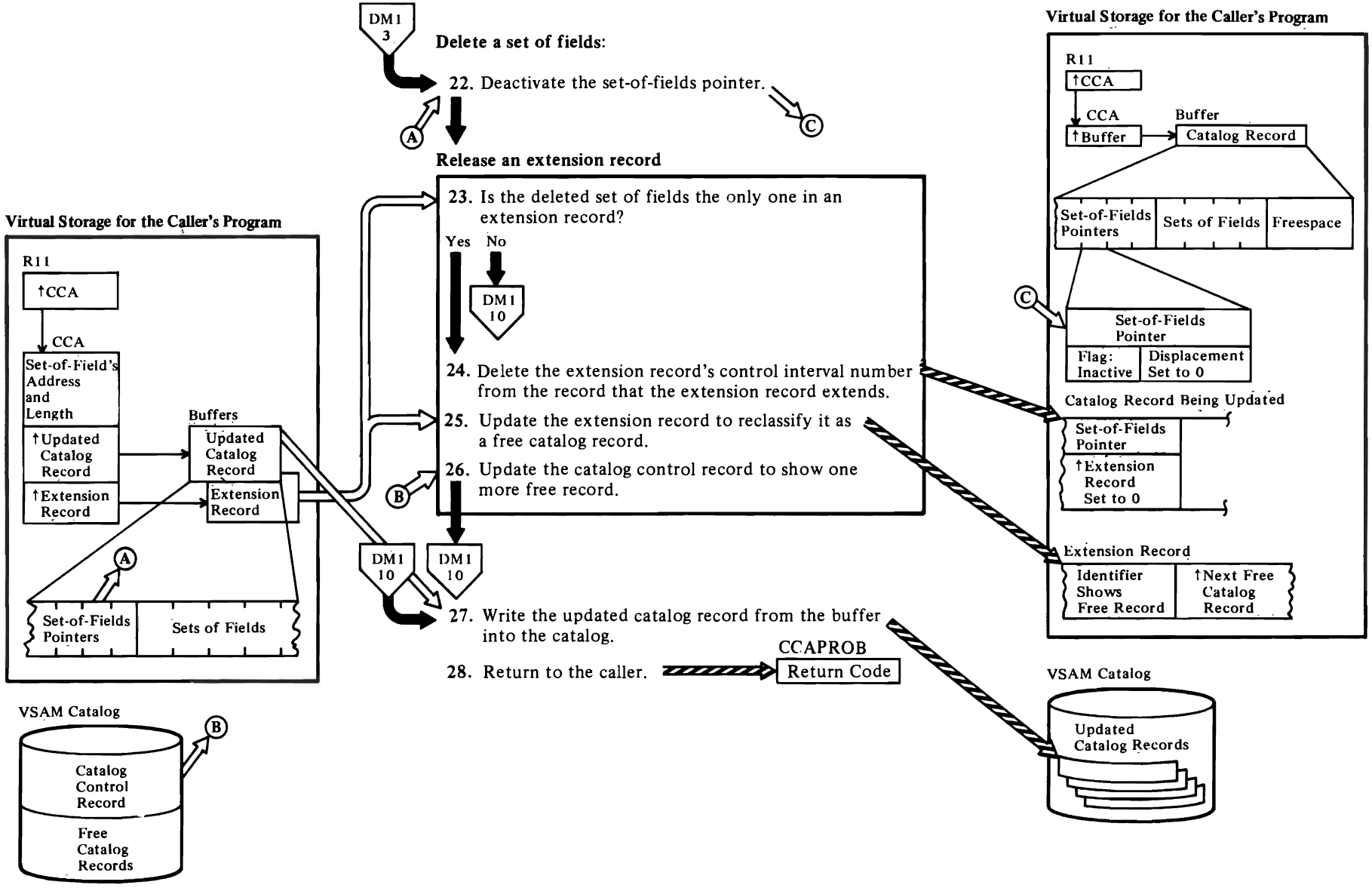
**20 IGG0CLAX: IGGPALT2**

The set of fields' data is split so that part remains in the original record and part is moved into the extension record.

**21 IGG0CLAX: IGGPALT2**

Each set of fields pointer is updated to show the new position of its set of fields.

# Diagram DM3. Modify a Catalog Record Field's Value



## Notes for Diagram DM3

**21 IGG0CLAV: IGGPSFPL (calls IGGPXEL2 (IGG0CLBT)) which in turn calls IGGPDEL2 (IGG0CLAV))**

Every set of fields to be deleted is examined by the derived field processing routine (IGGPDEL2) before being passed on to the normal delete set of fields processing routine (IGGPDEL2). The derived field processing routine ensures that certain nonexistent sets of fields are not deleted and that the bit map set of fields is updated.

**IGG0CLAV: IGGPDEL2**

The set of fields pointer's "inactive" flag is; set on, thereby deactivating it. The set of fields pointer's type code and sequence number fields are unchanged. The set of fields is removed from the record and the displacement field is set to 0, if the field is in the same record as the set-of-fields pointer; otherwise, the following is done:

**23 IGG0CLAV: IGGPDEL2**

If the extension record contains no data after the set of fields is removed, the record is reclaimed by catalog management as a free control interval.

**25 IGG0CLAV: IGGPDEL2**

See "Data Areas" for details about the catalog record.

**26 IGG0CLAV: IGGPDEL2**

The catalog control record (CCR) contains the count of free control intervals available to the catalog and the control interval number of a free control interval. All free control intervals are chained together. When a control interval is added to the free control interval chain, catalog management puts the CCR's free control interval number into the new free control interval and puts the new free control interval's control-interval number into the CCR. The CCR's free-control-interval count is increased by 1.

See "Data Areas" for details about the catalog record and the extension record.

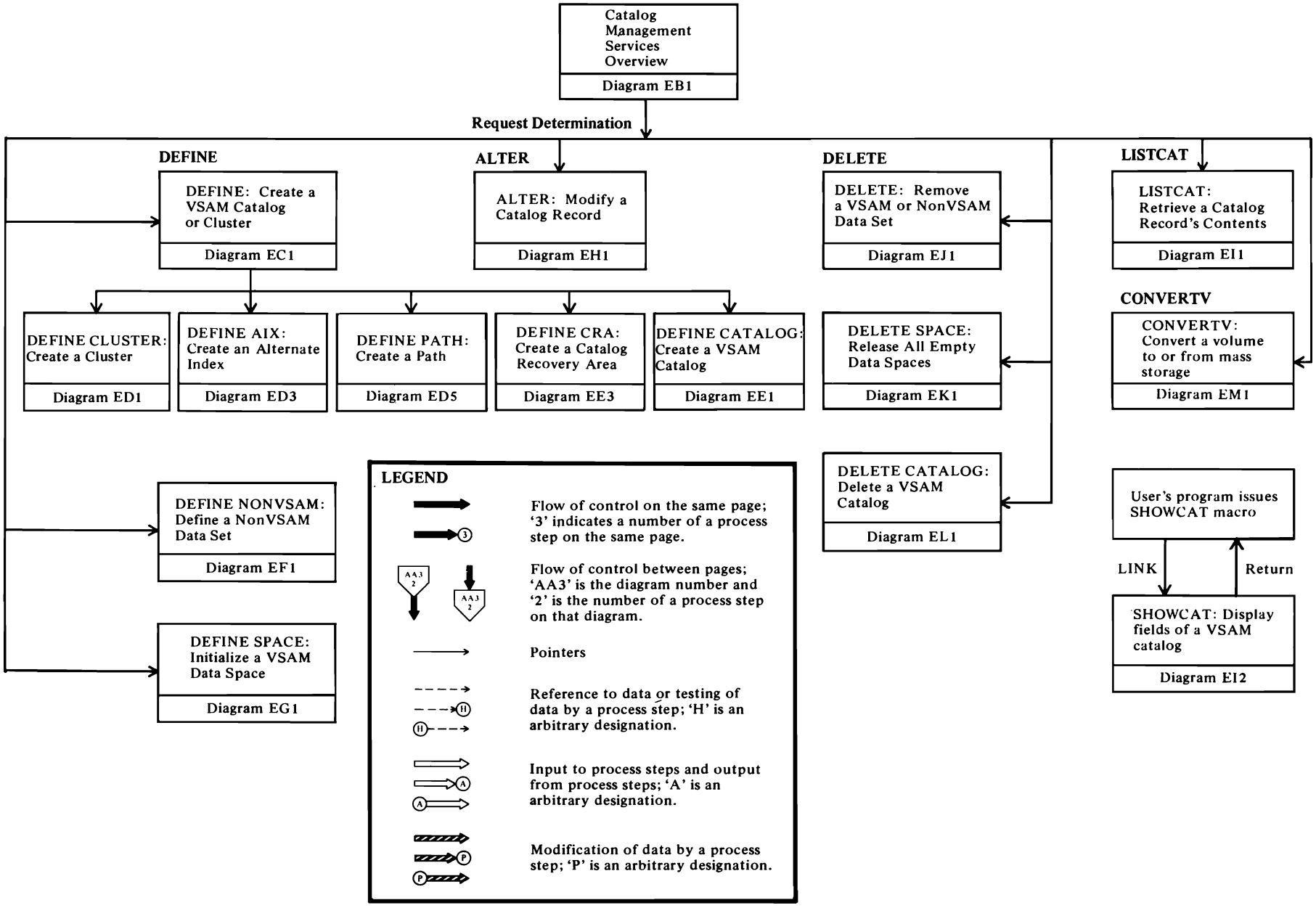
**27 IGG0CLAV: IGGPMOD**

**28**

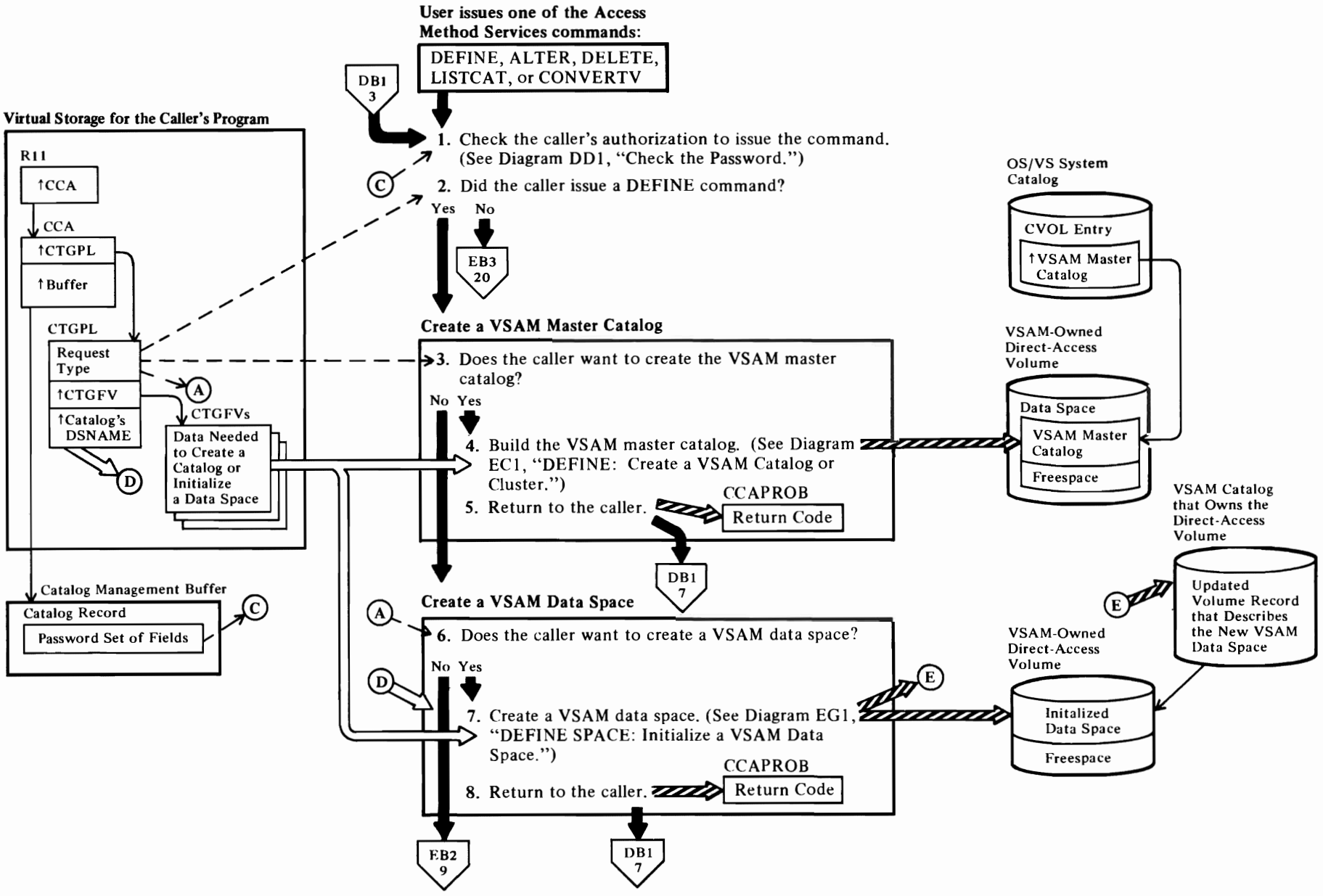
The modify routine returns to the caller (a catalog management routine) when the catalog record field is updated.

See "Diagnostic Aids" for details about catalog management return codes and error codes.

# Diagram EA1. Catalog Management Services Table of Contents



# Diagram EB1. Catalog Management Services Overview



## Notes for Diagram EB1

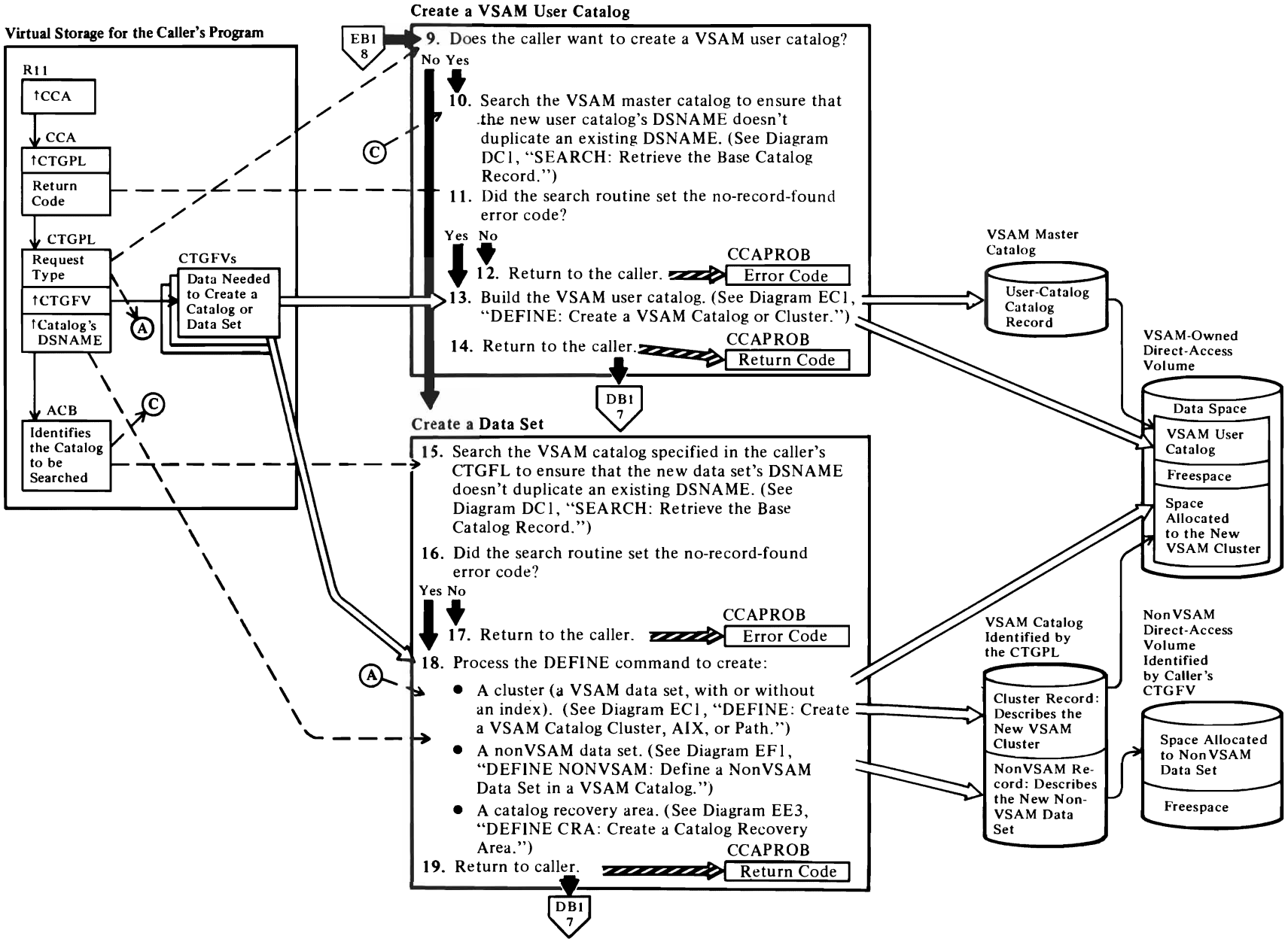
Catalog Management Services is a group of VSAM catalog management modules that respond to Access Method Services commands. The Catalog Management Services Driver routine, IGGPCDVR (IGG0CLAT), calls other Catalog Management Services routines as described in the table below.

See "Data Areas" for details about the ACB.

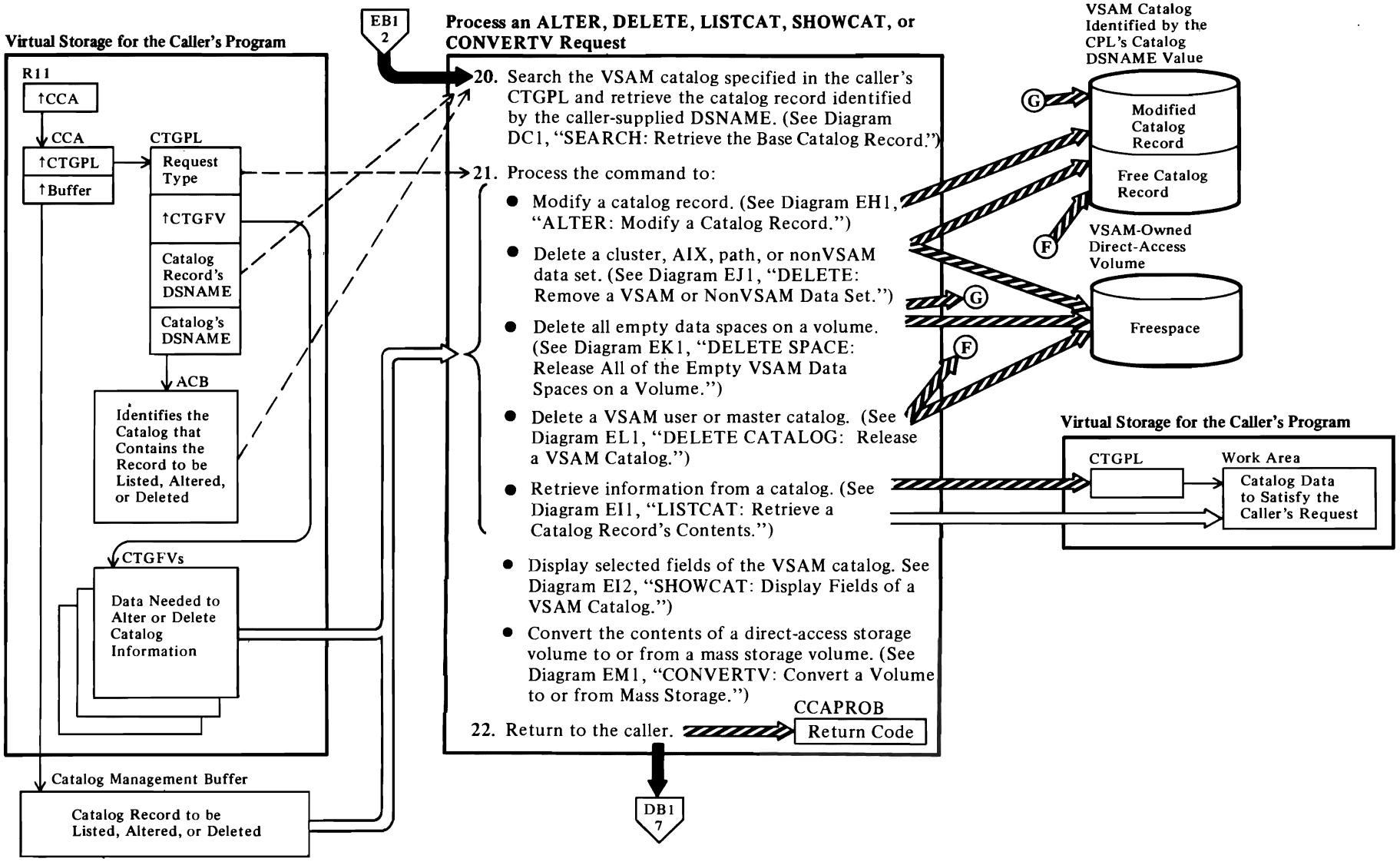
See "Diagnostic Aids" for details about catalog management return codes and error codes.

| | IGG0CLAT calls: | Related Method of Operation Diagram: |
|---|---|---|
| DEFINE CLUSTER, AIX, PATH, DEFINE MASTERCATALOG, and DEFINE USERCATALOG | IGG0CLAL | Diagram EC1 |
| DEFINE CRA | IGG0CLB4 | Diagram EE3 |
| DEFINE NONVSAM | IGG0CLBH | Diagram EF1 |
| DEFINE SPACE | IGG0CLAQ | Diagram EG1 |
| ALTER | IGG0CLBD | Diagram EH1 |
| LISTCAT | IGG0CLBQ | Diagram EI1 |
| DELETE TYPE (CLUSTER, AIX, PATH) and DELETE TYPE (NONVSAM) | IGG0CLBG | Diagram EJ1 |
| DELETE TYPE (SPACE) | IGG0CLBL | Diagram EK1 |
| DELETE TYPE (MASTERCATALOG) and DELETE TYPE (USERCATALOG) | IGG0CLAF | Diagram EL1 |
| CONVERTV | IGG0CLBZ | Diagram EM1 |

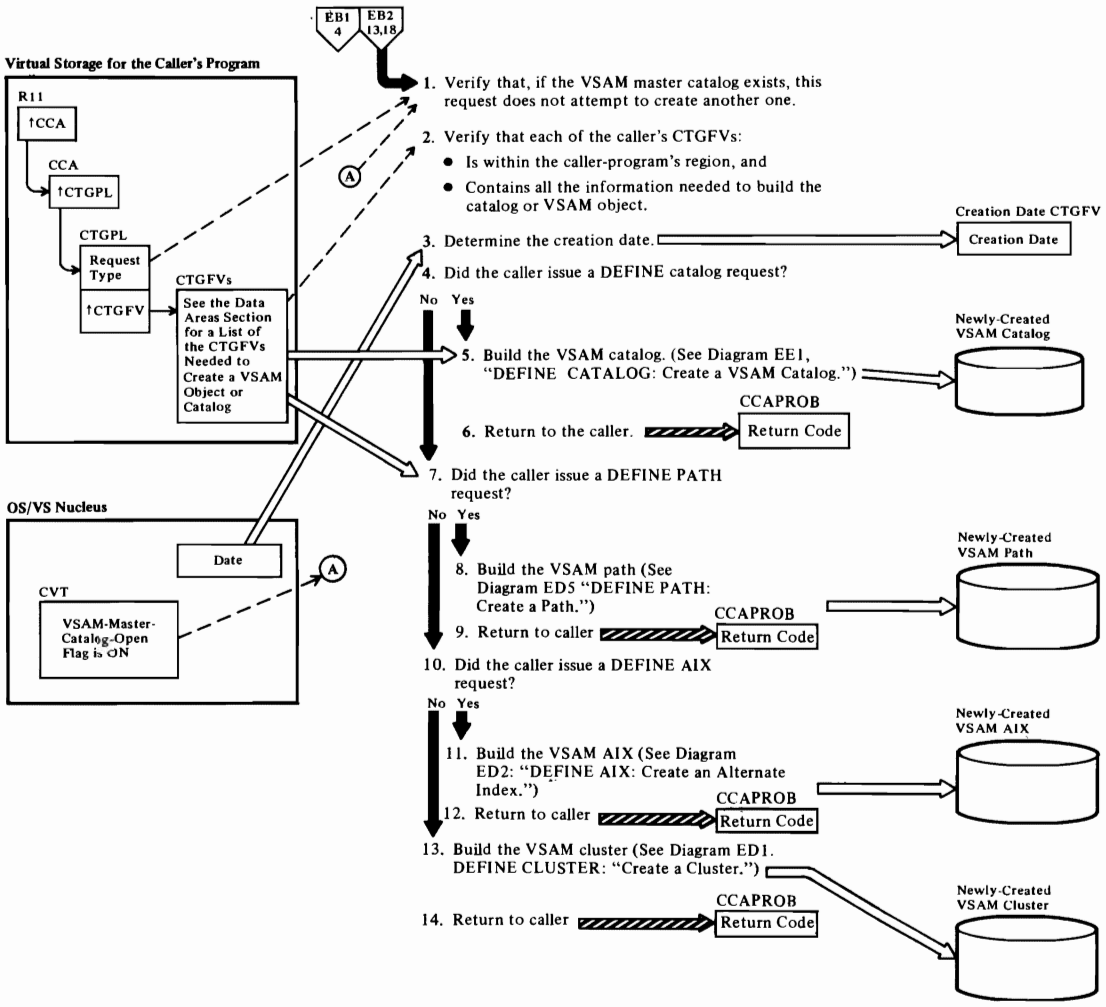
# Diagram EB2. Catalog Management Services Overview



# Diagram EB3. Catalog Management Services Overview



## Diagram EC1. DEFINE: Create a VSAM Catalog, Cluster, AIX or Path



## Notes for Diagram EC1

When the user issues the Access Method Services DEFINE command to create a catalog (either user or master) or a VSAM object, the catalog management services DEFINE: Initial Processing modules ensure that the caller provided all the information necessary to create a catalog or VSAM object.

**1 IGG0CLAL: IGGPDEF**

**2 IGG0CLAL: IGGPDEDE**

CTGFVs are checked by routines (internal procedures) in the IGG0CLAL and IGG0CLAN modules, as shown in the table below.

**3 IGG0CLAL: IGGPDEDE**

**5 IGG0CLAN: IGGPDSCB** (calls **IGGPDCDA (IGG0CLAP)** and **IGGPDEFC (IGG0CLAS)**)

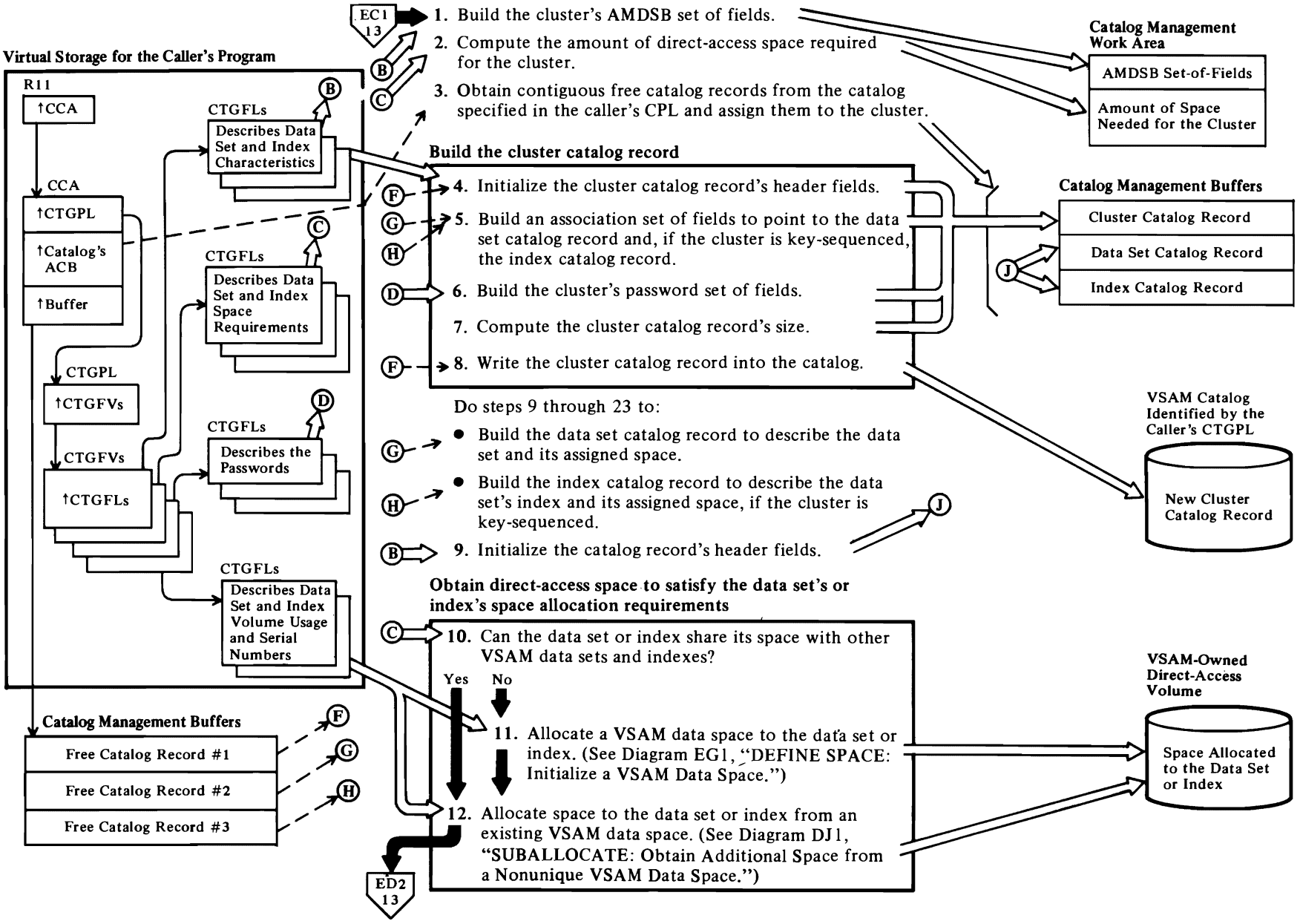
**8 IGG0CLAL: IGGPDDEP**

**10 IGG0CLAL: IGGPDCWC**

**13 IGG0CLAN: IGGPDSCB** (calls **IGGPDRDA (IGG0CLAN)** and **IGGPDBDI (IGG0CLAJ)**)

| CTGFV | Module Procedure |
|---|---|
| Cluster CTGFV | IGG0CLAL: IGGPDCWC |
| AIX CTGFV | IGG0CLAL: IGGPDCWC |
| Caller's work area | IGG0CLAL: IGGPDCWC |
| Data CTGFV structure | IGGOCLAL: IGGPDFSC |
| Index CTGFV structure | IGG0CLAL: IGGPDFSC |
| Data dsname CTGFV | IGG0CLAL: IGGPDDNP |
| Index dsname CTGFV | IGG0CLAL: IGGPDDNP |
| Path CTGFV | IGG0CLAL: IGGPDDEP |
| Data space CTGFV | IGG0CLAL: IGGPDBVC |
| Catalog's space CTGFV | IGG0CLAL: IGGPDCSF |
| Key range CTGFV | IGG0CLAL: IGGPDRPG |
| Space parameter CTGFLs | IGG0CLAN: IGGPDSPF |
| Buffer size CTGFLs | IGG0CLAN: IGGPDBSF |
| Average record size CTGFLs | IGG0CLAN: IGGPDALR |

# Diagram ED1. DEFINE CLUSTER: Create a Cluster



## Notes for Diagram ED1

This figure describes the processes performed by catalog management services routines when the user issues the Access Method Services DEFINE CLUSTER command in the form:

```
DEFINE
CLUSTER
| CATALOG (catname/password) |
(parameterlist)
```

where:

- *catname* is the name of the catalog that will contain the cluster, data, and index catalog records that describe the user's data set.
- *password* is the catalog's master, control interval, or update password, if the catalog is protected by passwords.
- *parameterlist* is a list of optional and required parameters that define the cluster's characteristics.

See *OS/VS2 SVS Independent Component: Access Method Services* for details about the DEFINE command parameters.

**1 IGG0CLAN: IGGPDRDA**

The AMDSB contains the cluster's statistics and fixed characteristics. Each time the cluster is opened, the AMDSB is retrieved from the data catalog record. When the cluster is closed, the AMDSB is updated and rewritten into the data catalog record.

See "Data Areas" for details about the AMDSB.

**2 IGG0CLAN: IGGPDSPC**

The field vector table contains addresses of buffer-size and record-length field parameter lists (CTGFLs). This data is used to determine the data set's control-interval and control-area size. If the data set is key-sequenced, other buffer-size and record-length CTGFLs determine the index's control-interval and control-area size. If the key-sequenced data set is divided into key ranges, the size of each key range is determined.

**3 IGG0CLAN: IGGPDCCE**

A user's data set is described by a cluster catalog record, a data catalog record, and, if the data set is key-sequenced, an index catalog record.

See "Data Areas" for details about the catalog record.

**8 IGG0CLAN: IGGPDCCE**

The DEFINE routine issues an ADDREC macro instruction to write the cluster record into the catalog.

**9 IGG0CLAJ: IGGPDBDI**

See "Data Areas" for details about the data catalog record.

**10 IGG0CLAJ: IGGPDSPO**

**11 IGG0CLAJ: IGGPDSPO** (calls **IGGPDEFS (IGG0CLAQ)**)

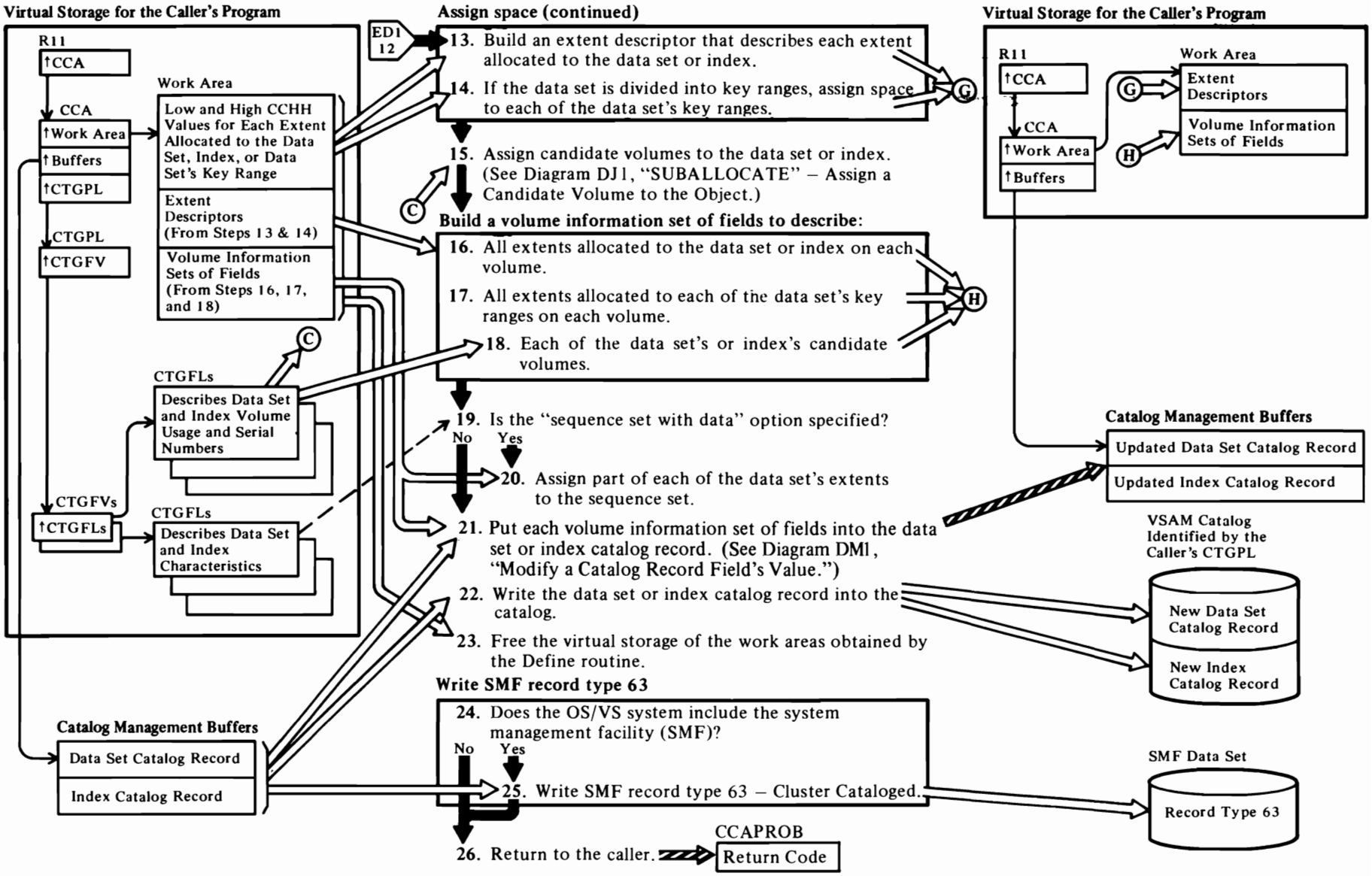
A data space group set of fields is added to the volume catalog record, and the data set's name is added to the volume catalog record's data set directory.

See "Data Areas" for details about the volume catalog record, the data space group set of fields, and its data set directory entry set of fields.

**12 IGG0CLAJ: IGGPDSPO** (calls **IGGPSALL (IGG0CLAR)**)

The data set's name is added to the volume catalog record's data set directory entry set of fields, and the volume catalog record's data space group set of fields is updated to show the cylinders and tracks allocated to the new data set or index.

# Diagram ED2. DEFINE CLUSTER: Create a Cluster



## Notes for Diagram ED2

**13 IGG0CLAJ: IGGPDSEX**

**15 IGG0CLAJ: IGGPDCNV**

A candidate volume is available to contain part of the cluster if more space is needed later. None of the candidate volume's space is allocated to the data set or index when the cluster is created.

**16 IGG0CLAK: IGGPDBVO**

Each volume that contains a part of the data set or index is described by a volume information set of fields within the data set or index catalog record.

See "Data Areas" for details about the data catalog record, the index catalog record, and the volume information set of fields.

**17 IGG0CLAK: IGGPDRNG**

If the data set is divided into key ranges, each key range's space on a volume is described in a separate volume information set of fields. If the key range's space is on more than one volume, each volume that contains part of the key range's space is described is a separate volume information set of fields.

**18 IGG0CLAK: IGGPDBCV**

**20 IGG0CLAK: IGGPDSSP**

If the "sequence set with data" option is specified, part of the data set's space is allocated to the index for sequence set records. The low and high CCHH values in the index record's volume information set of fields are those of the extent obtained for the data set. The low and high RBA values are for the sequence set and are relative to the index addresses.

**21 IGG0CLAK: IGGPDMOP**

**22 IGG0CLAK: IGGPDMOP**

A catalog management routine writes the completed data set or index catalog record into the VSAM catalog and frees the catalog management buffer that contains the record.

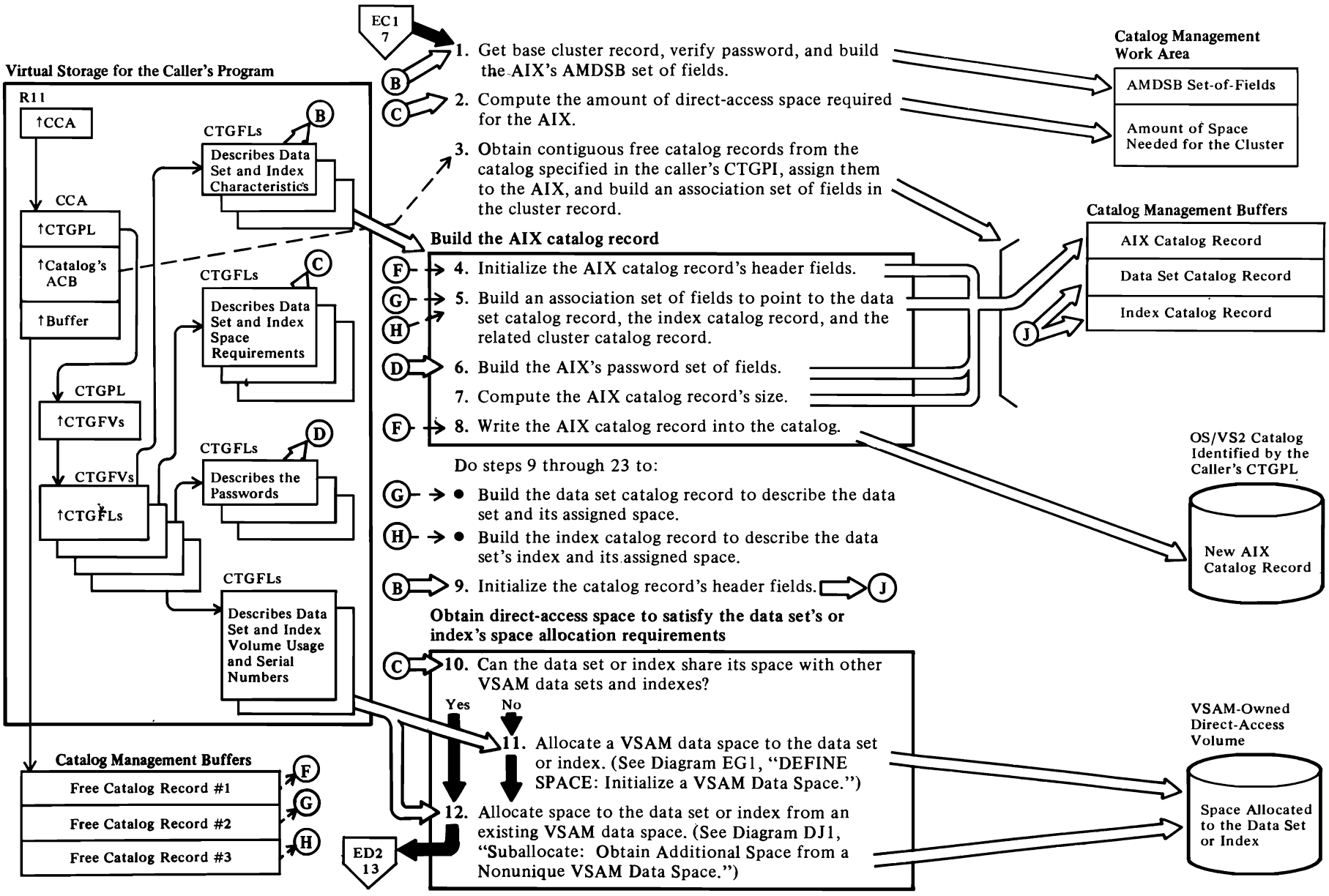
**23 IGG0CLA8: IGGPDFRS**

**25 IGG0CLAJ: IGGPDBDI** (calls **IGGPSMFA (IGG0CLBV)**)

See *OS/VS System Management Facilities (SMF)* for details of SMF record type 63—VSAM Data Set Cataloged. Record type 63 is written after a VSAM cluster or catalog is defined and whenever the definition is altered.

**26** See "Diagnostic Aids" for details about catalog management return codes and error codes.

# Diagram ED3. DEFINE AIX: Create an Alternate Index



## Notes for Diagram ED3

This figure describes the processes performed by catalog management services routines when the user issues the Access Method Services DEFINE AIX command in the form:

```
DEFINE
AIX
(parameterlist)
[CATALOG(catname/password)]
```

where:

- *catname* is the name of the catalog that will contain the AIX, data set, and index catalog records that describe the user's data set.
- *password* is the catalog's master, control interval, or update password, if the catalog is protected by passwords.
- *parameterlist* is a list of optional and required parameters that define the AIX's chacteristics.

See *OS/VS2 SVS Independent Component: Access Method Services* for details about the DEFINE command parameters.

**1 IGG0CLAN: IGGPDRDA**

The AMDSB contains the cluster's statistics and fixed characteristics. The AMDSB set of fields is in the Data catalog record (for an entry-sequenced data set) and in the Data and Index catalog records (for a key-sequenced data set). Each time the cluster is opened, the AMDSB is retrieved from the data set catalog record. When the cluster is closed, the AMDSB is updated and rewritten into the data set catalog record.

**2 IGG0CLAN: IGGPDSPC**

The field vector table (CTGFV) contains addresses of buffer-size and record-length field parameter lists (CTGFLs). This data is used to determine the data set's control-interval and control-area size. If the data set is key-sequenced, other buffer-size and record-length CTGFLs determine the index's control-interval and control-area size. If the key-sequenced data set is divided into key ranges, the size of each key range is determined.

**IGG0CLBX**

IGG0CLBX determines the amount of secondary storage needed. For Pagespaces, IGG0CLBX calculates this space using track overflow where appropriate.

**IGG0CLBY: IGGPDRSP**

IGGPDRSP is called to calculate the space parameters.

**3 IGG0CLAG: IGGPAOCI**
**IGG0CLB9: IGGPMODC**

IGGPAOCI is called to obtain three contiguous catalog control intervals to contain the AIX, data set, and index catalog records.

**8 IGG0CLAN: IGGPDCCE**

The DEFINE routine issues an ADDREC macro instruction to write the AIX record into the catalog.

**9 IGG0CLAJ: IGGPDBDI**

**10 IGG0CLAJ: IGGPDSPO**

**11 IGG0CLAJ: IGGPDSPO** (calls **IGGPDEFS (IGG0CLAQ)**)

A Data Space Group set of fields is added to the volume catalog record, and the data set's name is added to the volume catalog record's data set directory.

**12 IGG0CLAJ: IGGPDSPO** (calls **IGGPSALL (IGG0CLAR)**)

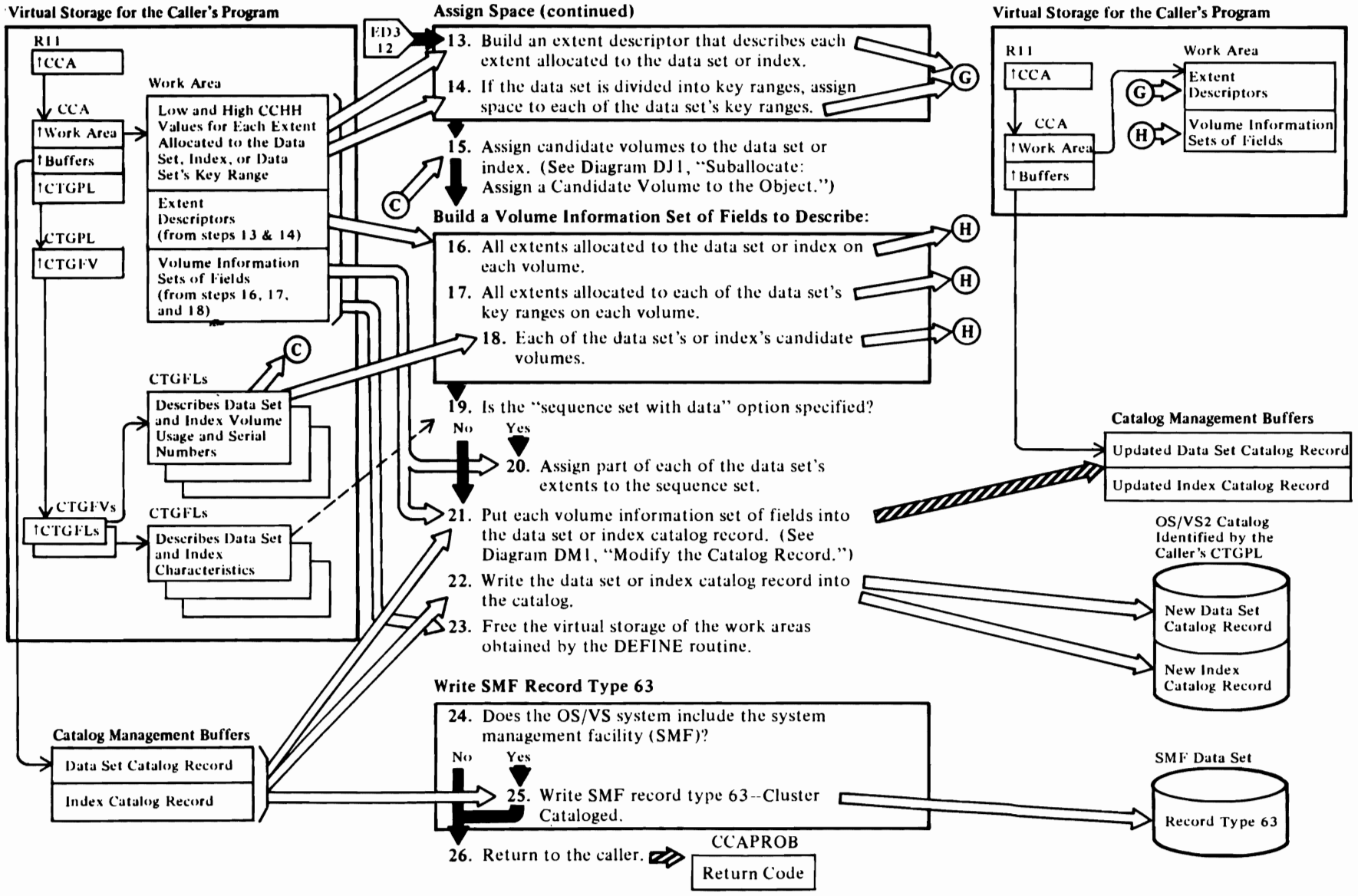
The data set's name is added to the volume catalog record's Data Set Directory Entry set of fields, and the volume catalog record's Data Space Group set of fields is updated to show the cylinders and tracks allocated to the new data set or index.

For additional information about topics related to DEFINE AIX processing, see:

"Data Areas."

Catalog record description and format
Data set catalog record description and format
Volume catalog record description and format
Data set group set of fields description
Data set directory entry set of fields descrption
Access Method data set statistics block (AMDSB) description and format

# Diagram ED4. DEFINE AIX: Create an Alternate Index



## Notes for Diagram ED4

**13 IGG0CLAJ: IGGPDSEX**

**14 IGG0CLAJ: IGGPDSPO**

Each key range is assigend physical space and space allocation continues until all key ranges have been assigned space.

**15 IGG0CLB0: IGGPDCNV**

A candidate volume is available to contain part of the cluster if more space is needed later. None of the candidate volume's space is allocated to the data set or index when the AIX is created.

**16 IGG0CLAK: IGGPDBVO**

Each volume that contains a part of the data set or index is described by a volume information set of fields within the data set or index catalog record.

**17 IGG0CLAK: IGGPDRNG**

If the data set is divided into key ranges, each key range's space on a volume is described in a separate volume information set of fields. If the key range's space is on more than one volume, each volume that contains part of the key range's space is described in a separate volume information set of fields.

**18 IGG0CLAK: IGGPDBCV**

IGGPDBCV builds a volume information set of fields for each candidate volume of the data set.

**20 IGG0CLAK: IGGPDSSP**

If the "sequence set with data" option is specified, part of the data set's space is allocated to the index for sequence set records. The low and high CCHH values in the index record's volume information set of fields are those of the extent obtained for the data set. The low and high RBA values are for the sequence set and are relatie to the index addresses.

**21 IGG0CLAK: IGGPDMOP**

IGGPDMOP calls IGGPMOD to add each volume information set of fields to the record.

**22 IGG0CLAK: IGGPDMOP**

A catalog management routine writes the completed data set or index catalog record into the VSAM catalog and frees the catalog management buffer that contains the record.

**23 IGG0CLA8: IGGPDFRS**

IGGPDFRS frees all unneeded storage resources.

**25 IGG0CLAJ: IGGPDBDI** (calls **IGGPSMFA (IGG0CLBV)**)

See *OS/VS System Management Facilities (SMF)* for details of SMF record type 63—VSAM Data Set Cataloged. Record type 63 is written after a VSAM AIX is defined and whenever the definition is altered.

**26** For additional information about topics related to DEFINE AIX processing, see:

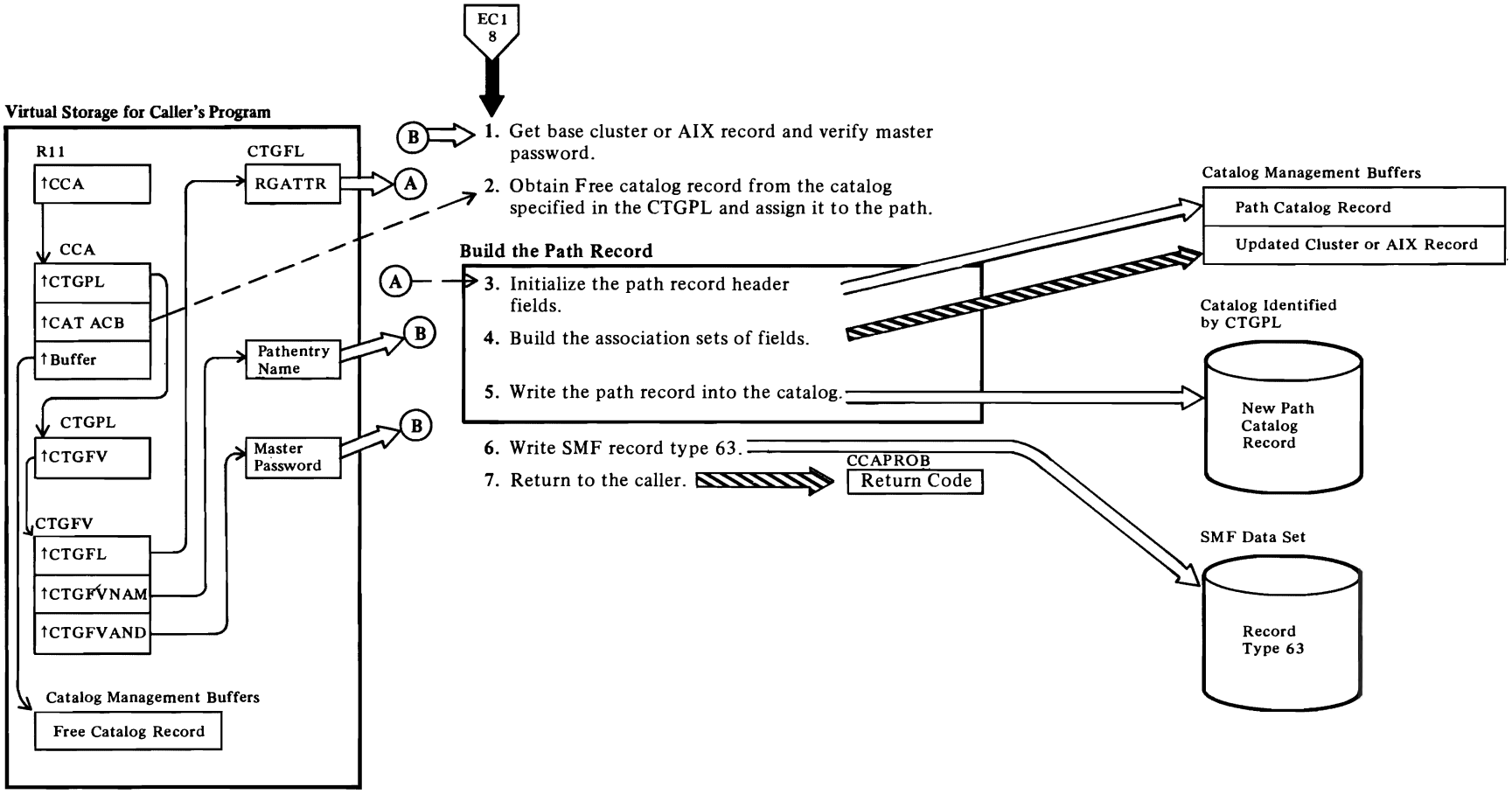
"Data Areas."

Data set catalog record description and format
Index catalog record description and format
Volume information set of fields description

"Diagnostic Aids."

Catalog management return codes

# Diagram ED5. DEFINE PATH: Create a Path



## Notes for Diagram ED5

This figure describes the processes performed by the catalog management services routines when the user issues the Access Method Services DEFINE PATH command in the form:

```
DEFINE PATH (NAME(name)
PATHENTRY(entryname/password)
(parameterlist))
```

where:

- *name* specifies the name to be given to the alternate index/base cluster pair (i.e., the path).
- *entryname* specifies the name of the alternate index or cluster which is to be considered as the entry to the path.
- *password* specifies the master-level password.
- *parameterlist* is a list of optional and required parameters that define the path's characteristics.

See *OS/VS2 SVS Independent Component: Access Method Services* for details about the DEFINE command.

**1 IGG0CLB9: IGGPPRPW**

**2 IGG0CLAG: IGGPAOCI**

IGGPAOCI is called to obtain one catalog control interval to contain the path record.

**3 IGG0CLB9: IGGPBAWP**

**4 IGG0CLB9: IGGPBAWP** and **IGGPBAMC**

Builds an association set of fields to point to the cluster and index records and, if the cluster is key-sequenced, the index record. If the path is being built over the AIX, then the pointers to the AIX, the AIX data, and the AIX index records are also included. An association set of fields is also built in the cluster or AIX record to point to the path record.

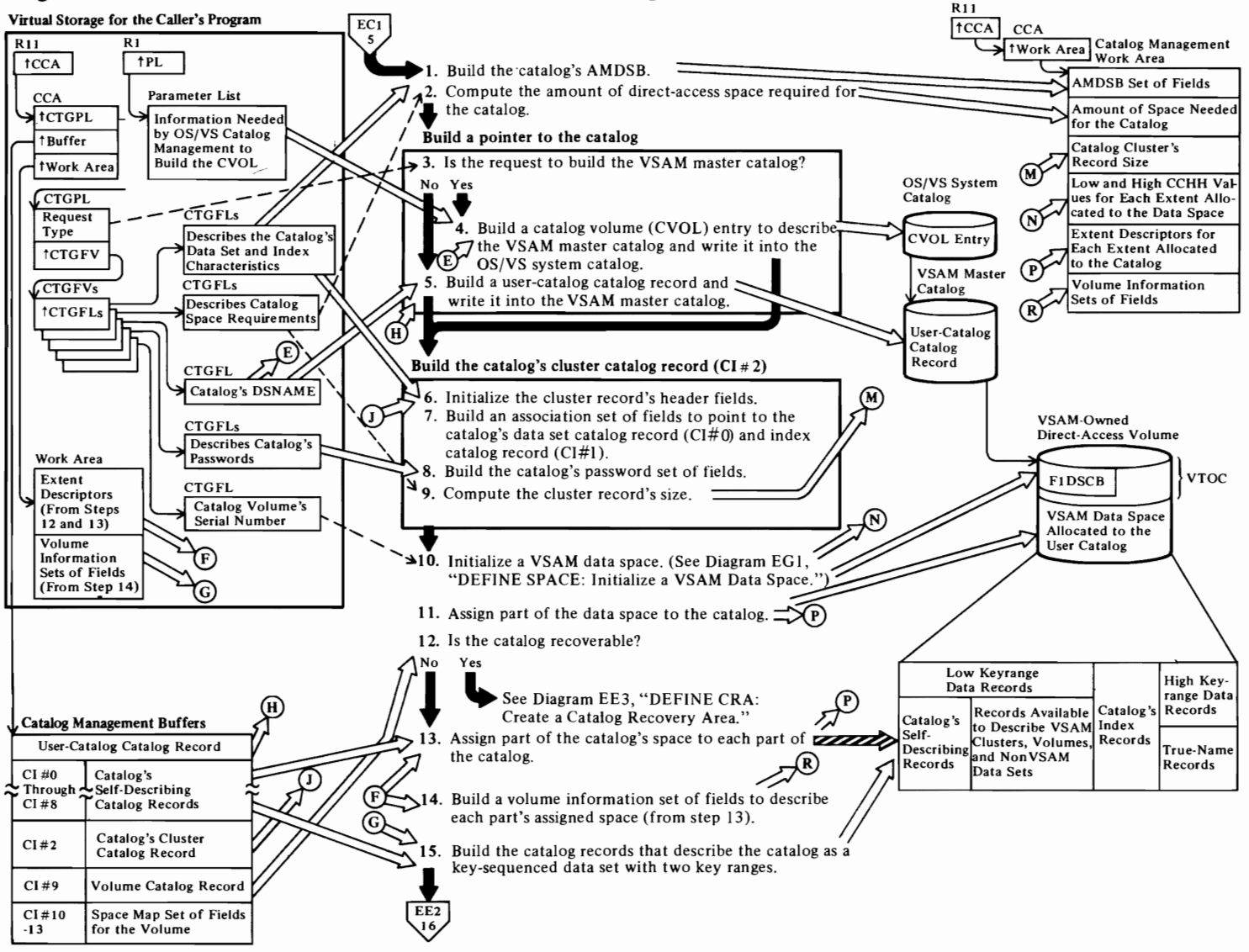
**5 IGG0CLB9: IGGPBAWP**

The DEFINE routine issues a modify with a put/add option to insert all the association sets of fields into the path record and to write the record into the catalog. These association sets of fields had been extracted previously from the cluster and the AIX records, if the path is being built over an AIX.

**6 IGG0CLBV: IGGPSMFA**

Record type 63 is written after a VSAM path is defined.

# Diagram EE1. DEFINE CATALOG: Create a VSAM Catalog



## Notes for Diagram EE1

This figure describes the processes performed by the catalog management services routines when the user issues the Access Method Services DEFINE MASTERCATALOG or DEFINE USERCATALOG command in the form:

DEFINE MASTERCATALOG

or

DEFINE USERCATALOG
[CATALOG (catname/password)]
(parameterlist)

where:

- *catname* is the name of the VSAM master catalog, which will contain the catalog record that describes the VSAM user catalog.
- *password* is the VSAM master catalog's master, control interval, or update password, if the master catalog is protected by passwords.
- *parameterlist* fields are described in *OS/VS2 SVS Independent Component: Access Method Services*.

**1 IGG0CLAP: IGGPDCDA**

The AMDSB contains the catalog's statistics and fixed characteristics. Each time the catalog is opened, the AMDSB is retrieved from the catalog's data set catalog record (control interval number 0). When the catalog is closed, the AMDSB is updated and rewritten into the data set catalog record.

See "Data Areas" for details about the AMDSB.

**2 IGG0CLAP: IGGPDCSP**

The field vector table contains addresses of buffer-size and record-length CTGFLs. This data is used to determine the catalog's control-interval and control-area size, and the amount of space required for the catalog.

**4 IGG0CLAP: IGGPDCPC**

The OS/VS system catalog contains a data set entry that describes the VSAM master catalog as a private catalog to the OS/VS system catalog.

**5 IGG0CLAP: IGGPDCPC**

See "Data Areas" for details about the user-catalog catalog record.

**7 IGG0CLAN: IGGPDCCE**

The cluster catalog record contains an associated-data-set set of fields to locate the catalog's data set catalog record (control interval number 0) and an associated-index set of fields to locate the catalog's index catalog record (control interval number 1).

See "Data Areas" for details about the cluster catalog record.

**10 IGG0CLAS: IGGPDCSP (calls IGGPDEFS (IGG0CLAQ))**

The VSAM catalog is always built in a data space that can contain other VSAM data sets and indexes. A new data space is allocated to VSAM by the OS/VS DADSM Allocate routine, and the data space is assigned to the new catalog.

See "Data Areas" for details about the volume catalog record, the data space group set of fields, and the data set directory entry set of fields.

**11 IGG0CLAS: IGGPDCSP (calls IGGPSALL (IGG0CLAR))**

A data set directory entry set of fields containing the cluster's control interval number is added to the volume catalog record.

**12 IGG0CLAS: IGGPDCSP (calls IGGPDCRA (IGG0CLB4))**

**13 IGG0CLAS: IGGPDCLD**

The catalog might contain records that describe user's VSAM data sets, user's nonVSAM data sets, direct-access volumes, and (in the VSAM master catalog) VSAM user catalogs.

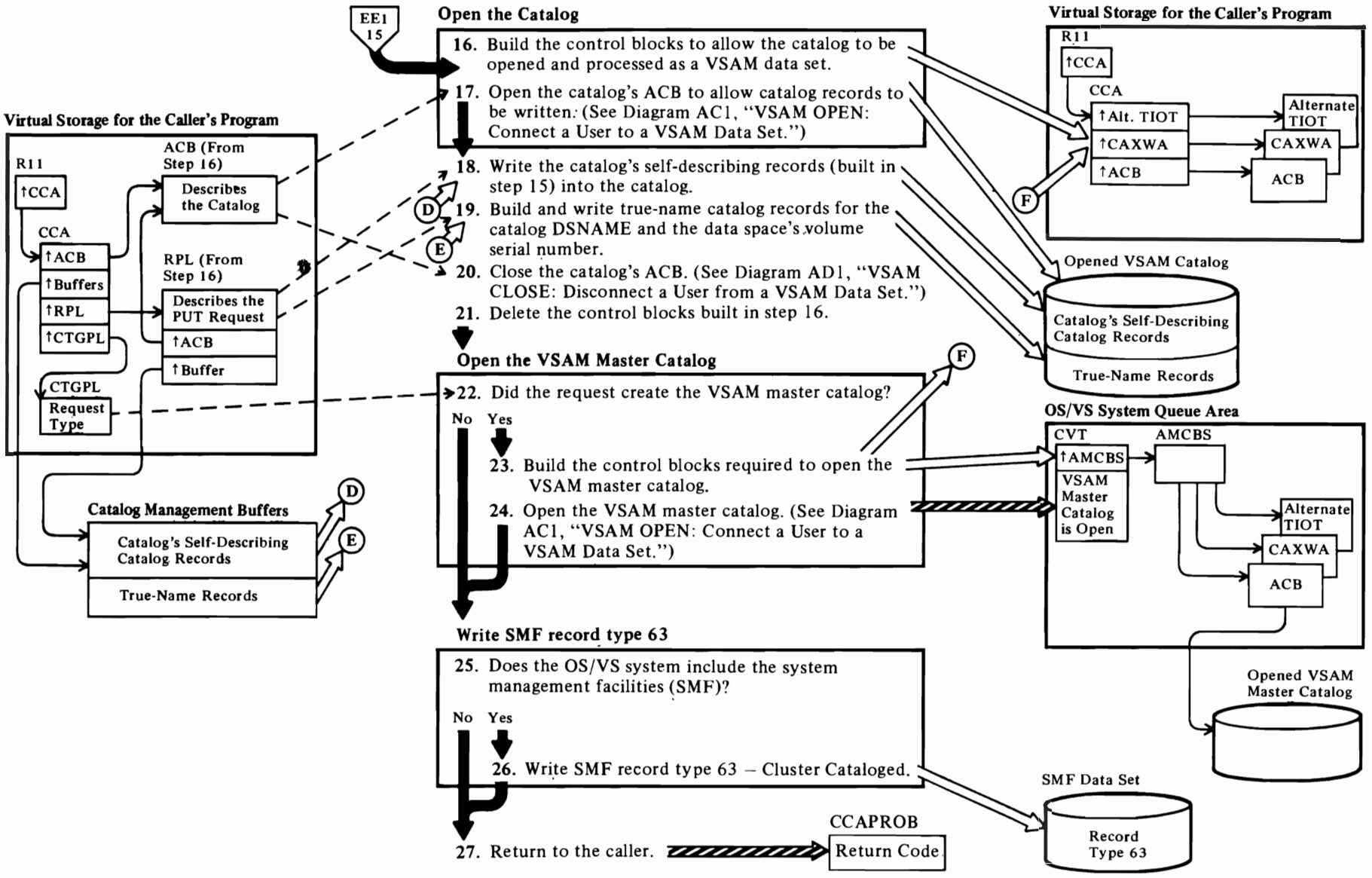
See "Data Areas" for details about the catalog record.

**14 IGG0CLAS: IGGPDCVO**

**15 IGG0CLAS: IGGPDCBE**

See "Data Areas" for details about the first preformatted records in the catalog. These records define the catalog as a key-sequenced VSAM data set, describe the space allocated to the catalog's data records, index records, and true name records, describe the free space control intervals within the catalog's data space, and describe the allocated and unallocated tracks on a catalog's volume.

# Diagram EE2. DEFINE CATALOG: Create a VSAM Catalog



## Notes for Diagram EE2

**16 IGG0CLAE: IGGPDCCB**

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

**17 IGG0CLAE: IGGPDCCB**

The ACB describes the catalog as a VSAM data set to VSAM record management routines.

**18 IGG0CLAE: IGGPDCPR**

**19 IGG0CLAE: IGGPDCPR**

See "Data Areas" for details about the true name catalog record.

**23 IGG0CLAE: IGGPDCME** (calls **IGGPMCO2 (IGG0CLAD))**

**24 IGG0CLAD: IGGPMCO2**

**26 IGG0CLBV: IGGPSMFA**

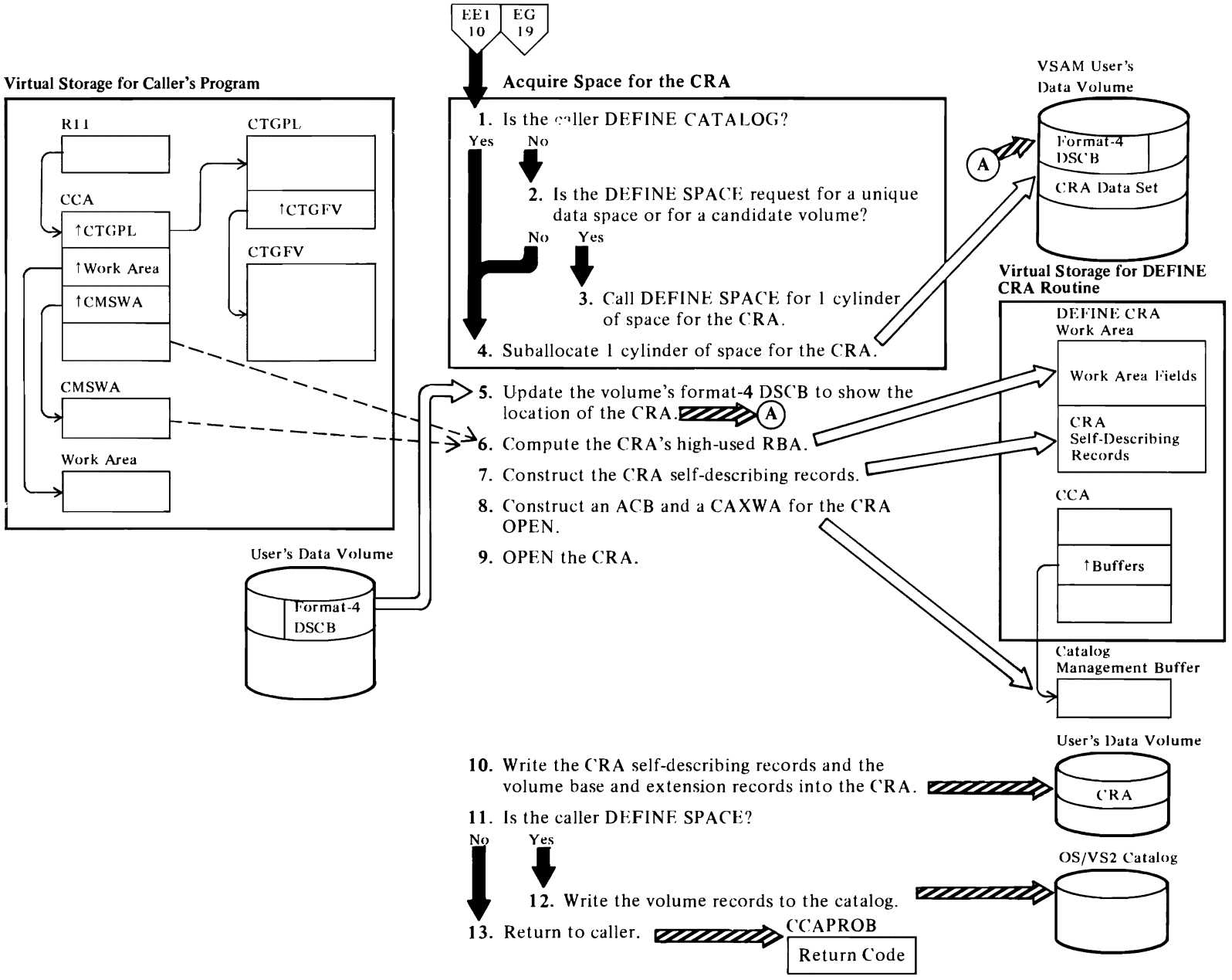
See *OS/VS System Management Facilities (SMF)* for the format of SMF record type 63—VSAM Data Set Cataloged. Record type 63 is written after a VSAM catalog is defined for the cluster, data, and index components and whenever the catalog's definition is altered. definition is altered.

**27**

The catalog management services DEFINE routine sets a return code in the CCA's CCAPROB field and returns to the caller whenever an error is detected.

See "Diagnostic Aids" for details about catalog management return codes.

# Diagram EE3. DEFINE CRA: Create a Catalog Recovery Area



## Notes for Diagram EE3

**1 IGG0CLB4: IGGPSTRG**

**2 IGG0CLB4: IGGPSTRG**

**3 IGG0CLB4: IGGPRDEF**

If the caller is DEFINE SPACE and the request is for a unique data space or for a candidate volume, IGGPRDEF will construct an interface to recursively call DEFINE SPACE (IGGPDEFS) to obtain one cylinder of space for the CRA.

**4 IGG0CLB4: IGGPSBAL**

Suballocate one cylinder for the CRA from the data space.

**5 IGG0CLB4: IGGPFMT4**
**IGG0CLB4: IGGPF4RD, IGGPF4WR**

**6 IGG0CLB4: IGGPCHIU**

**7 IGG0CLB4: IGGPCCIO, IGGPCI15**
**IGG0CLDA: IGGPMODI**

The CRA self-describing records are constructed in main storage; the MODIFY function is called via IGGPMODI to complete the record construction.

**8 IGG0CLB4: IGGPCACB, IGGPCXWA**

An ACB and a CAXWA are built in a catalog management buffer for use by OPEN.

**9 IGG0CLB4: IGGPOCRA**

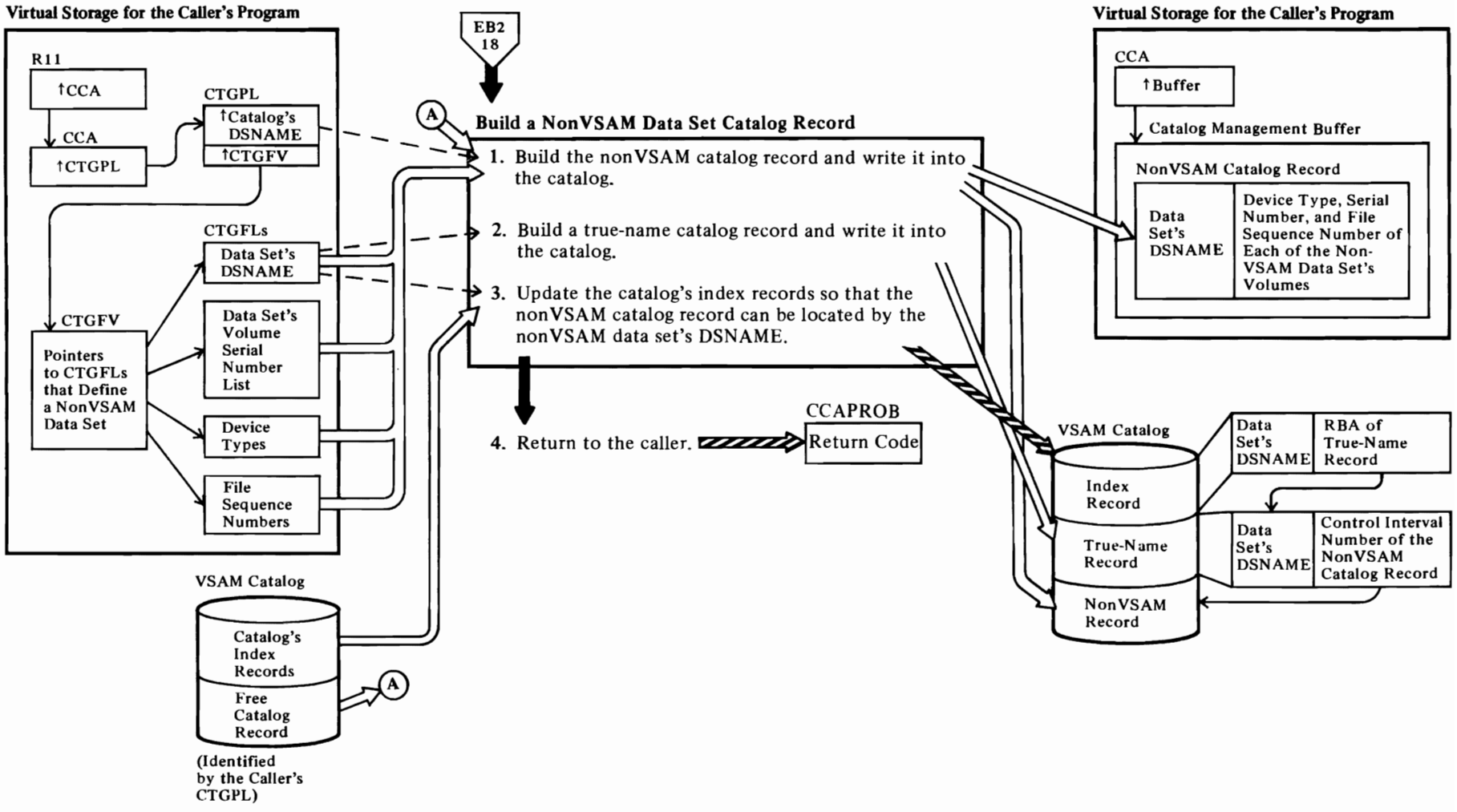
**10 IGG0CLDA: IGGPWCRA**

The CRA self-describing records are written to the volume. The volume base and extension records are also written to the CRA. For a DEFINE CATALOG caller, the records are pointed to by an address array in the CMS workarea; for the DEFINE SPACE caller, the volume records are chained from a pointer in the CCA.

**12 IGG0CLDA: IGGPWCAT**

The chained volume records are also written into the catalog if DEFINE SPACE is the caller.

# Diagram EF1. DEFINE NONVSAM: Define a NonVSAM Data Set in a VSAM Catalog



## Notes for Diagram EF1

This figure describes the processes performed by the catalog management services routines when the user issues the Access Method Services DEFINE NONVSAM command in the form:

```
DEFINE
NONVSAM (parameterlist)
[CATALOG (catname/password)]
```

where:

- *catname* is the name of the catalog that will contain the nonVSAM catalog record that describes the user's nonVSAM data set.
- *password* is the catalog's master or update password, if the catalog is protected by passwords.
- *parameterlist* fields are described in *OS/VS2 SVS Independent Component: Access Method Services*.

See "Data Areas" for details about the nonVSAM catalog record.

**1 IGG0CLBH: IGGPDEFA, IGGPDAIN and IGGPDAVO**

The DEFINE NONVSAM routine builds and transfers the nonVSAM catalog record from a catalog management buffer in virtual storage to the VSAM catalog specified by the caller's DEFINE command.

If the nonVSAM data set is being defined in a recoverable catalog, the catalog's volume serial number and device type are saved in the nonVSAM catalog record.

**2 IGG0CLBH: IGGPDEFA**

See "Data Areas" for details about the true name catalog record.

**3 IGG0CLBH: IGGPDEFA**

See "Data Areas" for details about the catalog index record.

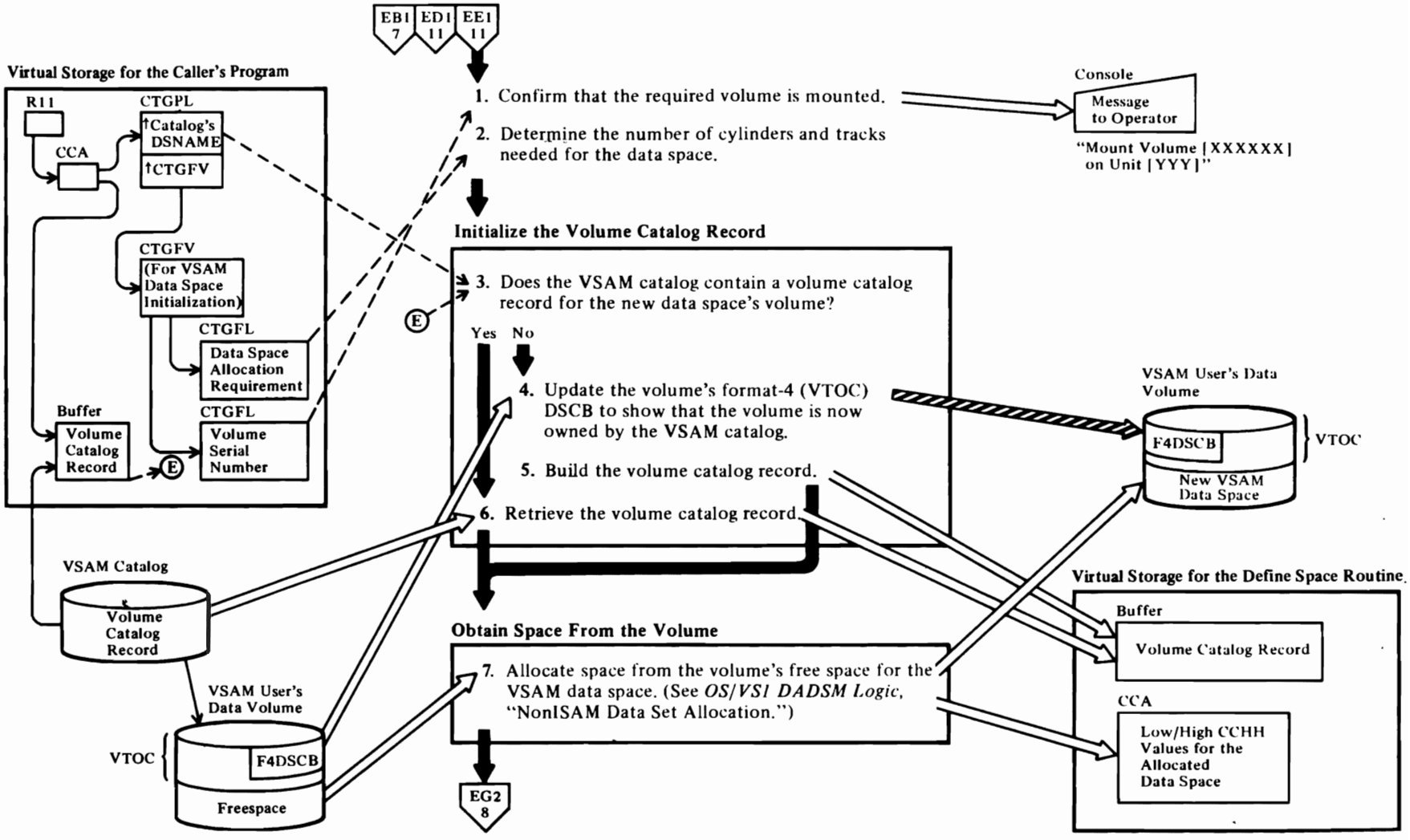
**4**

See "Diagnostic Aids" for details about catalog management return codes.

**Note:** This figure also describes the processes performed when the user issues an IMPORT command to connect a VSAM user catalog (created on another OS/VS system and defined in that system's VSAM master catalog) to the VSAM master catalog. This process is similar to defining a nonVSAM data set, and should not be confused with the process described in Diagram EE1, DEFINE CATALOG: Create A VSAM Catalog.

The VSAM user catalog record is similar to the nonVSAM catalog record, except the record's ID (identifier) value is "U" and there is one volume information set of fields to describe the volume containing the user catalog.

# Diagram EG1. DEFINE SPACE: Initialize a VSAM Data Space



## Notes for Diagram EG1

This diagram describes the processes performed by the catalog management services routines when the user issues the Access Method Services DEFINE SPACE command in the form:

```
DEFINE SPACE
[CATALOG (catname/password)]
parameterlist
```

where:

- *catname* is the name of the catalog that contains the volume catalog record that will describe the VSAM data space.
- *password* is the catalog's master, control interval, or update password, if the catalog is protected by passwords.
- *parameterlist* fields are described in *OS/VS2 SVS Independent Component: Access Method Services.*

**1 IGG0CLAQ: IGGPDEFS** and **IGGPVMTV**

**2 IGG0CLA6: IGGPCRTC**

The user can specify the data space's cylinder and track requirements, or he can specify a number of records and the length of each record, to define the data space's allocation requirements.

**3 IGG0CLAQ**

If a volume catalog record exists for the volume, and if the volume already contains a VSAM data space, a Data Space Group set of fields is added to the volume catalog record to describe the new VSAM data space. A new format-1 (identifier) DSCB is added to the volume's VTOC to describe the new extent.

**4 IGG0CLAQ: IGGPCOBT**

If the volume is a candidate volume (one that is eligible to contain a VSAM data space, but doesn't yet) the volume's format-4 (VTOC) DSCB is updated to show that the VSAM catalog is now the volume owner.

**5 IGG0CLAQ: IGGPIVER**

See "Data Areas" for details about the volume catalog record.

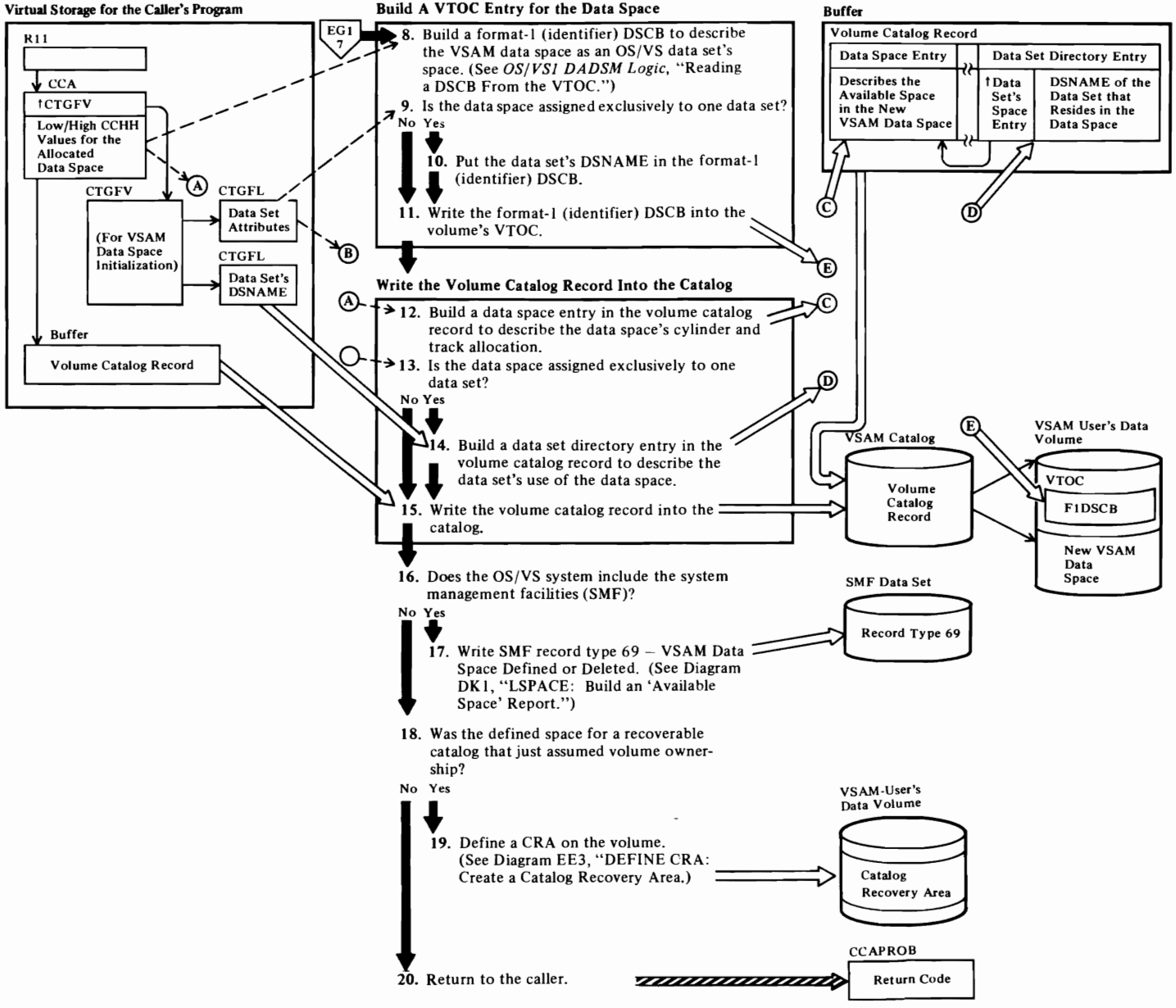
**6 IGG0CLAQ: IGGPDEFS** (calls **IGGPSCAT (IGG0CLAH)**)

The volume catalog record is identified by the volume's serial number.

**7 IGG0CLA6: IGGPBJFB**
**IGG0CLAQ: IGGPDEFS**

Construct a JFCB and call the DADSM allocate function.

## Diagram EG2. DEFINE SPACE: Initialize a VSAM Data Space



## Notes for Diagram EG2

**8 IGG0CLAQ: IGGPDEFS**

The format-1 (identifier) DSCB describes the VSAM data space as an OS/VS data set to the OS/VS DADSM routines.

**9 IGG0CLAQ: IGGPCOBT**

**10 IGG0CLAQ: IGGPCOBT**

A VSAM data space assigned exclusively to one data set is, to DADSM, the same as one of the extents of an OS/VS data set. The data space is described by a format-1 (identifier) DSCB that contains the data set's dsname. If a data space can be assigned to more than one data set, its format-1 (identifier) DSCB contains a dsname generated by the DEFINE SPACE routine.

**11 IGG0CLAQ: IGGPCOBT**

**12 IGG0CLAQ: IGGPCSHG** and **IGGPCSDG**

See "Data Areas" for details about the volume catalog record and Data Space Group set of fields.

**14 IGG0CLAQ: IGGPCDSD**

The volume catalog record contains the identifier of each data set that resides (in part or in full) on the volume.

See "Data Areas" for details about the volume catalog record.

**17 IGG0CLAQ: IGGPDEFS** (calls **IGGPLSP (IGG0CLBK)**)

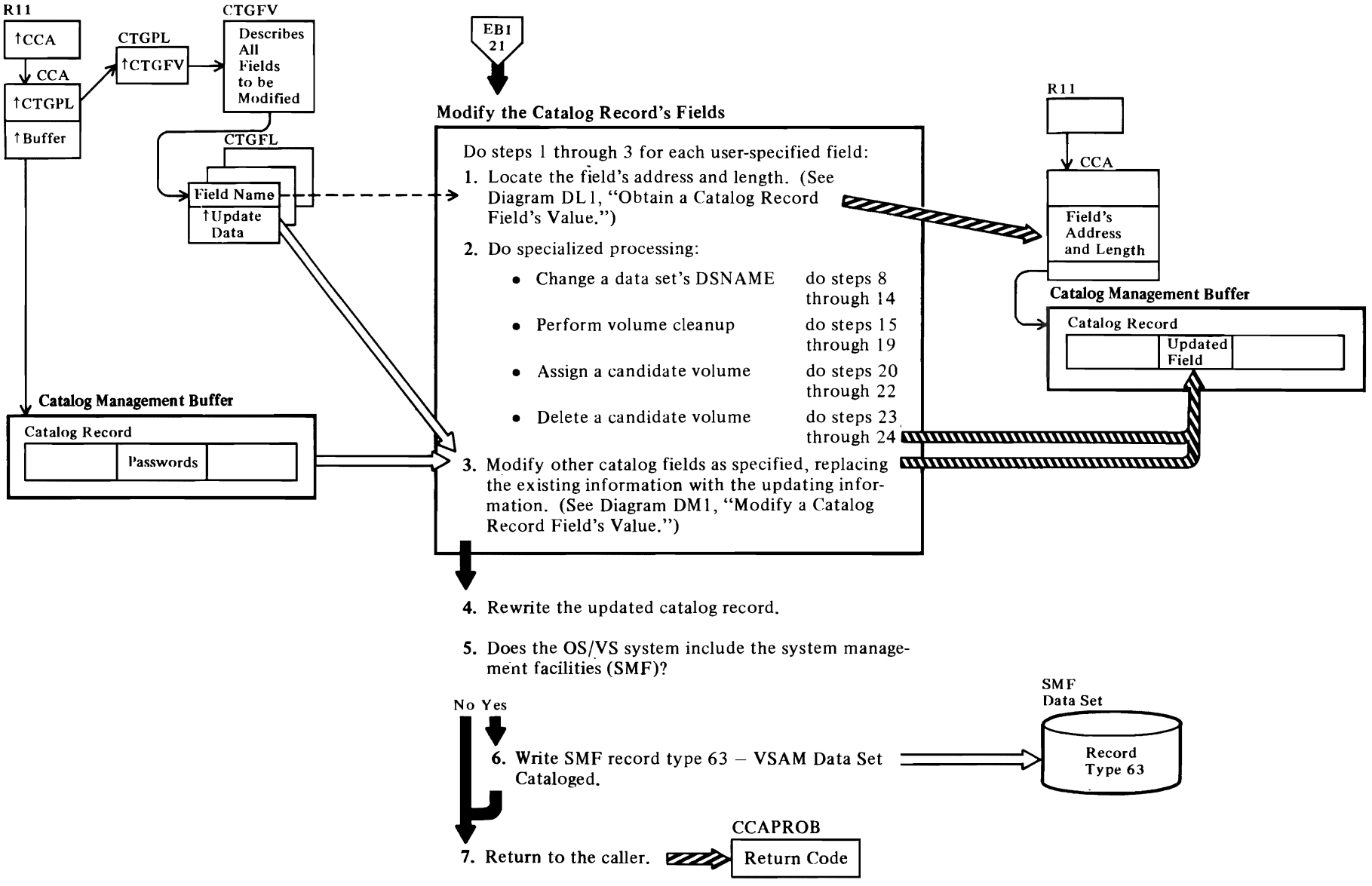
See *OS/VS System Management Facilities (SMF)* for the format of SMF record type 69—VSAM Data Space Defined or Deleted. Record type 69 is written when a VSAM data space is created or when its available space is allocated to a VSAM data set or index.

**19**

A catalog recovery area is defined on the volume on which the space is being defined if ownership is taken by a recoverable catalog and the defined space does not contain the catalog.

See "Diagnostic Aids" for details about catalog management return codes and error codes.

# Diagram EH1. ALTER: Modify a Catalog Record



## Notes for Diagram EH1

The ALTER command enables the user to modify some of the information he established when he created a VSAM data set.

This figure describes the processes performed by catalog management services routines when the user issues the Access Method Services ALTER command in the form:

```
ALTER
(entryname/password)
[CATALOG (catname/password)]
parameterlist
```

or

```
ALTER
entryname/password
FILE(dname)
REMOVEVOLUMES (volser[bvolser...])
```

where:

- *entryname* is the dsname or volume serial number that identifies the catalog record to be modified.
- *password* is the record's (identified by entryname) master password.
- *CATALOG* identifies the VSAM catalog that contains the record to be modified, and supplies the correct password for that catalog.
- *parameterlist* fields are described in *OS/VS2 SVS Independent Component: Access Method Services*.

or:

- *entryname,* for the volume cleanup function, is the name of the OS/VS master catalog.
- *password* is the master password of the master catalog specified in entryname.
- *dname* specifies the name of a DD statement that identifies the volume(s) to be scratched. This parameter is required for the volume cleanup function.
- *volser* specifies the volume serial number(s) of the volume(s) on which all VSAM space is to be removed and VSAM ownership is to be relinquished. Volumes owned by the master catalog cannot be specified on a cleanup request.

**1 IGG0CLBD: IGGPALT**

See "Data Areas" for details about the CCA, CTGPL, and CTGFL.

**2 IGG0CLBD: IGGPALT**

When the data set name or allocated candidate volumes are changed, other catalog records besides the data set catalog record must be updated.

**3 IGG0CLBD: IGGPALT**

**4 IGG0CLBD: IGGPALMD**

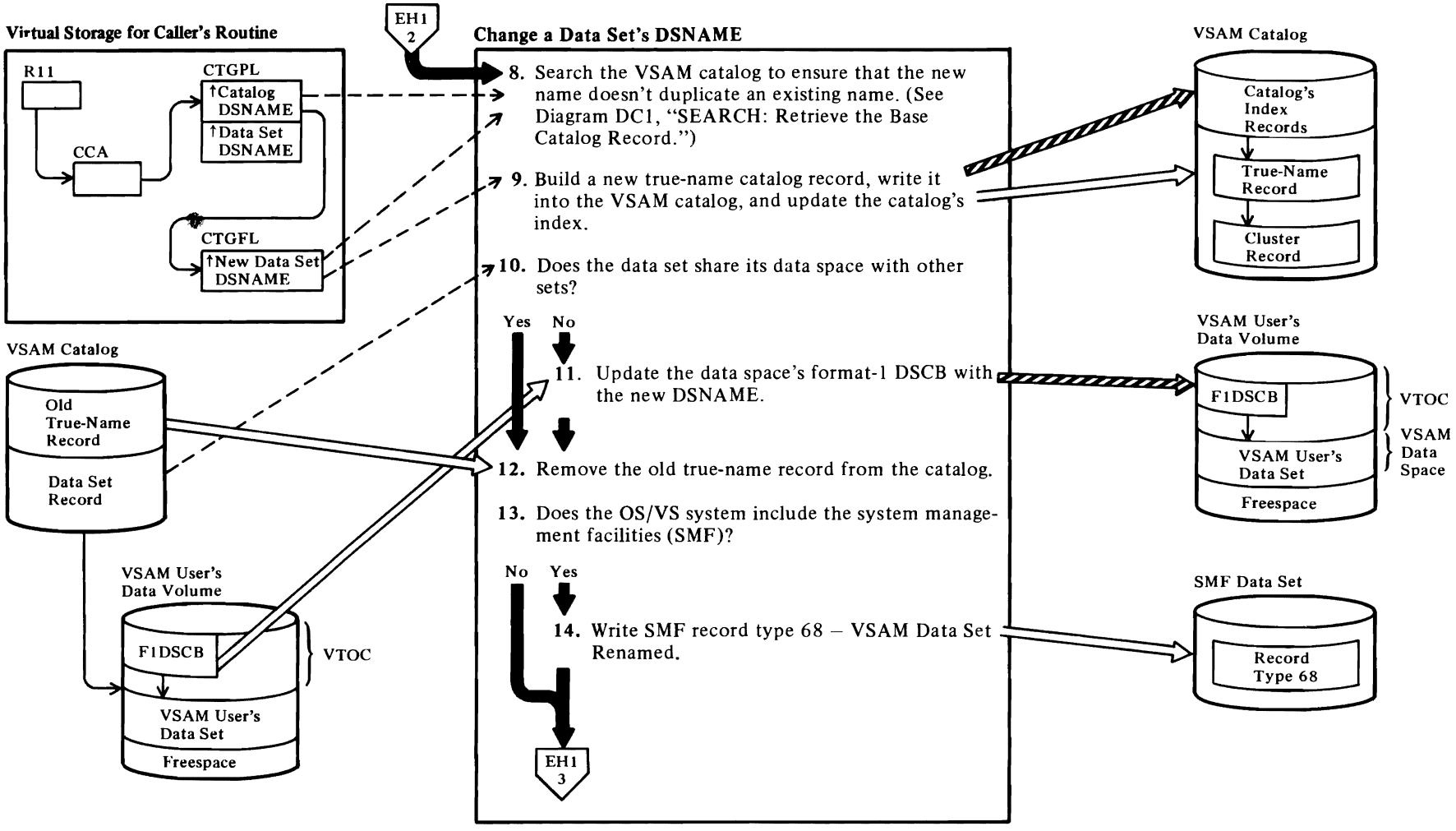
**6 IGG0CLBD: IGGPALT**

See *OS/VS System Management Facilities (SMF)* for the format of SMF record type 63 VSAM Data Set Cataloged. Record type 63 is written after a nonVSAM data set, cluster, or catalog is defined and when the definition is altered. One SMF record is written for each modified catalog record.

**7 IGG0CLBD: IGGPALT**

See "Diagnostic Aids" for details about catalog management return codes and error codes.

# Diagram EH2. ALTER: Modify a Catalog Record



## Notes for Diagram EH2

**8 IGG0CLBD: IGGPALNM**

The catalog specified by the ALTER command's CATALOG parameter is examined. No other catalogs are examined.

**9 IGG0CLBD: IGGPALNM**

See "Data Areas" for details about the VSAM catalog organization, the catalog index record, the true name catalog record, and the data catalog record.

**11**

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

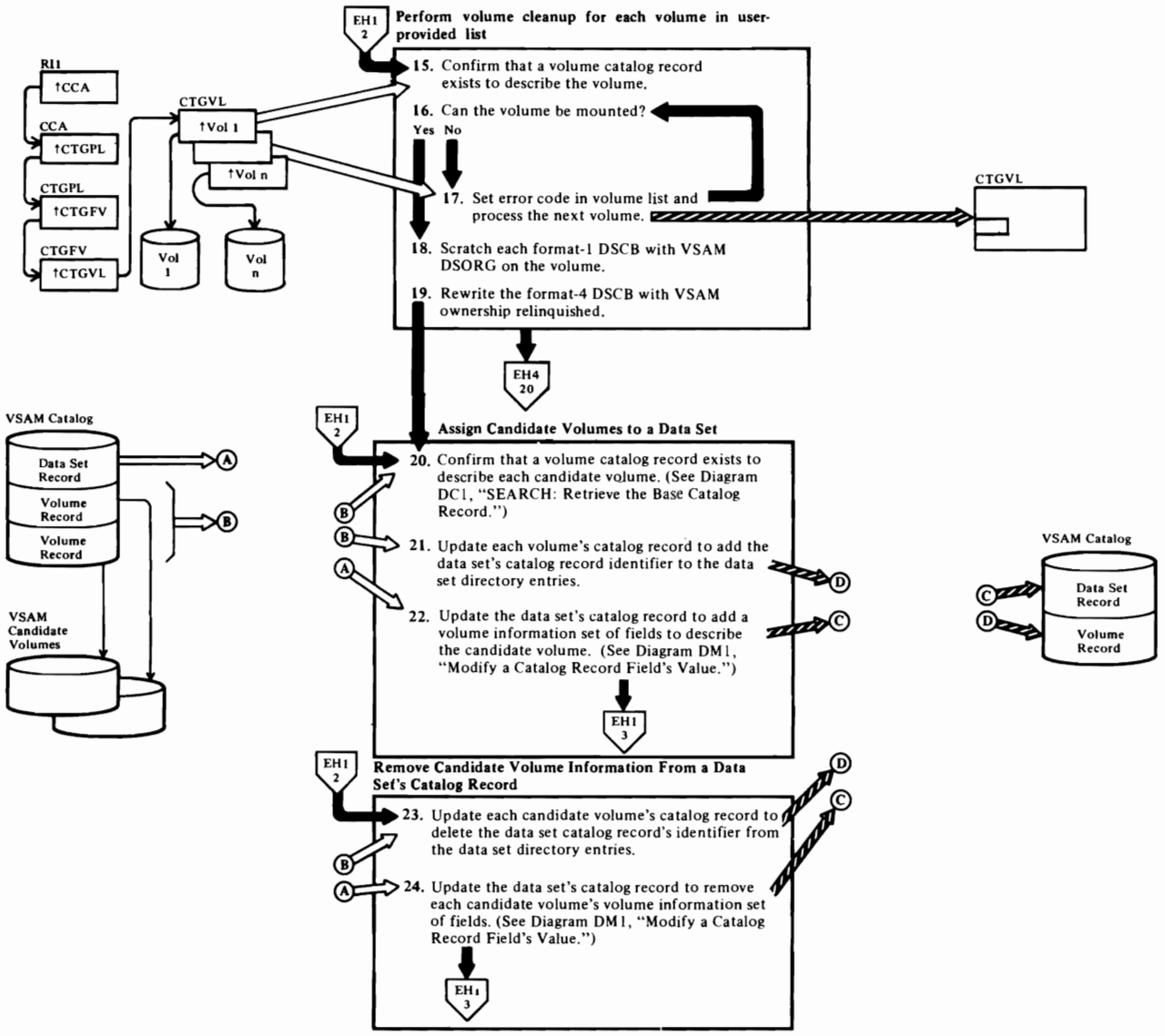
**12 IGG0CLBD: IGGPALF1**

The name and control interval fields in the data set's true name record are set to 0, and the true name record's identifier field is set to C"F".

**14 IGG0CLBD: IGGPALT**

See *OS/VS System Management Facilities (SMF)* for the format of SMF record type 68 VSAM Data Set Renamed.

## Diagram EH3. ALTER: Modify a Catalog Record



## Notes for Diagram EH3

**15 IGG0CLBE: IGGPALVL**

If volume cleanup is required, IGGPALVL calls IGGPVRD (IGG0CLBN). Each volume in the user-provided volume serial number list is processed, if its volume catalog record exists.

**16 IGG0CLBN: IGGPVRD**

The volume(s) specified in the user-provided volume serial number list is mounted. After a successful mount, IGGPRVRD calls IGGPVRCV (IGG0CLBN).

**17 IGG0CLBN: IGGPVRD**

If a volume cannot be mounted, a volume-not-mounted condition is indicated in the volume serial list and passed back to the user.

**18 IGG0CLBN: IGGPVRCV**

Scratch each format-1 DSCB that has VSAM DSORG indicated.

**19 IGG0CLBN: IGGPVRCV**

The VSAM ownership, VSAM timestamp, the sum TT fields, and, if applicable the recovery timestamp are removed from the format-4 DSCB and the DSCB is rewritten on the volume.

**20 IGG0CLBD: IGGPALT** (calls **IGGPALVL (IGG0CLBE))**

**IGG0CLBE: IGGPALVA**

IGGPALVA calls IGGPSALL (suballocate) to assign the candidate volume to the data set. If a volume catalog record does not exist for the candidate volume, the suballocate routine returns an error code.

See Diagram EG1, DEFINE SPACE: Assign a VSAM Data Space to a Catalog, for details on how a volume catalog record is built.

**21 IGG0CLBE: IGGPALVA**

The volume catalog record contains a data set directory that describes each VSAM data set's use of the volume's VSAM space.

See "Data Areas" for details about the volume catalog record.

**22 IGG0CLBE: IGGPALVA** (calls **IGGPALSA (IGG0CLBE), IGGPSALL (IGG0CLAR),** and **IGGPMOD (IGG0CLAV))**

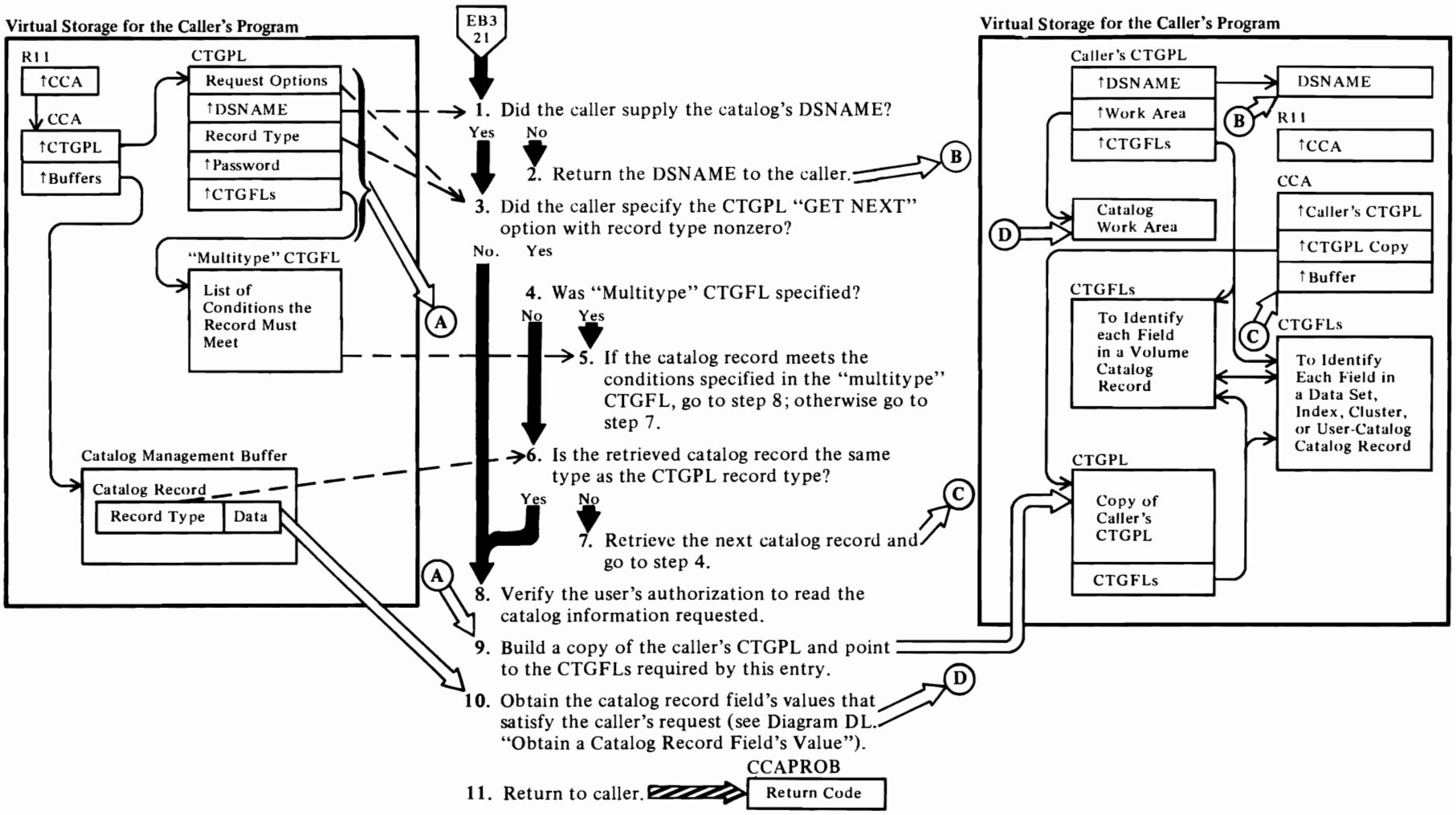
See "Data Areas" for details about the data catalog record and its volume information set of fields.

**23 IGG0CLBN: IGGPALVR**

**24 IGG0CLBN: IGGPALVR** (calls **IGGPMOD (IGG0CLAV))**

See "Data Areas" for details about the data catalog record and volume information set of fields.

# Diagram EI1. LISTCAT: Retrieve a Catalog Record's Contents



## Notes for Diagram EI1

The LISTCAT command enables the user to list all or a part of a VSAM user or master catalog's contents. This figure describes the processes performed by the catalog management services routines when the user issues the Access Method Services LISTCAT command in the form:

```
LISTCAT
[CATALOG (catname[/password][dname])]
[OUTFILE (dname) |
ENTRIES(entryname[/password])]
[CLUSTER | DATA | INDEX | SPACE |
NONVSAM | USERCATALOG |
ALTERNATEINDEX | PATH]
[ALL | NAME | VOLUME | ALLOCATION |
HISTORY]
[CREATION(days)]
[EXPIRATION(days)]
[NOTUSABLE]
```

where:

- *CATALOG* identifies the VSAM catalog that contains the user-requested data:
  - *catname* is the dsname of a VSAM user catalog or the VSAM master catalog.
  - *password* is one of the catalog's passwords, if the catalog is password protected. If the user requests password information from the catalog, he must specify the catalog's master password. All other catalog information is available to the user if he specifies the catalog's read password.
  - *dname* specifies the DD name of the catalog to be listed.
- *OUTFILE* specifies an optional alternate listing output data set and identifies it by dname.
- *ENTRIES* is a list of catalog record identifiers:
  - *entryname* is the dsname or volume serial number that identifies a catalog record. If the LISTCAT command includes an ENTRIES parameter list, only those catalog records identified by entrynames are listed.
  - *password* is one of the catalog record's passwords. If the catalog's password is supplied, the catalog record's password is ignored. Otherwise, the catalog record's master password allows its password information to be listed; its read password allows all other information to be listed, but suppresses the password information.
- [CLUSTER | DATA | INDEX | SPACE | NONVSAM | USERCATALOG ALTERNATEINDEX | PATH]

  is a list that specifies the types of catalog records to be listed. If both the ENTRIES and this 'types list' parameter lists are specified, only those catalog records that are identified by an entryname and are included in the list of types are returned to the caller.
- [ALL | NAME | VOLUME | ALLOCATION | HISTORY] specifies what part of each record to list.
- CREATION specifies the minimum age an object must be to be listed.
- EXPIRATION specifies the maximum number of days remaining before expiration an object may have to be listed.
- NOTUSABLE specifies that only those data or index entries made not usable by a force delete (as opposed to entries made not usable by system failure, etc.) are to be listed.

The LISTCAT parameters are described in *OS/VS2 SVS Independent Component: Access Method Services*.

**1 IGG0CLBQ: IGGPLSTC**

If the first character of the catalog's name is blank, the caller wants the catalog name returned.

**3 IGG0CLBQ: IGGPLSTC**

If the caller did not specify the CTGPL "GET NEXT" option and a nonzero CTGPL record type, only the one original entry pointed to by the CTGPL is listed.

**5 IGG0CLBQ: IGGPLSTC**

The multitype CTGFL specifies conditions which must be met by the retrieved record. The possible conditions are:

a. Must be of a specified record type
b. Must have a certain usability state
c. Must meet a creation date value
d. Must meet an expiration date value

**6 IGG0CLBQ: IGGPLSTC**

If there is no multitype CTGFL, the retrieved record must be the same record type as the CTGPL record type.

**7 ICC0CLBQ: IGGPLSTC**

The next record is retrieved by specifying the GET NEXT option to VSAM Record Management.

**9 IGG0CLBQ: IGGPLSTC**

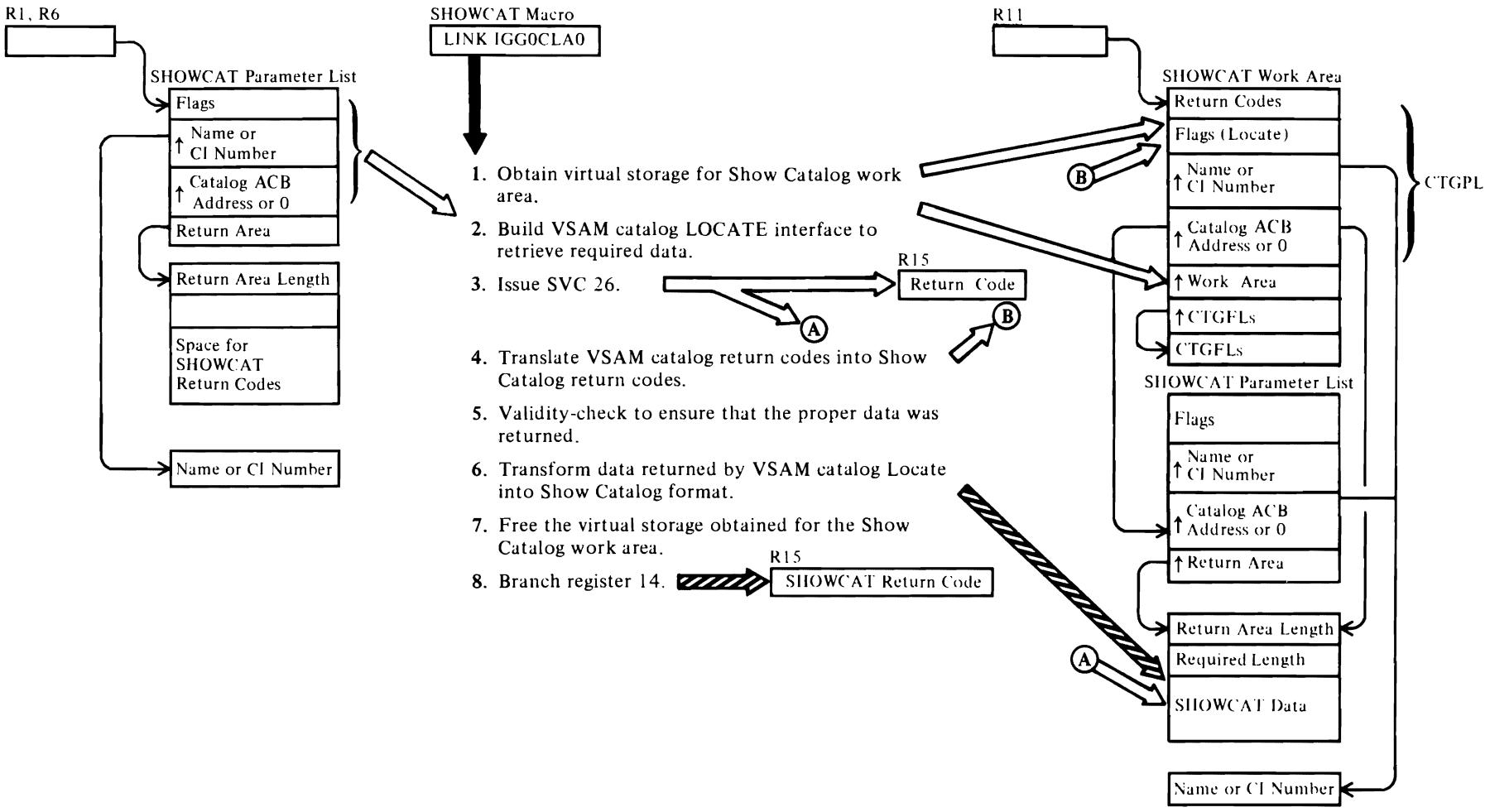
If the entry is a volume record, only volume CTGFLs are pointed to from the CTGPL; otherwise, only nonvolume CTGFLs are pointed to from the CTGPL.

**11 IGG0CLBQ: IGGPLSTC**

When all requested information has been retrieved, the Listcat routine sets a return code in CCAPROB and returns to the caller, VSAM Catalog Management Services Common Processing.

See "Diagnostic Aids" for details about catalog management return codes and error codes.

# Diagram EI2. Show Catalog Processing



## Notes for Diagram EI2

The Show Catalog processor, IGG0CLA0, enables the user to obtain selected information from the VSAM catalog. This specialized user interface is mapped by macro IGGSHWPL and is invoked by the SHOWCAT macro. See *OS/VS2 SVS Independent Component: VSAM Options for Advanced Applications* for a complete description of the SHOWCAT macro.

The SHOWCAT macro generates the Show Catalog parameter list and issues a LINK to module IGG0CLA0. Note that this module is *not* a part of the VSAM supervisory load module for SVC26, IGG0CLC9.

**1** The Show Catalog processor builds its conditional GETMAIN parameter list in the user-provided return area.

**2** The VSAM Catalog Locate interface is built in the Show Catalog work area acquired in step 1. The CTGPL work area address points to the user return area.

**3** The VSAM Catalog Locate function sets the required return area length field in the user return area and places the requested data into this return area.

**4** The Show Catalog processor has equivalent error codes for VSAM catalog return codes.

**5** The validity check ensures that data was actually returned and that the proper entry type is being requested.

**6** The transformation causes upgrade associations and nonupgrade associations to be returned in a consistent format.

**7** The Show Catalog processor always obtains a fixed amount of virtual storage for its work area. The user is responsible for providing an area of sufficient size to contain the returned data. If his area is not of sufficient size, he can use the required return area length field to obtain enough virtual storage to reissue his request.

For additional information about topics related to Show Catalog processing, see:

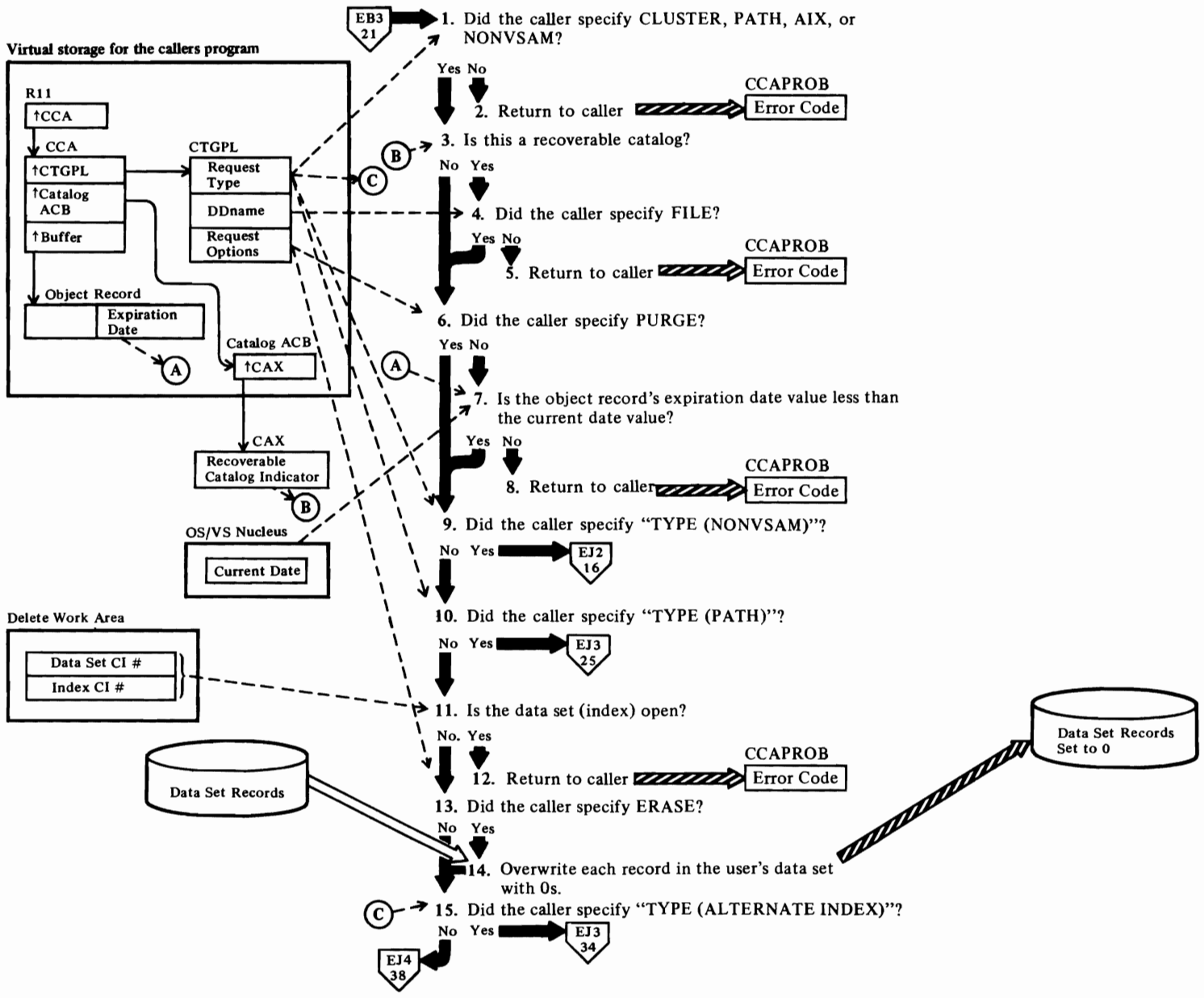
"Data Areas."

Catalog parameter list (CTGPL) format and description Catalog

"Diagnostic Aids."

Catalog management return codes

# Diagram EJ1. DELETE: Remove a VSAM or NonVSAM Data Set



## Notes for Diagram EJ1

The DELETE command enables the user to remove from the catalog all information about a specified VSAM object or nonVSAM data set.

This figure describes the processing performed by the catalog management routines when the user issues the Access Method Services DELETE command in the form:

```
DELETE
(entryname/password)
[CATALOG(catname/password)]
[CLUSTER | AIX | PATH | NONVSAM]
```

where:

- *entryname* is the data set name of the VSAM object or nonVSAM data set to be deleted.
- *password* is the master password of the VSAM object to be deleted.
- *CATALOG* identifies the catalog that contains the record to be deleted and specifies the catalog's password.
- *CLUSTER | AIX | PATH | NONVSAM* specifies the type of object to be deleted. Deletion of these types is described in this diagram. To delete VSAM data spaces on a volume, see Diagram EK1; to delete a VSAM catalog, see Diagram EL1.

The DELETE command's parameters are described in *OS/VS2 SVS Independent Component: Access Method Services.*

**1 IGG0CLBG: IGGPDEL**

If the CATALOG parameter is not specified, the catalog record identified by the ENTRY parameter's entryname is found by a search of each catalog named by the user's JCL JOBCAT and STEPCAT DD statements, followed by a search of the VSAM master catalog. The catalog record identifier is examined to determine the record type and verify that the TYPE parameter, if specified, is correct.

**3 IGG0CLBG: IGGPDEL**

If the catalog is a recoverable catalog, a DD statement must be specified for the CRA (catalog recovery area) volume.

**6 IGG0CLBG: IGGPDEL**

If the user specified PURGE, the data set's expiration date is ignored. See *OS/VS2 SVS Independent Component: Access Method Services* for details about the PURGE and RETAIN parameters.

**7 IGG0CLBG: IGGPDEL**

If the user who created the data set specified the expiration date, the data set cannot be deleted until after that date (unless the PURGE parameter is specified; see step 6).

**9 IGG0CLBG: IGGPDEL**

If the request type is nonVSAM, go to step 16 (EJ2). Open determination and Erase processing do not apply to nonVSAM data sets.

**10 IGG0CLBG: IGGPDEL** (calls **IGGPDEPT (IGG0CLB5)**)

If the request type is PATH, go to step 25 (EJ3). Open determination, Erase, volume record, and release space processing do not apply to path catalog records.

**11 IGG0CLBG: IGGPDEL** (calls **IGGPDOPN (IGG0CLBG)**)

If the data set or the index of the alternate index or cluster is already opened, the deletion of the VSAM data set will not be allowed.

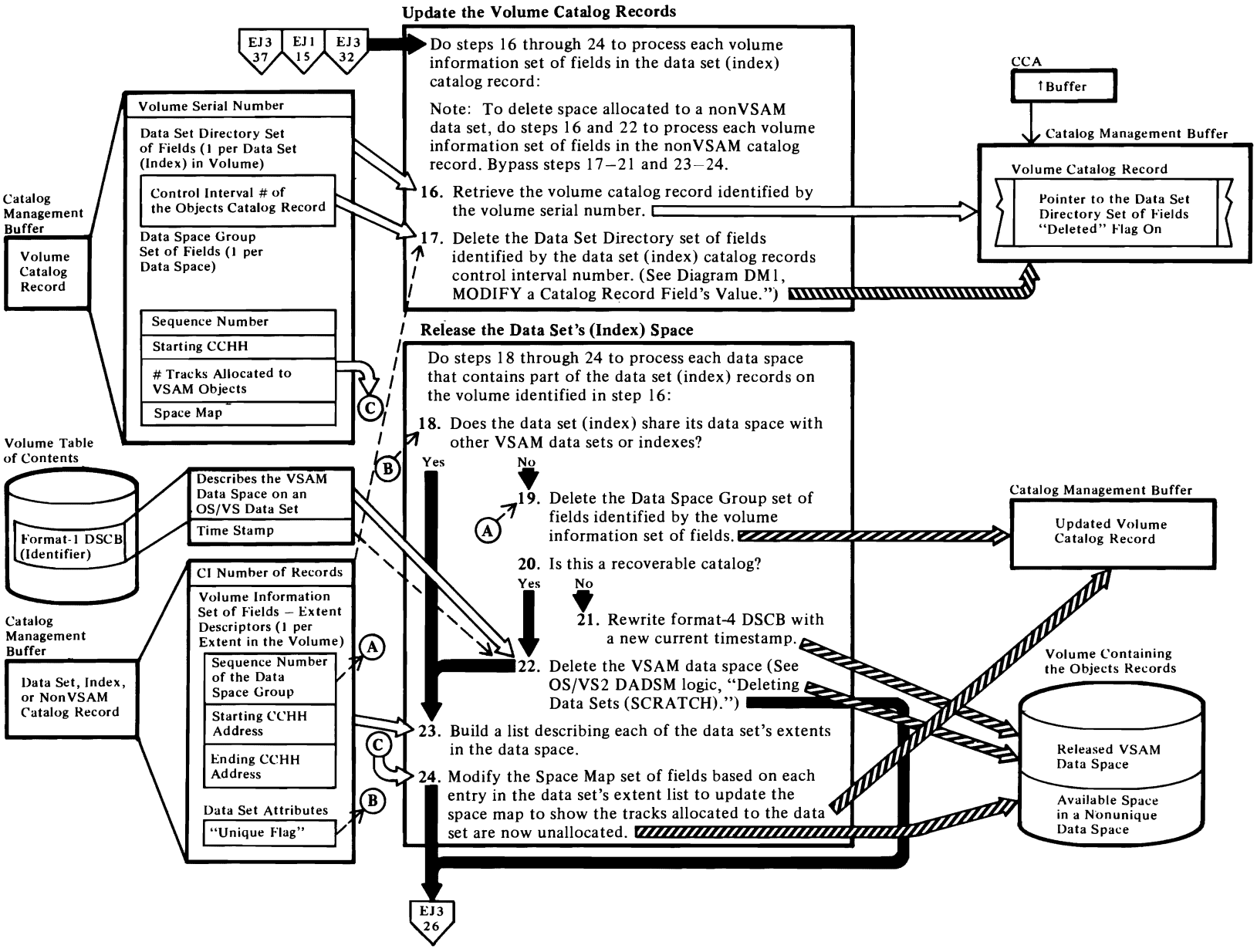
**13 IGG0CLBG: IGGPDEL** (calls **IGGPERAS (IGG0CLBG)**)

Each of the data set's control areas is overwritten with zeros.

**15 IGG0CLBG: IGGPDEL** (calls **IGGPDEAX (IGG0CLB5)**)

Go to step 34 (EJ3) to explicitly delete an alternate index from the catalog. An alternate index is similar to a key-sequenced cluster, except a base cluster is always associated with an alternate index.

# Diagram EJ2. DELETE: Remove a VSAM or NonVSAM Data Set



## Notes for Diagram EJ2

**16 IGG0CLA7: IGGDEVG**

Each volume information set of fields is retrieved from the data set (index) catalog record. If the data set (index) is unique, the volume is mounted.

**IGG0CLA7: IGGPVMSC**

The volume catalog record is retrieved by forming a 44-byte true name from the volume serial number field in the volume information set of fields. The 44-byte true name for the volume catalog record is the 6-byte volume serial number followed by 38 zeros.

**17 IGG0CLA7: IGGPDEDD**

The volume catalog record also contains a Data Set Directory set of fields to describe each VSAM data set that is contained, partially or completely, on the volume. If the volume is a candidate volume for a data set or index, the data set or index is not described by a Data Set Directory set of fields.

**18 IGG0CLA7: IGGPVMSC**

**19 IGG0CLA7: IGGPDEDD**

If the data set's (index's) space is not shared (the "unique" flag in the data set attributes field is on), the data space group set of fields described by the volume information set of fields (sequence number of Data Space Group field) is deleted.

**20 IGG0CLA7: IGGPVMSC**

If the catalog is a recoverable catalog, the timestamp in the volume catalog record and format-4 DSCB in the volume's VTOC are not altered.

**21 IGG0CLA7: IGGPDF4T**

A new timestamp is obtained from the system and the old timestamp in the volume catalog record and format-4 DSCB in the volume VTOC are rewritten with the new current timestamp.

**22** The OS/VS DADSM Scratch routines are called by issuing SVC29. The extents of the data space's identifier (format-1 DSCB) and extension (format-3 DSCB) are added to a format-5 DSCB. A free VTOC record (format-0 DSCB) is written over each of the data space's format-1 and format-3 DSCBs.

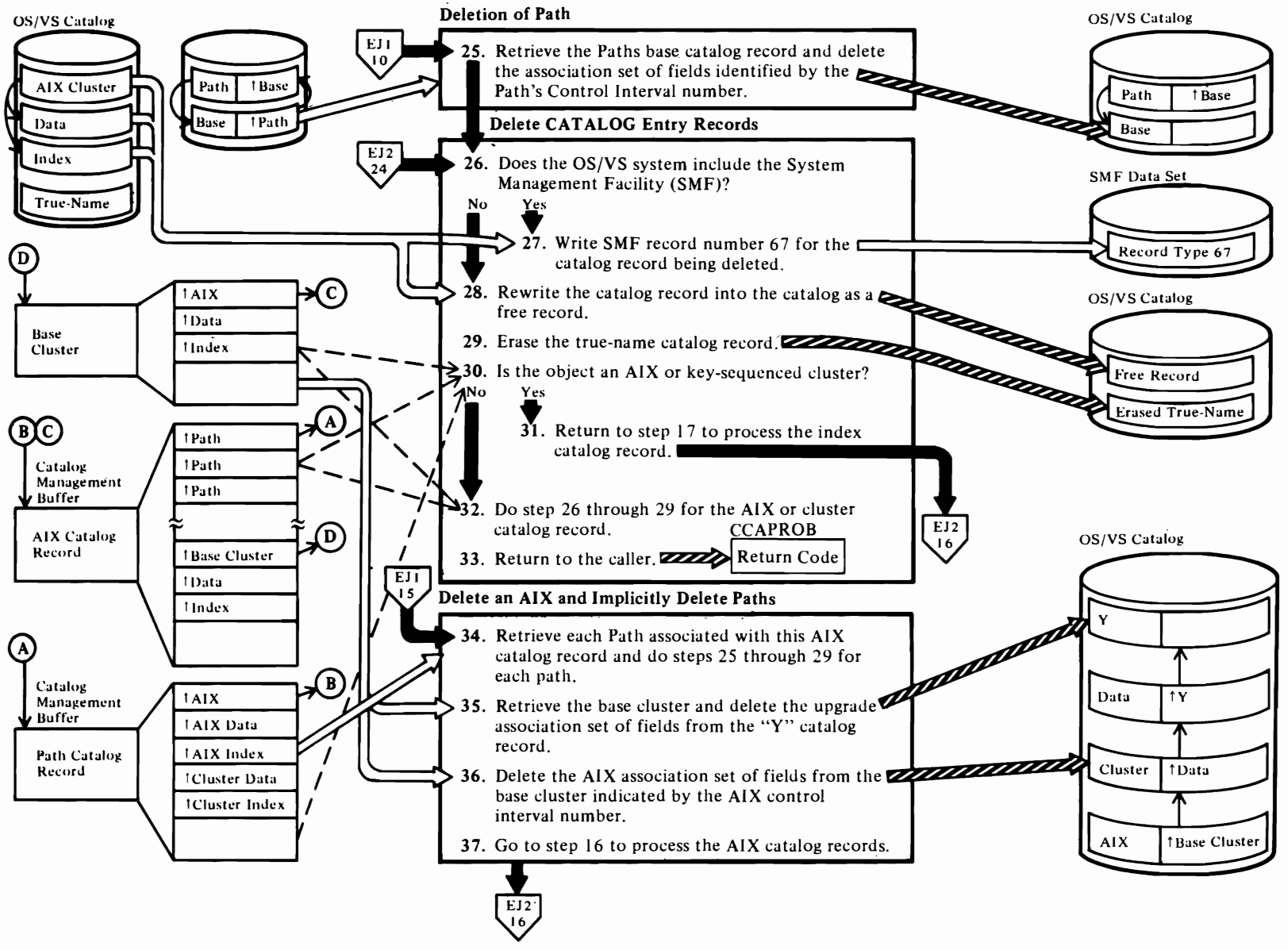
**23 IGG0CLBF: IGGPSSCR**

Each entry in the list identifies one of the data set's (index's) extents in one of the data spaces on the volume.

**24 IGG0CLBF: IGGPSSCR**

Each of the data space's extents is described in the Data Space Group set of fields.

# Diagram EJ3. DELETE: Remove a VSAM or NonVSAM Data Set



## Notes for Diagram EJ3

**25 IGG0CLB5: IGGPDEPT**

Retrieve the path catalog record's "base" catalog record (the base catalog record can be either a cluster or an alternate index catalog record) and delete the association set of fields in the base catalog record that describes the path's control interval number. This action unchains the path catalog record from the catalog structure.

**27 IGG0CLB5: IGGPDCLS**

See *OS/VS System Management Facilities (SMF)* for the format of SMF record type 67. Record type 67 is written when a VSAM cluster, path, alternate index, or nonVSAM data set defined in a VSAM catalog is deleted.

**29 IGG0CLB5: IGGPDCLS**

The DELETE routines erase the data set's true-name record and delete all references to the data set's DSNAME in the catalog's index.

**30 IGG0CLB5: IGGPDCLS**

If the object catalog record type is an alternate index or a key-sequenced cluster, steps 16 through 29 are performed to delete the index catalog record.

**32 IGG0CLB5: IGGPDCLS**

Steps 26 through 29 are performed for the cluster or alternate index catalog records.

**34 IGG0CLB5: IGGPDEAX** (calls **IGGPDIPT (IGG0CLB5)**)

When an alternate index is deleted, all paths associated with the alternate index are implicitly deleted first. This process is performed by retrieving each path record and completing steps 25 through 29.

**35 IGG0CLB5: IGGPDUPG** (calls **IGGPUDEL (IGG0CLB1)**)

The alternate index base cluster is retrieved and the upgrade association set of fields described by the alternate index's data set (index) control interval numbers is deleted from the "Y" catalog record associated with the cluster's data set.

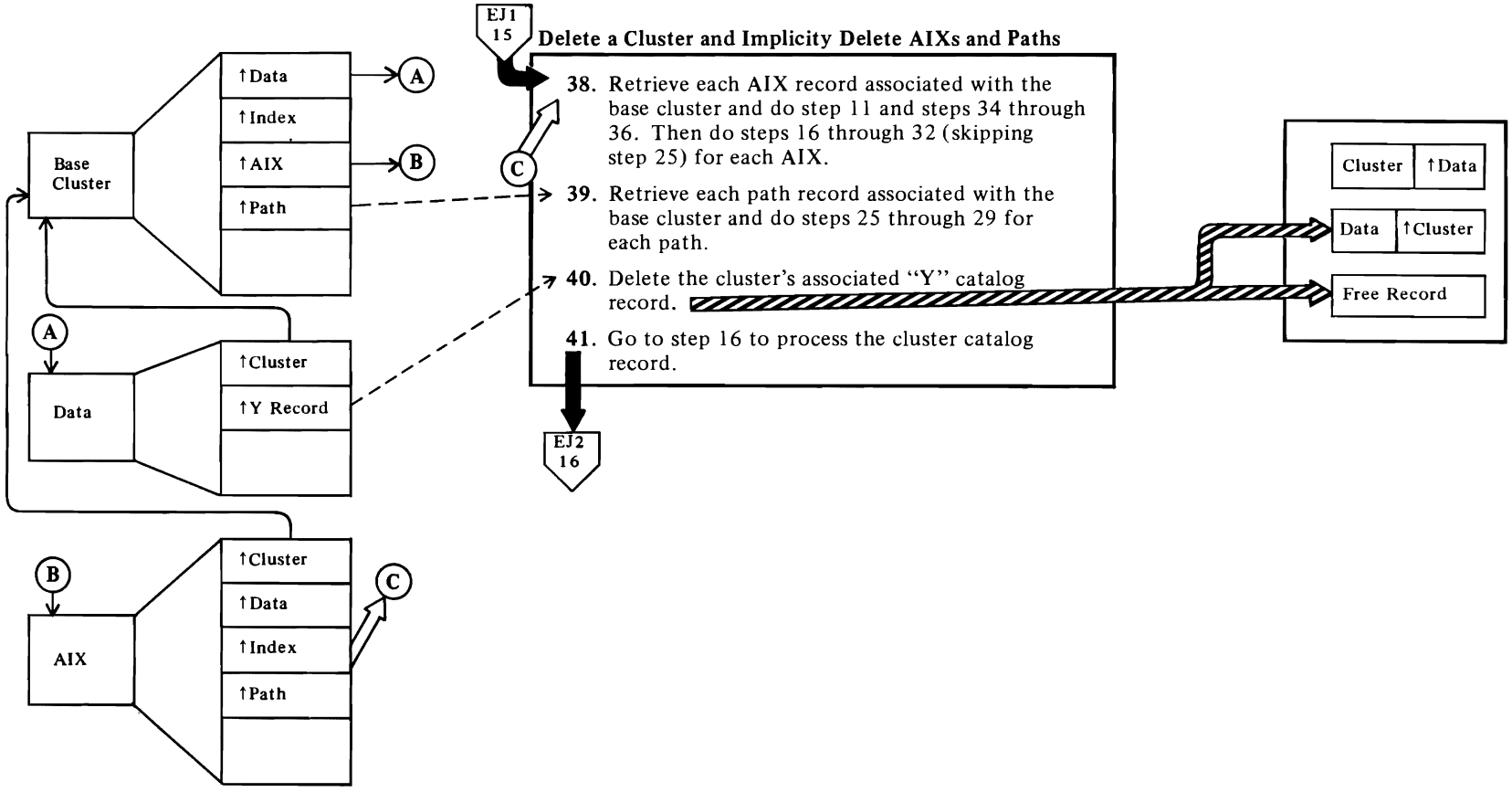
**36 IGG0CLB5: IGGPDEAX**

Delete the association set of fields in the base catalog record that describes the alternate index control interval number. This action unchains the alternate index catalog record from the catalog structure.

**37 IGG0CLB5: IGGPDEAX**

Complete steps 16 through 33 to process the alternate index data set, and index catalog records.

# Diagram EJ4. DELETE: Remove a VSAM or NonVSAM Data Set



## Notes for Diagram EJ4

**38** When a cluster is deleted, all associated alternate index paths are implicitly deleted first, followed by all the associated alternate indexes and the associated cluster paths.

Each alternate index record is retrieved and step 11 is performed to assure that the data sets (index) are not opened. Steps 34 through 36, 16 through 24, and 26 through 32 are performed to implicitly delete all associated alternate indexes and alternate index paths.

**39** **IGG0CLBG: IGGPDEL** (calls **(IGGPDIPT (IGG0CLB5))**

Retrieve each cluster's associated path record and perform steps 25 through 29 for each path.

**40** **IGG0CLBG: IGGPDEL** (calls **IGGPDUPG (IGG0CLB5))**

The "Y" catalog record associated with the cluster data set record and the related association set of fields in the data set record is deleted.

**41** **IGG0CLBG: IGGPDEL** (calls **(IGGPDCLS (IGG0CLB5))**

To process the cluster, data set, and index catalog records, perform steps 16 through 33 (skip step 25).

For additional information about topics related to DELETE processing see:

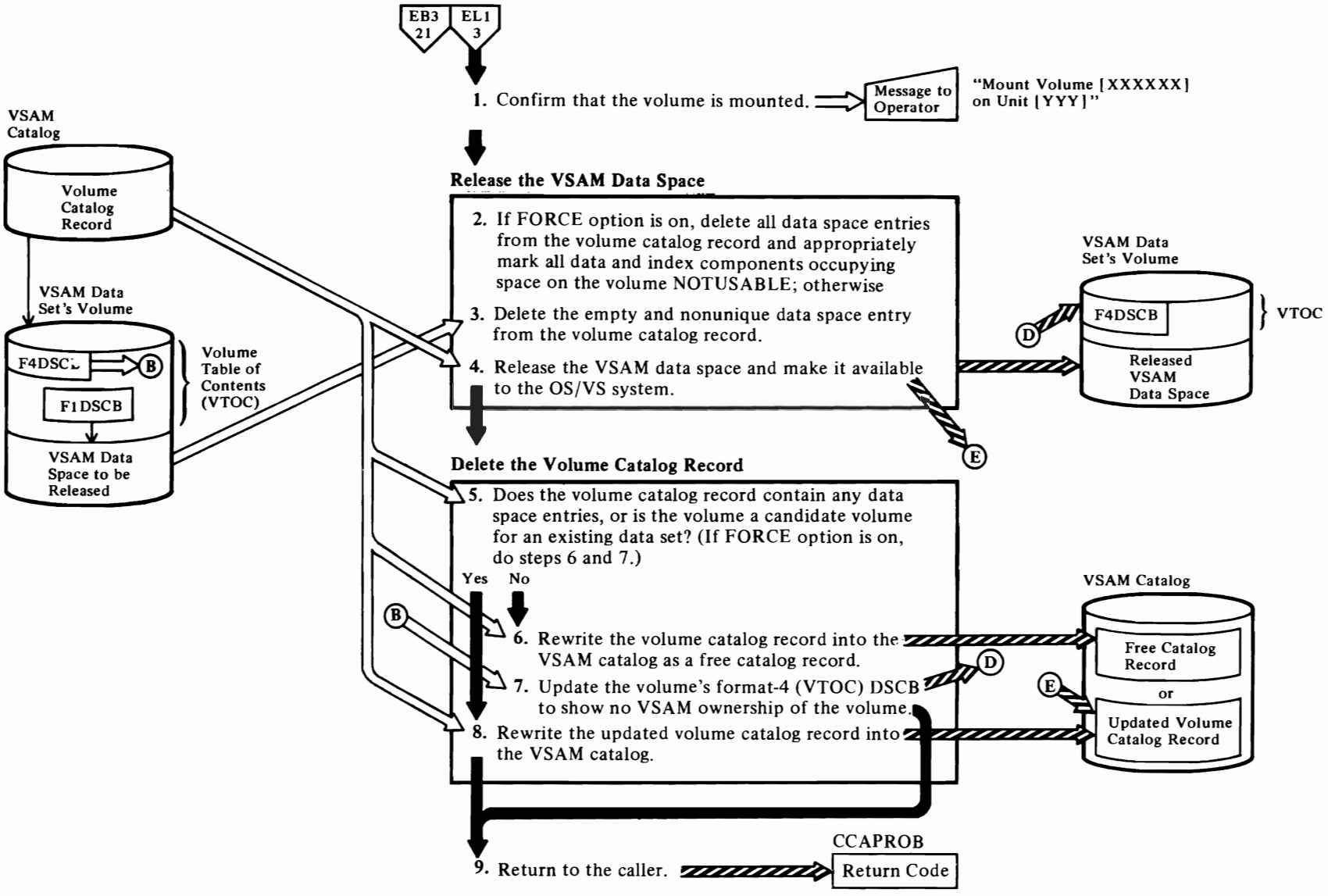
"Data Areas."

Volume catalog record description
Data set directory set of fields description and format

"Diagnostic Aids."

Catalog management return codes

# Diagram EK1. DELETE SPACE: Release All of the Empty VSAM Data Spaces on a Volume



## Notes for Diagram EK1

The DELETE SPACE command enables the user to release all VSAM data spaces on a specified volume. This figure describes the processes performed by the catalog management services DELETE SPACE routine when the user issues the Access Method Services DELETE SPACE command in the form:

```
DELETE SPACE
(entryname/password)
[CATALOG (catname/password)]
[FILE(dname)]
[FORCE]
```

where:

- *entryname* is the volume serial number of a direct access volume containing VSAM data spaces to be deleted.

  *catname* is the name of the catalog that contains the volume's catalog record.

- *password* is the catalog's master, control interval, or update password.

- *FILE* identifies the JCL statement that causes the volume to be mounted.

The DELETE command parameters are described in *OS/VS2 SVS Independent Component: Access Method Services*.

**1 IGG0CLBL: IGGPDELS** and **IGGPDLVM**

If the volume isn't already mounted and available for use, the DELETE SPACE routine issues the appropriate mount message to the operator.

**2 IGG0CLAI: IGGPFDSP and IGGPDFMI**

FORCE DELETE uses DADSM SCRATCH to release all VSAM data space and make it available to other OS/VS system users.

**3 IGG0CLBL: IGGPDLSH, IGGPDLSD,** and **IGGPDLCB**

The volume catalog record contains a data space group set of fields to describe each VSAM data space on the volume.

See "Data Areas" for details about the volume catalog record and its data space group sets of fields.

**4 IGG0CLBL: IGGPDLSC**

The OS/VS DADSM Delete routine releases the empty nonunique VSAM data space and makes its space available to other OS/VS system users.

See *OS/VS2 DADSM Logic* for details about deleting an OS/VS data set (to DADSM, the same as a VSAM data space).

**5 IGG0CLBL: IGGPDELS**

When the volume is totally empty, the volume catalog record can be deleted from the catalog. This occurs when there are no data space group sets of fields and no data set directory entry sets of fields in the volume catalog record.

See "Data Areas" for details about the volume catalog record.

**6 IGG0CLBL: IGGPDLET**

**7 IGG0CLBL: IGGPDLET**

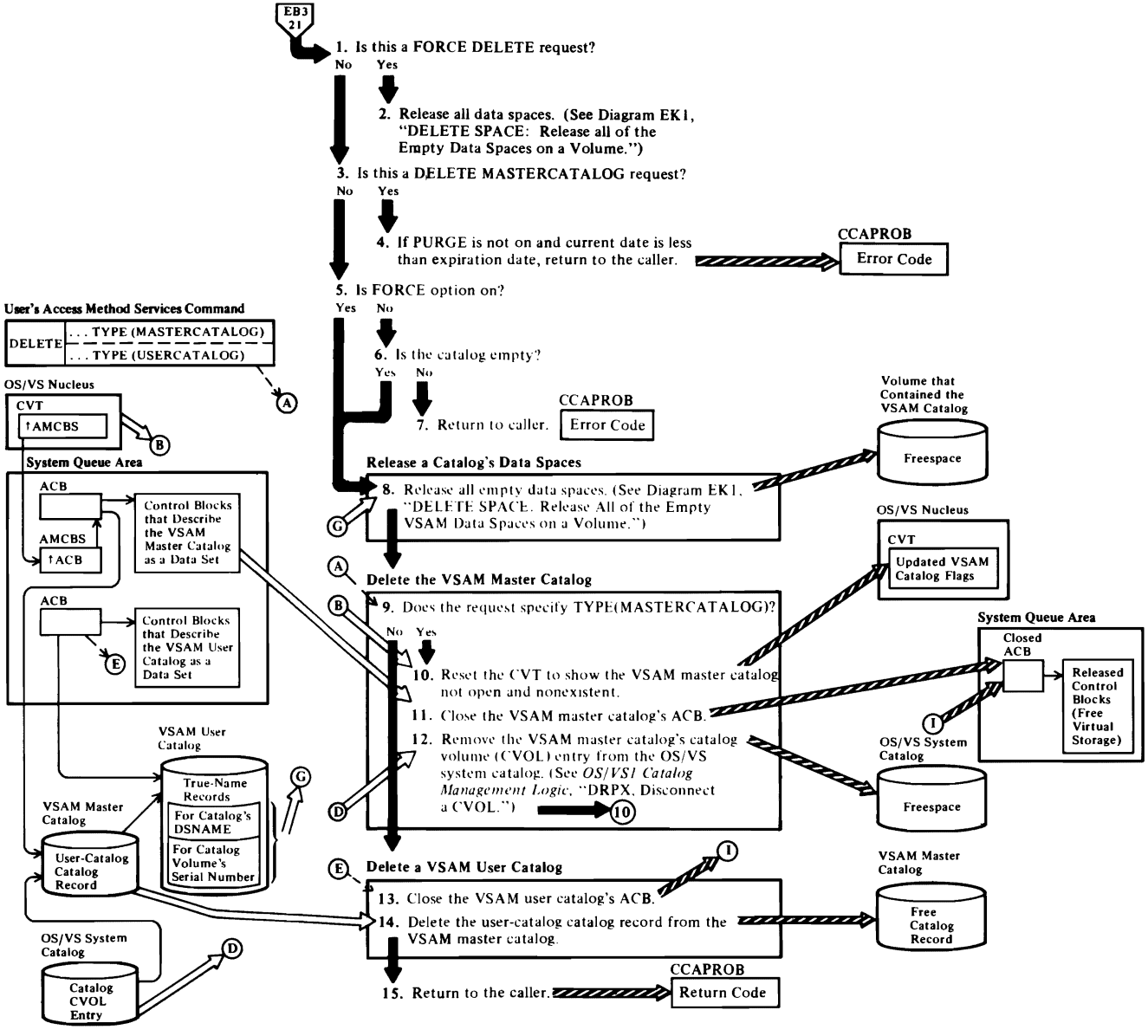
The format-4 (VTOC) DSCB is the first entry in a direct-access volume's VTOC. It contains the volume's owner's identification and information on how the volume is used.

See *OS/VS2 Data Areas* for DSCB details.

**8 IGG0CLBL: IGGPDLET**

See "Diagnostic Aids" for details about catalog management return codes and error codes.

# Diagram EL1. DELETE CATALOG: Release a VSAM Catalog



## Notes for Diagram EL1

The DELETE USERCATALOG and DELETE MASTERCATALOG commands enable the user to release a catalog's space and make it available to other OS/VS system users. The catalog must be empty (see step 1 notes) or the request is rejected. This figure describes the processes performed by the catalog management services DELETE CATALOG routines when the user issues the DELETE command in the following form:

```
DELETE
(entryname/password)
[USERCATALOG | MASTERCATALOG]
[FORCE]
```

where:

- *entryname* is the dsname of the catalog to be deleted.
- *password* is the user catalog's master password (allows user catalog deletions), or the VSAM master catalog's master password (allows master catalog deletions).
- *[USERCATALOG | MASTERCATALOG]* identifies the type of catalog being deleted.

The DELETE command parameters are described in *OS/VS2 SVS Independent Component: Access Method Services*.

**6 IGG0CLAF: IGGPDELC**

If the catalog contains more than two true name catalog records, it is not empty and cannot be deleted, unless the FORCE option is on.

See "Data Areas" for details about catalog organization and the true name catalog record.

**8 IGG0CLAF: IGGPSDSP**

The volume catalog record contains an entry for each VSAM data space allocated on the volume. Each entry contains the data necessary to free the data space.

See "Data Areas" for details about the volume catalog record.

Diagram EK1, DELETE SPACE, shows how each VSAM data space is released and it's space made available to other OS/VS system users.

**10 IGG0CLAF: IGGPDELC**

The communications vector table (CVT) points to the AMCBS which points to the control blocks which describe the VSAM master catalog to the OS/VS system.

See *OS/VS2 Supervisor Logic* for details about the CVT.

**11 IGG0CLAF**

See "Data Areas" for details about the ACB and for details of the control blocks that allow catalog records to be processed.

**12 IGG0CLAF: IGGPDELC**

The OS/VS system catalog contains a data set entry for each OS/VS private catalog and for the VSAM master catalog, but not for VSAM user catalogs.

See *OS/VS Catalog Management Logic* for OS/VS system catalog and data set entry information.

**13 IGG0CLAF: IGGPDELC**

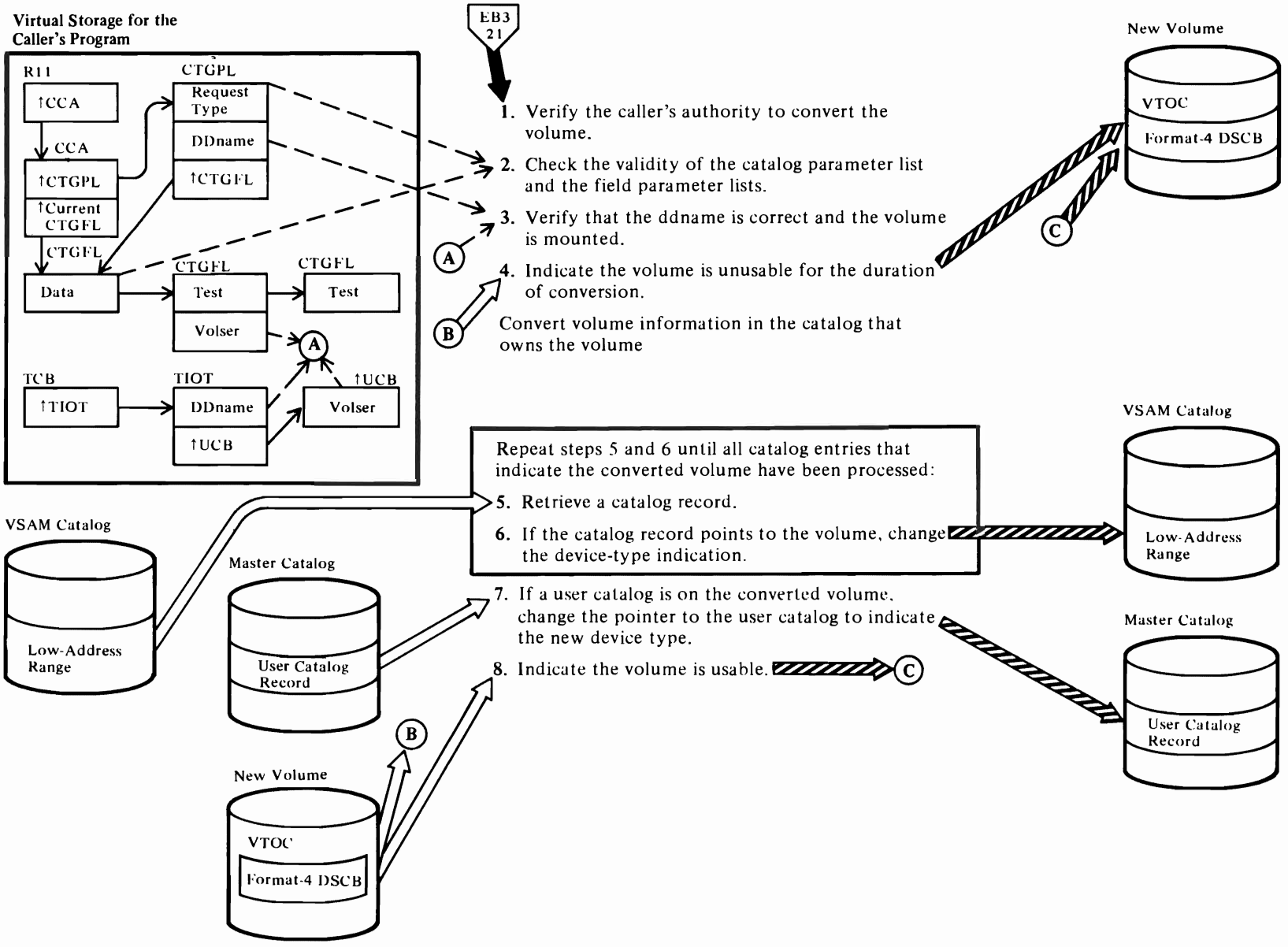
The VSAM user catalog is described, as a data set, by its ACB. See Diagram AD1, VSAM Close, for a description of closing a VSAM data set.

See "Data Areas" for details about the ACB.

**15**

See "Diagnostic Aids" for details about catalog management return codes and error codes.

# Diagram EM1. CONVERTV: Convert a Volume to or from Mass Storage



## Notes for Diagram EM1

The CONVERTV command enables the user to convert the contents of a direct-access storage volume to a mass storage volume, or *vice versa,* and to have catalog information that indicates the old device type changed to indicate the new device type.

This figure describes the processing performed by the catalog management services routines when the user issues the Access Method Services CONVERTV command in the form:

```
CONVERTV
FROMFILE(ddname)
TOFILE(ddname)
[RECATALOG(ALL | VSAMCATALOG)]
[CATALOG(catname[/password])]
```

where:

- *FROMFILE* and *TOFILE* identify the DD statements that cause allocation of the devices from and to which the volume's contents are being converted.
- *RECATALOG* indicates that a VSAM catalog owns the volume being converted and indicates the extent of recataloging that is to be done.
- *CATALOG* identifies the user catalog, if there is one, on the volume being converted.

CONVERTV is described in *OS/VS Mass Storage System (MSS) Services for Space Management.*

**1 IGG0CLBZ: IGGPCONV**

Issues the TESTAUTH macro, which checks whether the calling progran is an APF (authorized program function).

**2 IGG0CLBZ: IGGPVALI**

**3 IGG0CLBZ: IGGPCONV**

**4 IGG0CLBU: IGGPF4RD**

Reads the format-4 DSCB.

**IGG0CLBU: IGGPF4WR**

Writes the format-4 DSCB. If CONVERTV fails and a user subsequently attempts to open a data set on the volume, he will receive an error code from Open.

**IGG0CLAG: IGGPPUPC**

Writes the modified volume record. The identity of this record is passed to IGG0CLBZ from Access Method Services.

**IGG0LAG: IGGPRCCR**

Updates the catalog control record in the catalog that owns the volume to indicate the next free control interval.

**5 IGG0CLBZ: IGGPGALO**

**6 IGG0CLAV: IGGPMOD**

Modifies device-type fields in catalog records that point to the volume being converted.

**7 IGG0CLAH: IGGPSCAT**

Searches the master catalog for the user catalog record.

**IGG0CLAV: IGGPMOD**

Writes the modified master catalog record that points to the user catalog. In step 5, IGG0CLBZ discovers that a user catalog is on the volume being converted when it finds a user catalog record that indicates the volume's volume serial number.

**IGG0CLA3: IGGPRPLF**

Releases the master catalog from exclusive control.

**8** See step 4.

# PROGRAM ORGANIZATION

VSAM program listings are the key to VSAM's organization. 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 can 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

A compendium and its notes describe the flow of control between procedures and modules to perform a function. The compendium is a supplement to the function's method of operation diagram.

The compendium's notes describe how each procedure and module contributes to the completion of the function, and under what circumstances the procedure or module is called.

### *Reading Module Flow Compendiums*

Module flow 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 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.

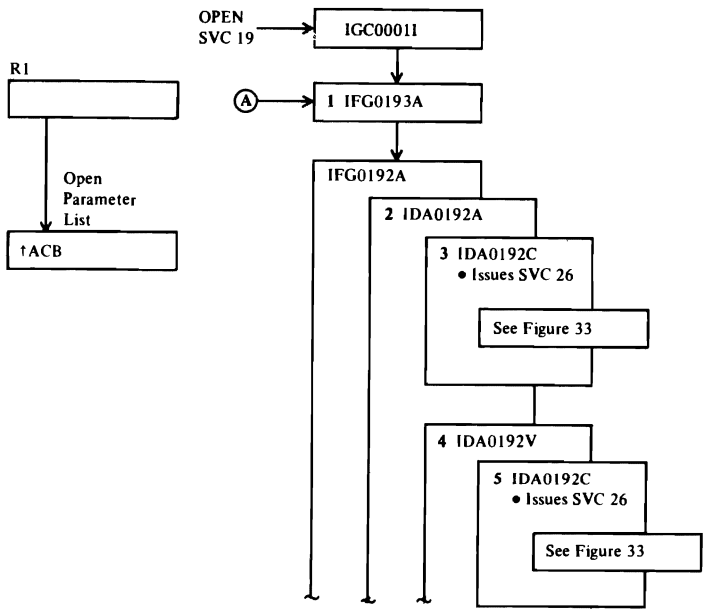



Figure 5. Program Organization Compendium Figure

Blocks that are indented (otherwise contained within another block) are called to perform a specified function and return, when finished, to the caller. 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. Figure 6 shows the descriptive notes to Figure 5.

**Notes for Figure 9**

1 IGC0001I, IFG0193A, and IFG0198N are OS/VS Open modules (see *OS/VS' Open/Close/EOV Logic* for details).

A IFG0191Y (in Figure 10) XCTLs to IFG0193A to open a VSAM catalog. Open-processing and return-to-the-caller continues as shown in this figure.

2 IDA0192A is the VSAM Open module.

3 IDA0192C calls VSAM Catalog Management (LOCATE) to retrieve information about the VSAM-object-being-opened from its VSAM catalog record.

4 IDA0192V ensures that the required minimum number of the object's direct-access volumes are mounted.

5 IDA0192C calls VSAM Catalog Management (LOCATE) to retrieve volume serial numbers from the object's VSAM catalog record.

6 IDA0192Z builds the following VSAM control blocks

| | | |
|---|---|---|
| AMB | Buffers | FDB |
| AMDSB | DEBs | IOS |
| ARDB | Dummy DEB | PLH |
| BUFC | DIWA | |

7 IDA0192W builds the CPA control block

8 IDA0192S writes SMF record type 62.

9 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.

Figure 6. Notes to Program Organization Compendium Figure

Catalog management procedures call certain procedures so frequently that, if each call were shown, the catalog management compendiums would be cluttered. For this reason, whenever a procedure calls one of these (frequently called) procedures, its module identifier (last two letters of the module name: IGG0CLxx) is listed instead of drawing a separate block to show the procedure call. The frequently-called modules are:

- AG: IGG0CLAG—Catalog Management Input/Output Procedures
- AV: IGG0CLAV: IGGPMOD—Modify Catalog Field(s)
- AZ: IGG0CLAZ: IGGPEXT—Extract Catalog Field(s)
- BI: IGG0CLBI—Catalog Management Input/Output Procedures
- BV: IGG0CLBV—SMF Record Processing Procedures
- B3: IGG0CLB3—SMF Record Processing Procedures

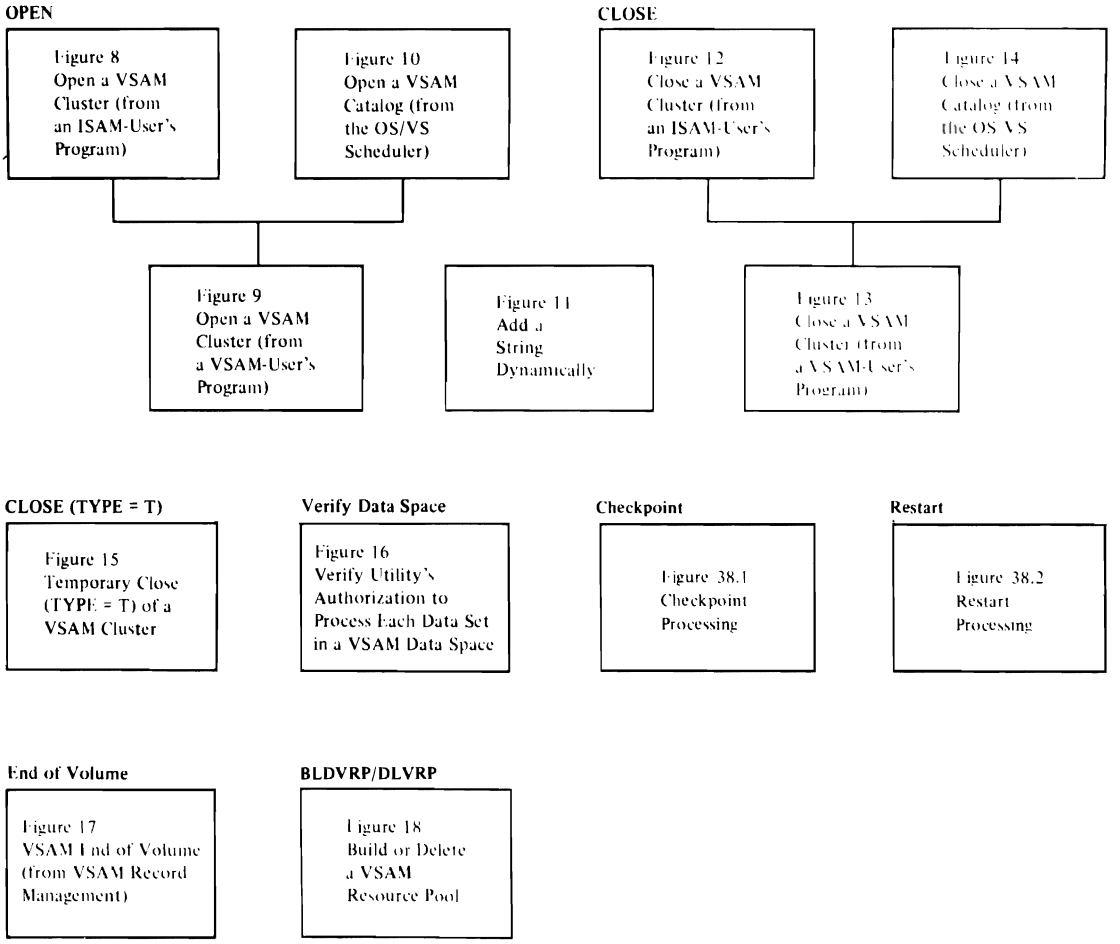



**Figure 7. Open/Close/End-of-Volume Program Organization Contents**

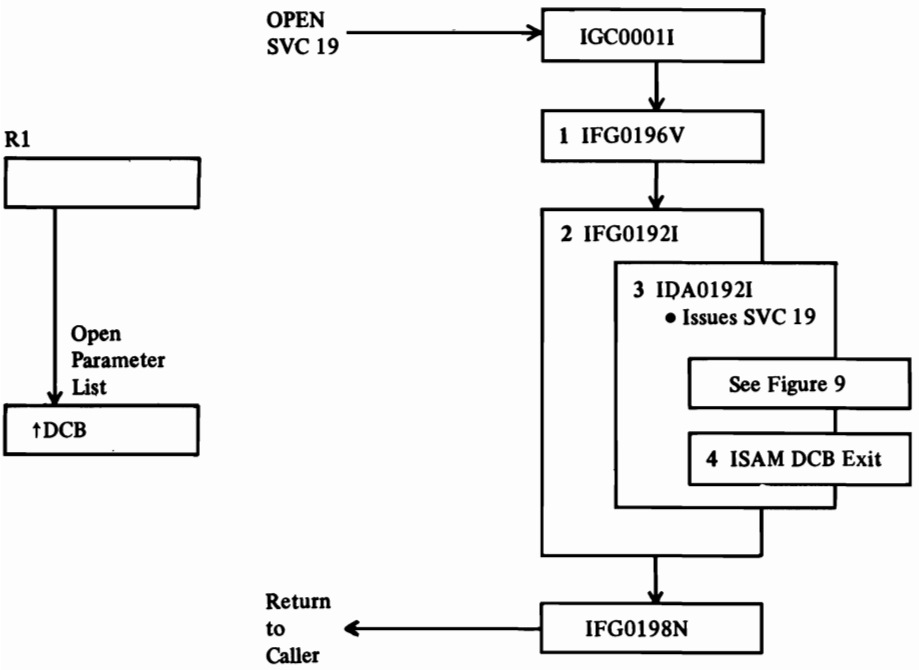



**Figure 8. Open a VSAM Cluster (From an ISAM-User's Program)**

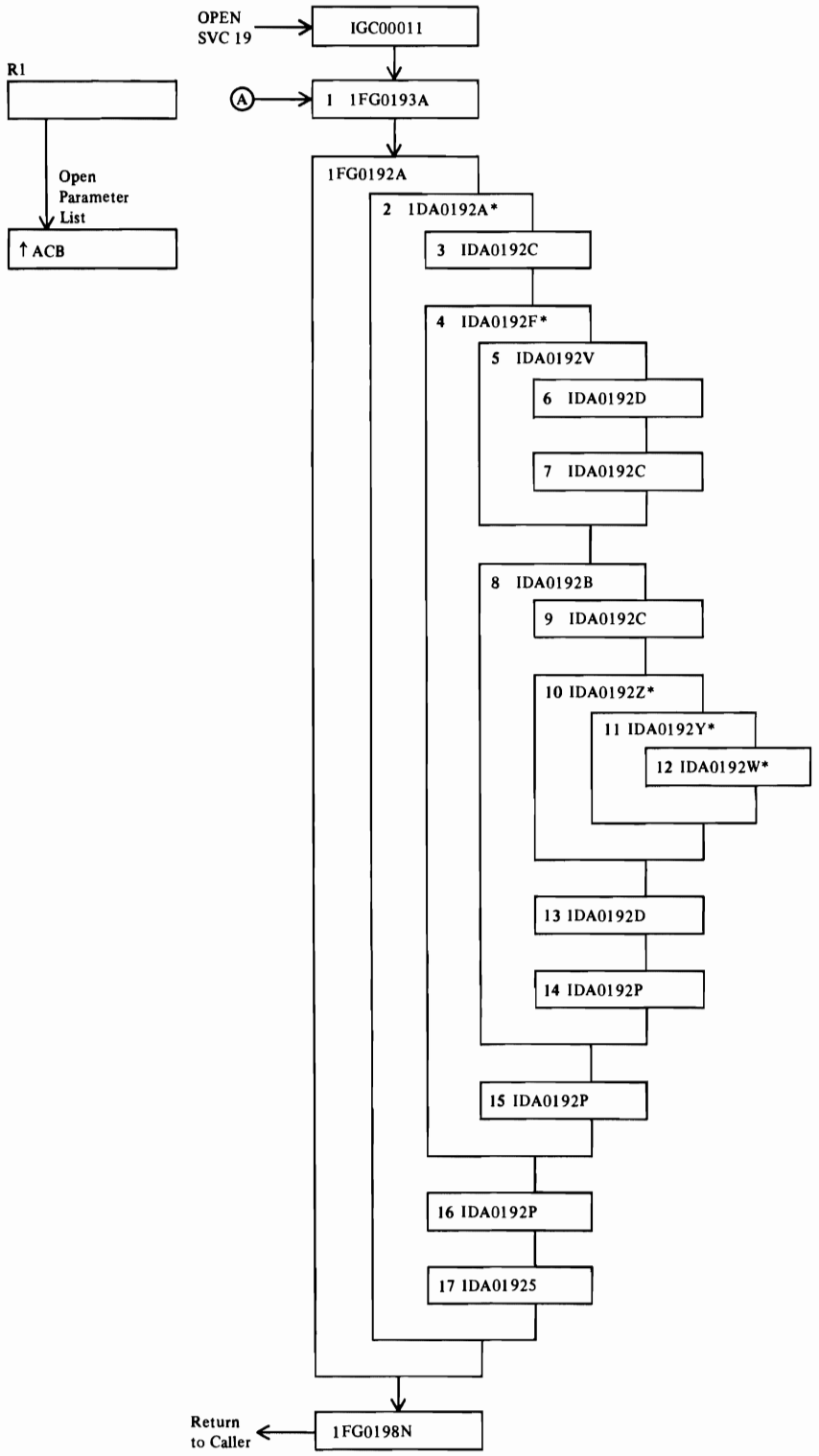
**Notes for Figure 8**

1 IGC0001I, IFG0196V, and IFG0198N are OS/VS Open modules (see *OS/VS Open/Close/EOV Logic* for details).

2 IFG0192I is an alias for IFG0192A.

3 IDA0192I is the ISAM Interface Open module. IDA0192I is an alias for IDA0192A. IDA0192I issues the VSAM OPEN macro instruction (SVC 19).

4 IDA0192I calls the ISAM DCB exit routine if the user's program has specified one.




NOTE: * indicates that the module calls IDA0192M for virtual storage.
IDA0192M is the VSAM Virtual-Storage Manager. It builds the HEB.

**Figure 9. Open a VSAM Cluster (From a VSAM-User's Program)**

**Notes for Figure 9**

**1** IGC0001I, IFG0193A, and IFG0198N are OS/VS Open modules (see *OS/VS Open/Close/EOV Logic* for details).

**A** IFG0191Y (in Figure 10) XCTLs to IFG0193A to open a VSAM catalog. Open-processing and return-to-the-caller continues as shown in this figure.

**2** IDA0192A is the VSAM Open module. It builds the following VSAM control blocks: BIB, WSHD, Dummy DEB.

**3** IDA0192C calls VSAM Catalog Management (LOCATE) to retrieve information about the VSAM object being opened from its VSAM catalog record.

**4** IDA0192F opens base, path, and upgrade clusters. It builds the following VSAM control blocks:

| | | |
|---|---|---|
| ACB | CMB | VAT |
| AMBL | UPT | VMT |

**5** IDA0192V ensures that the required minimum number of the object's direct-access volumes are mounted.

**6** IDA0192D stages (via ACQUIRE) data from mass storage to a direct-access storage device (staging drive).

**7** IDA0192C checks the time stamp.

**8** IDA0192B opens VSAM clusters.

**9** IDA0192C calls VSAM Catalog Management (LOCATE) to retrieve volume serial numbers from the object's VSAM catalog record.

**10** IDA0192Z builds the following control blocks:

| | | |
|---|---|---|
| AMB | DEB | IWA |
| AMDSB | EDB | LPMB |
| ARDB | IRB | |

**11** IDA0192Y builds the:

| | | |
|---|---|---|
| BUFC | IOB | RPL |
| Buffers | PLH | WAX |

**12** IDA0192W builds the CPA control block.

**13** IDA0192D stages (via a Mass Storage System ACQUIRE) data from mass storage to a direct-access storage device (staging drive).

**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 **14**.

**16** Same as **14**.

**17** IDA0192S writes SMF record type 62.

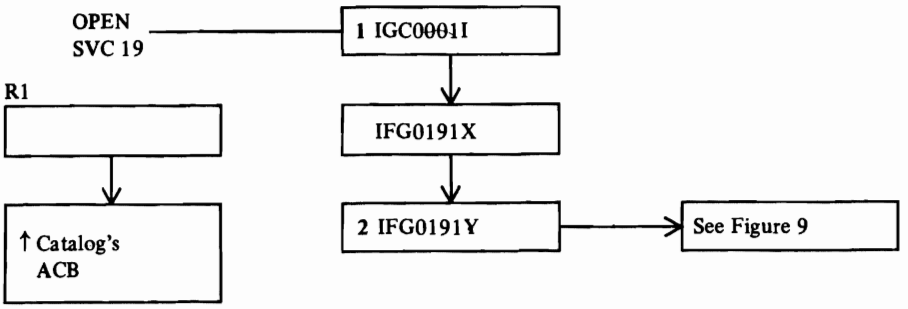



Figure 10. Open a VSAM Catalog (From the OS/VS Scheduler)

**Notes for Figure 10**

1 IGC0001I is an OS/VS Open module.

2 IFG0191X and IFG0191Y are VSAM Catalog Open: ACB Processing modules. These modules perform special processing for the catalog's ACB, then transfer control (using the XCTL macro instruction) to IFG0193A (in Figure 9).

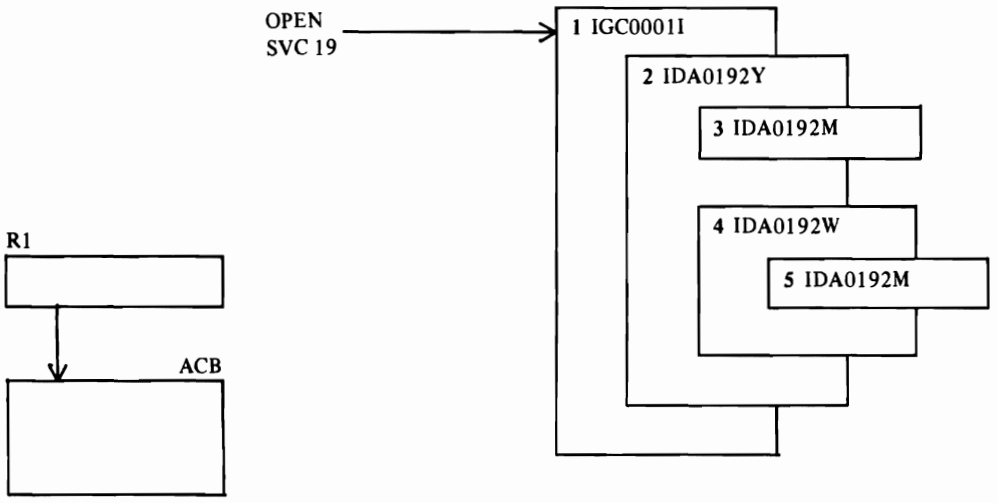



Figure 11. Add a String Dynamically

**Notes for Figure 11**

1 IGC001I determines whether OPEN is for a VSAM ACB.

2 IDA0192Y builds control blocks necessary for Record Management to complete I/O requests: PLH, IOB, PFL, BUFC, buffer.

3 IDA0192M allocates virtual storage for the requester to use to build control blocks. IDA0192M builds the HEB.

4 IDA0192W builds a channel program area for the added string.

5 See note for step 3.

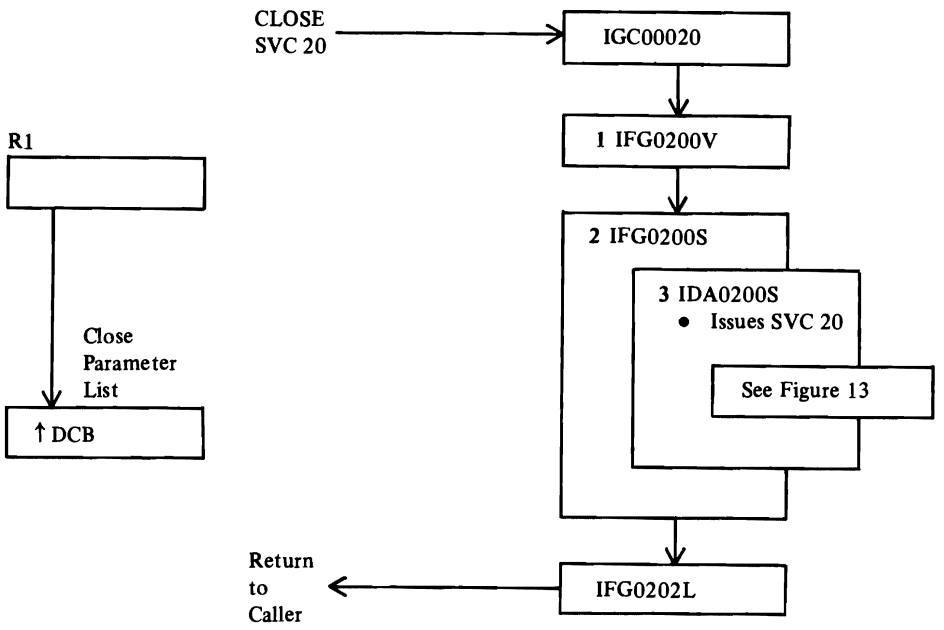



Figure 12. Close a VSAM Cluster (From an ISAM-User's Program)

**Notes for Figure 12**

1 IGC00020, IFG0200V, and IFG0202L are OS/VS Close modules (see *OS/VS' Open/Close/EOV Logic* for details).

2 IFG0200S is an alias for IFG0192A.

3 IDA0200S is the ISAM Interface Close module. IDA0200S is an alias for IDA0192A.

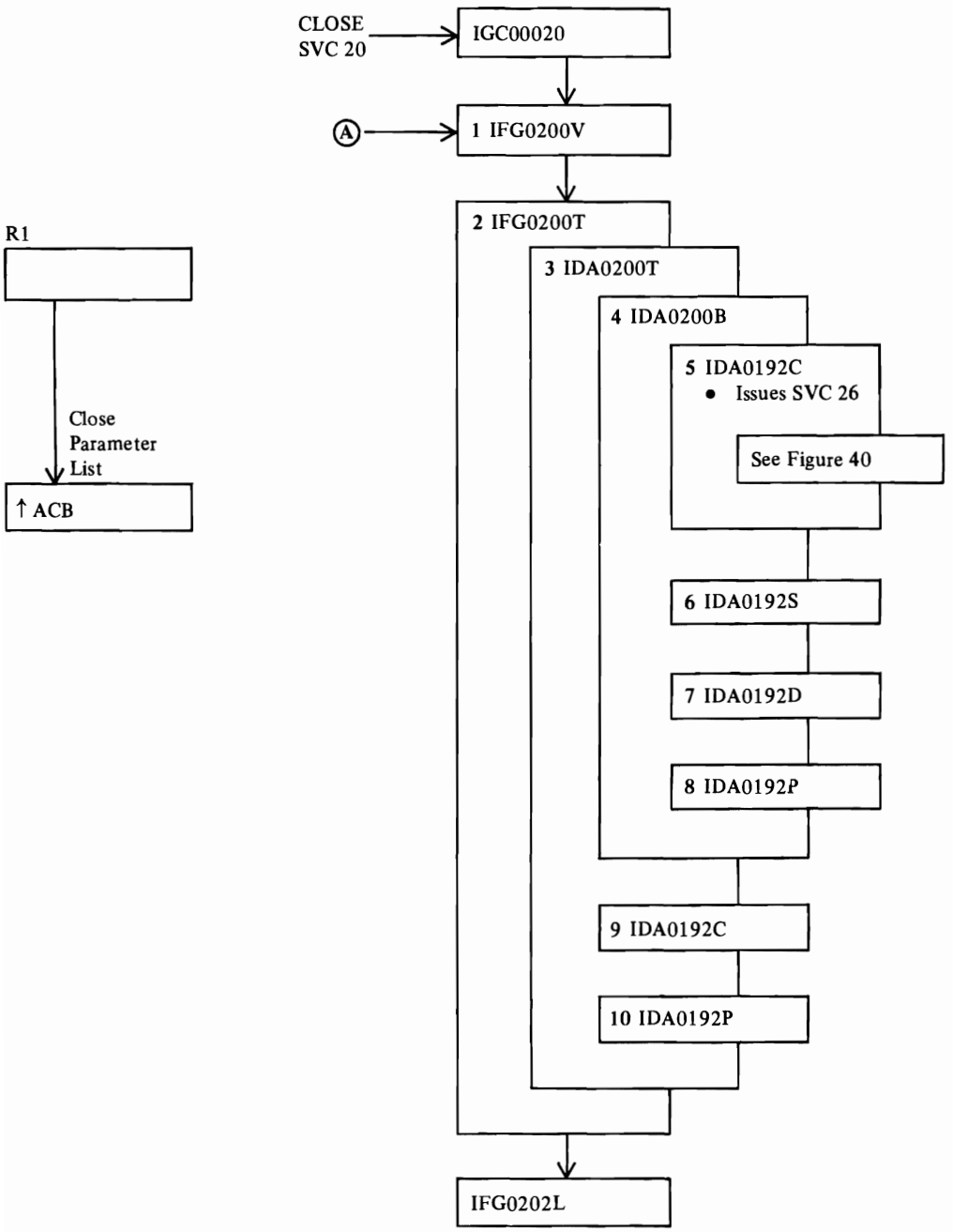



Figure 13. Close a VSAM Cluster (From a VSAM-User's Program)

**Notes for Figure 13**

**1** IGC00020, IFG0200V, and IFG0202L are OS/VS Close modules (see *OS/VS Open/Close/EOV Logic* for details).

**A** IFG0200N (in Figure 14) XCTLs to IFG0200V to close a VSAM catalog or catalog recovery area. Close-processing and return-to-the-caller continues as shown in this figure.

**2** IFG0200T is an alias for IFG0192A.

**3** IDA0200T is the VSAM Close module.

**4** IDA0200B closes VSAM clusters.

**5** IDA0192C calls VSAM Catalog Management (UPDATE) to modify statistical information in the object's VSAM catalog record.

**6** IDA0192S writes SMF record(s) type 64.

**7** IDA0192D destages (via a Mass Storage System RELINQUISH) the data from direct-access storage to mass storage.

**8** Whenever IDA0200B detects an error, IDA0192P issues a diagnostic message.

**9** When a catalog is being closed, IDA0192C calls VSAM Catalog Management (LOCATE) to indicate that Close has finished processing.

**10** IDA0192P issues a diagnostic message whenever IDA0200T detects an error.

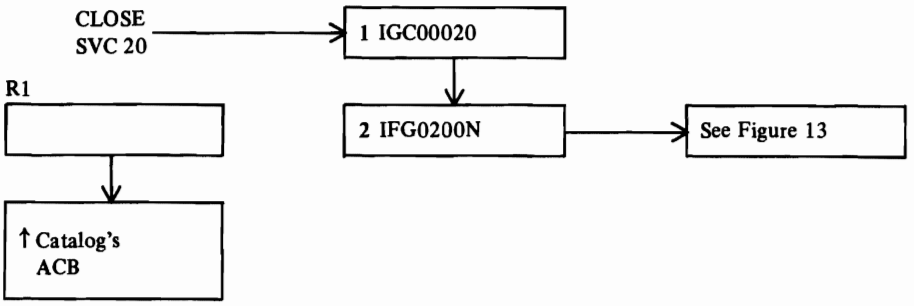



Figure 14. Close a VSAM Catalog (From the OS/VS Scheduler)

**Notes for Figure 14**

1 IGC00020 is an OS/VS Close module (see *OS/VS Open/Close/EOV Logic* for details).

2 IFG0200N is the VSAM Catalog Close: ACB Processing module. It performs special processing for the catalog's ACB, then XCTLs to IFG0200V (in Figure 13).

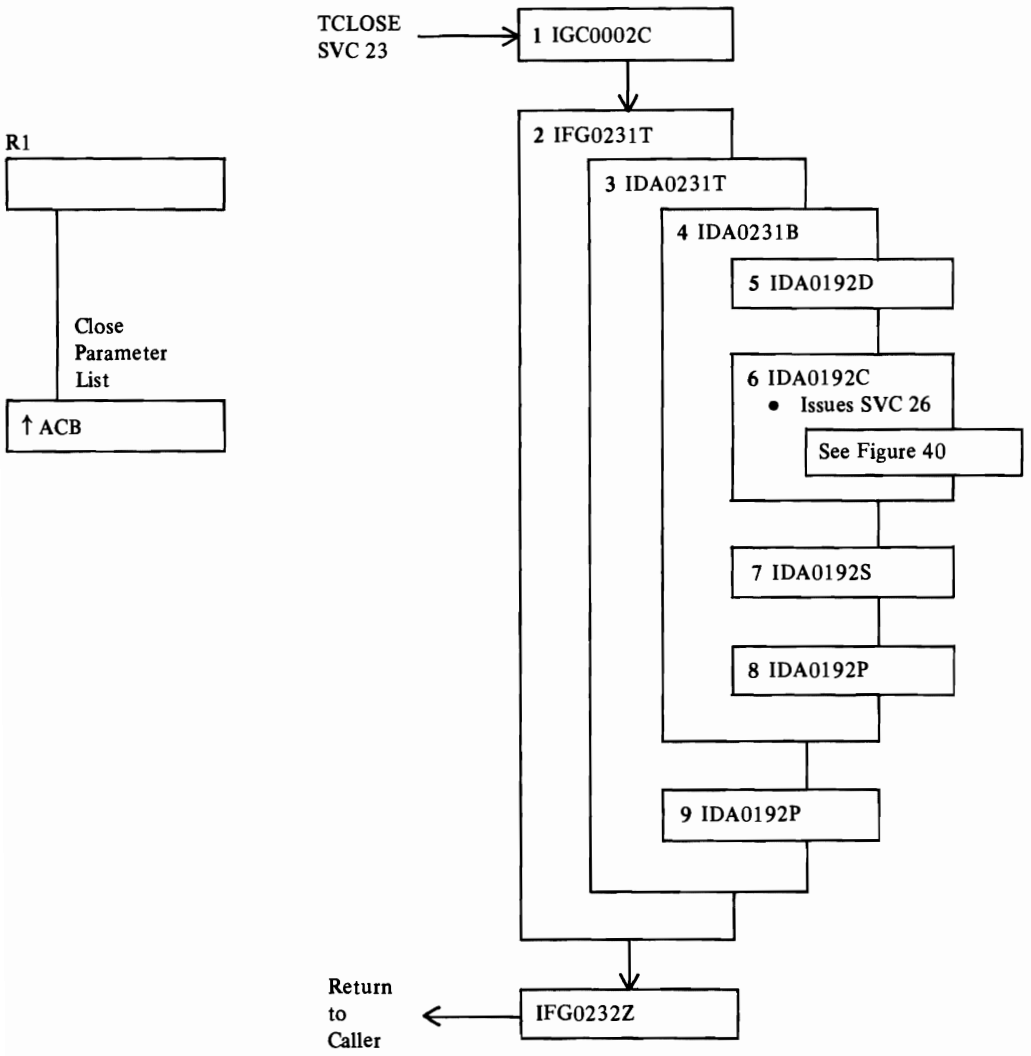



Figure 15. Temporary Close (TYPE=T) of a VSAM Cluster

**Notes for Figure 15**

1 IGC0002C and IFG0232Z are OS/VS CLOSE (TYPE=T) modules (see *OS/VS Open/Close/EOV Logic* for details).

2 IFG0231T is an alias for IFG0192A.

3 IDA0231T is the VSAM CLOSE (TYPE=T) module.

4 IDA0231B is the VSAM CLOSE (TYPE=T) module for closing clusters.

5 IDA0192D destages (via a Mass Storage System RELINQUISH) the data from direct-access storage to mass storage. If the data was not bound in direct-access storage, IDA0192D restages (via a Mass Storage System ACQUIRE) the data from mass storage to direct-access storage.

6 IDA0192C calls VSAM Catalog Management (UPDATE) to modify statistical information in the object's VSAM catalog record.

7 IDA0192S writes SMF record(s) type 64.

8 Whenever IDA0231B detects an error, IDA0192P issues a diagnostic message.

9 IDA0192P issues a diagnostic message whenever IDA0231T detects an error.

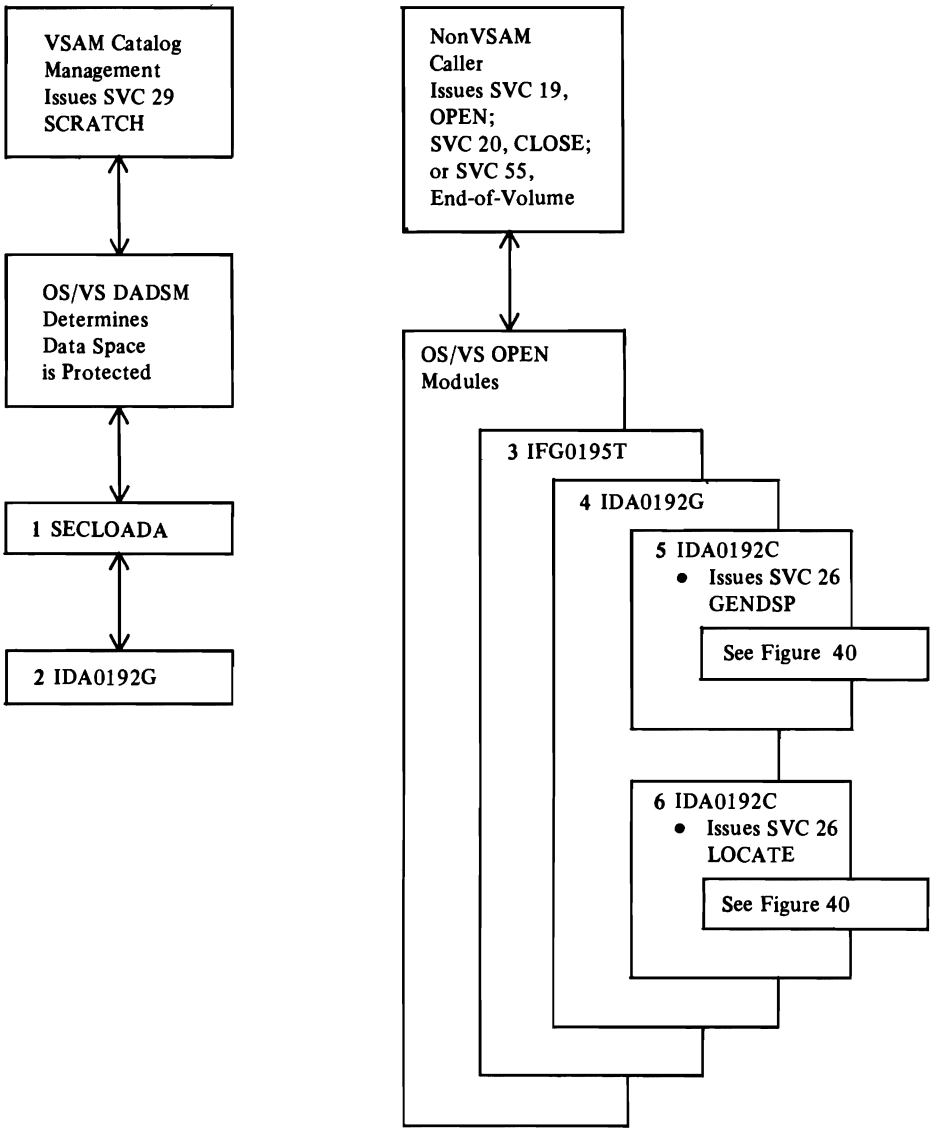



Figure 16. Verify a NonVSAM Caller's Authorization to Process Each Data Set in a VSAM Data Space

**Notes for Figure 16**

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 OS/VS DADSM Scratch Routine and the format-1 DSCB identifies a VSAM data space, SECLOADA passes control to IDA0192G. (See *OS/VS 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 EOV, IFG0195T determines that the caller is other than VSAM or ISAM-Interface and that the format-1 DSCB is protected. (See *OS/VS 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 VSAM (catalog management) to obtain the dsname of each VSAM data set in the data space.

6 IDA0192C issues SVC 26 (LOCATE) to catalog management to verify that the caller can supply each protected data set's master password.

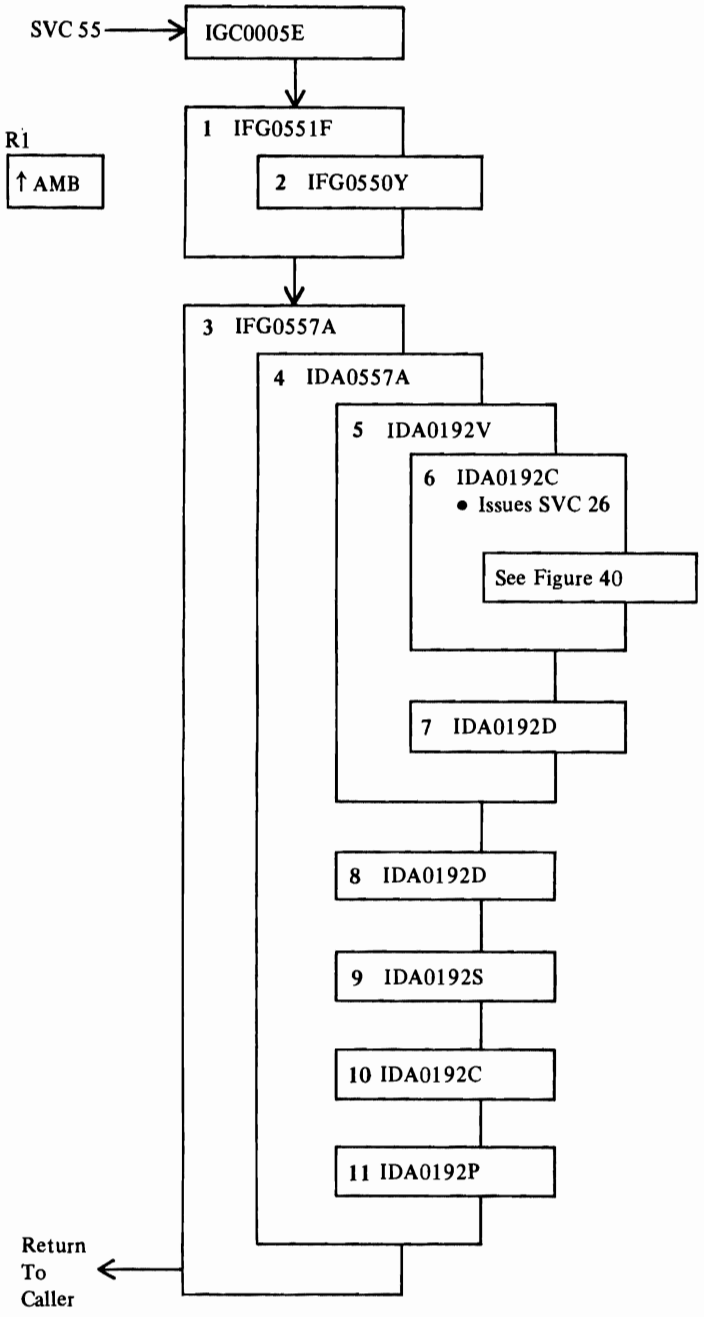



Figure 17. VSAM End of Volume (From VSAM Record Management: IDAEOVIF Procedure (in Module IDA019R5))

**Notes for Figure 17**

1 IGC0005E and IFG0551F are OS/VS End of Volume modules (see *OS/VS Open/Close/EOV Logic* for details).

2 IFG0550Y is an alias It performs special processing for the VSAM catalog's ACB and is called when EOV is issued against a VSAM catalog.

3 IFG0557A is an alias for IFG0192A.

4 IDA0557A is the VSAM End of Volume module.

5 IDA0192V ensures that the required volumes are mounted for the VSAM object.

6 IDA0192C calls VSAM Catalog Management (UPDATE) to modify information in the object's VSAM catalog record.

7 IDA0192D stages (via a Mass Storage System ACQUIRE) new extents to a direct-access storage device (staging drive).

8 Same as step 7.

9 IDA0192S writes SMF record(s) type 64.

10 IDA0192C calls VSAM Catalog Management (LOCATE and UPDATE) to locate and update information in the object's VSAM catalog record.

11 Whenever a VSAM End of Volume module detects an error, IDA0192P issues a diagnostic message and traces VSAM control blocks if the Generalized Trace Facility (GTF) is active.

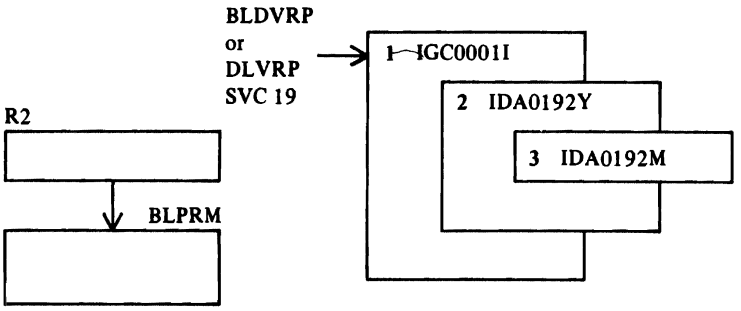



Figure 18. Build or Delete a VSAM Resource Pool

**Notes for Figure 18**

1 IGC0001I determines whether SVC19 is for BLDVRP or DLVRP.

2 IDA0192Y builds or deletes control blocks for a VSAM resource pool: VSRT, WSHD, CPA header, PLHs, BSPH, BUFCs, buffers. For BLDVRP, it chains the VSRT to the VAT; for DLVRP, it unchains it.

3 IDA0192M allocates virtual storage for the requester to use to build control blocks. IDA0192M builds the HEB.

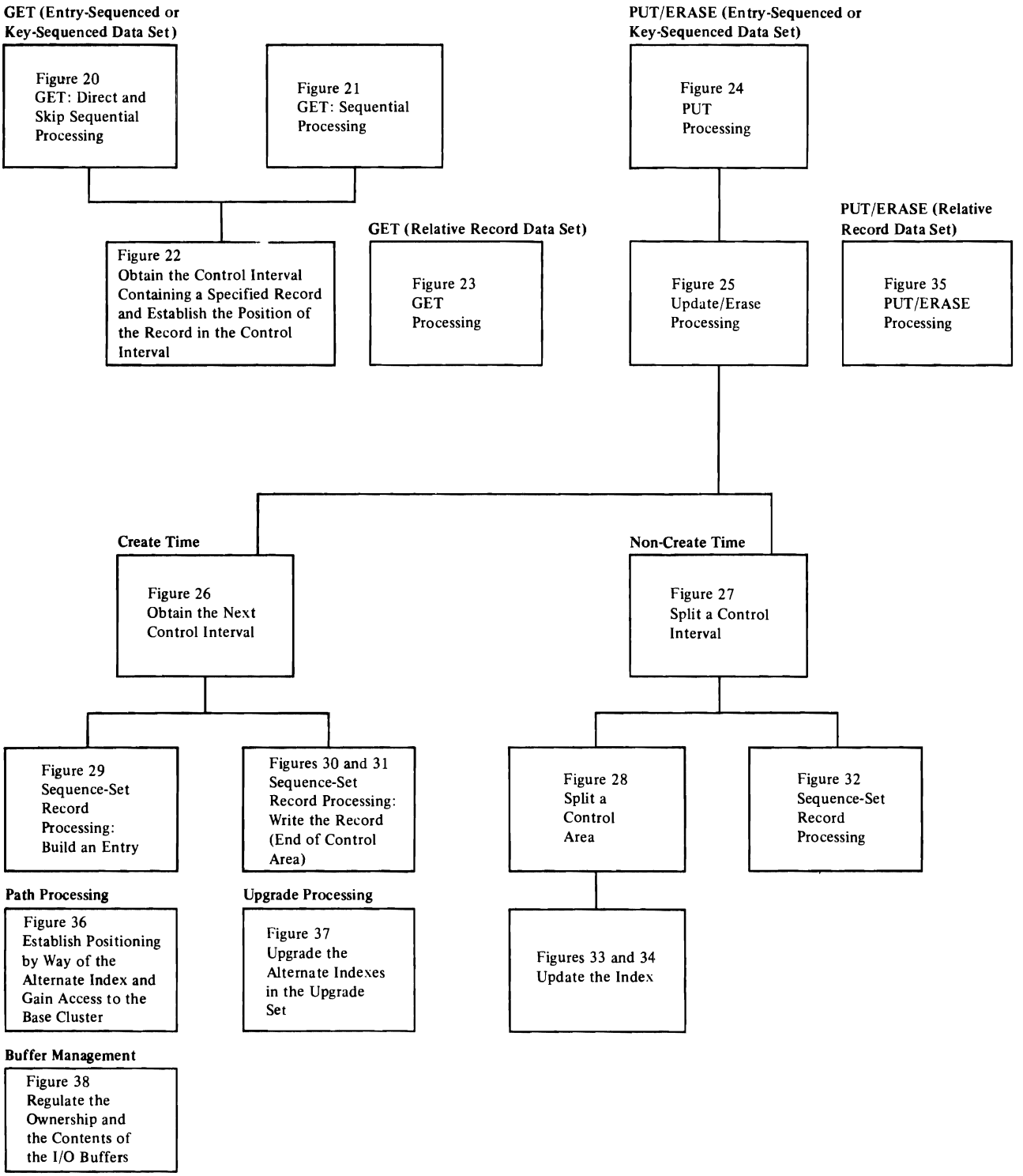



**Figure 19. Record Management Program Organization Contents**

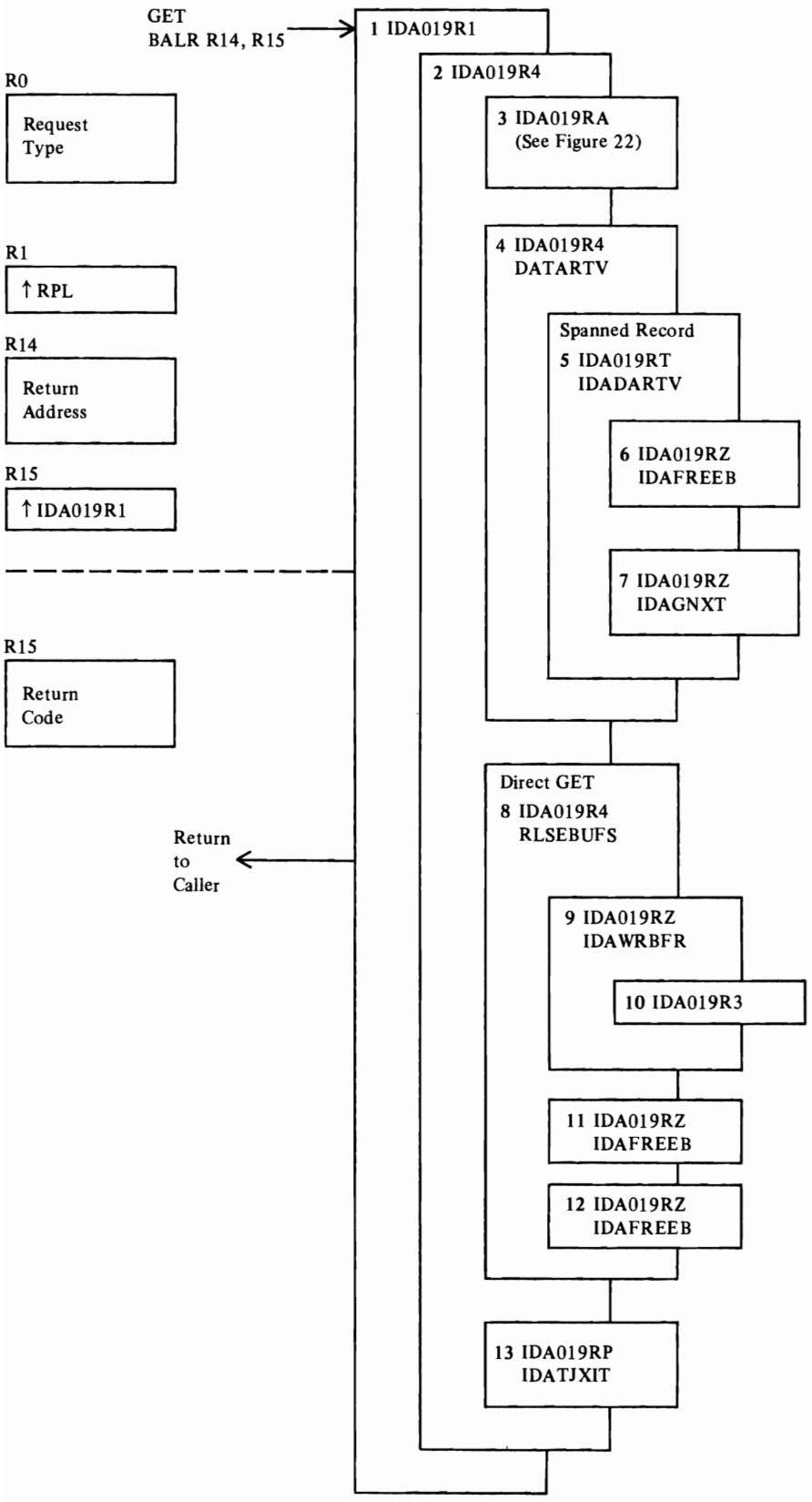



Figure 20. GET: Direct and Skip Sequential Processing (ESDS, KSDS)

**Notes for Figure 20**

1 IDA019R1 is the common Record Management request module. It verifies that the request is a valid Record Management macro instruction, 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 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 IDA019R3 builds the required I/O CCW chain and issues an EXCP macro instruction to rewrite the record on the direct access device.

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.

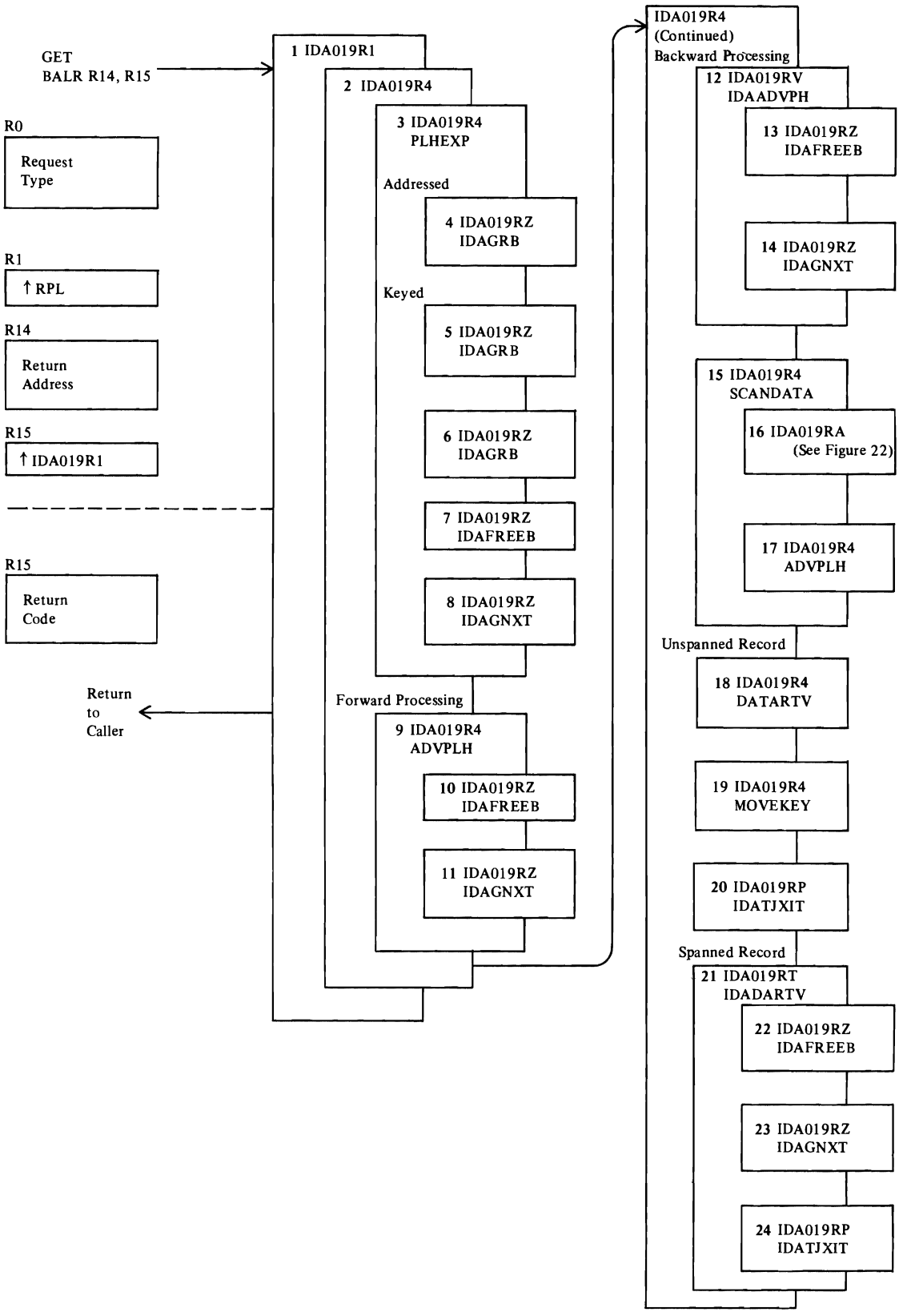



Figure 21. GET: Sequential Processing (ESDS, KSDS)

**Notes for Figure 21**

1 IDA019R1 is the common Record Management request module. It verifies that the request is a valid Record Management macro instruction, 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. IDA019R3 issues the I/O CCW chain required to read the control interval from the direct access device.

  5 If the request is keyed, IDAGRB reads in the first sequence-set control interval. IDA019R3 issues the I/O CCW chain required to read in the control interval from the direct access access device.

  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 so that it points 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 sequential control interval (by descending RBA), unless another buffer already contains the control interval. The PLH is set to point to the last data 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**.

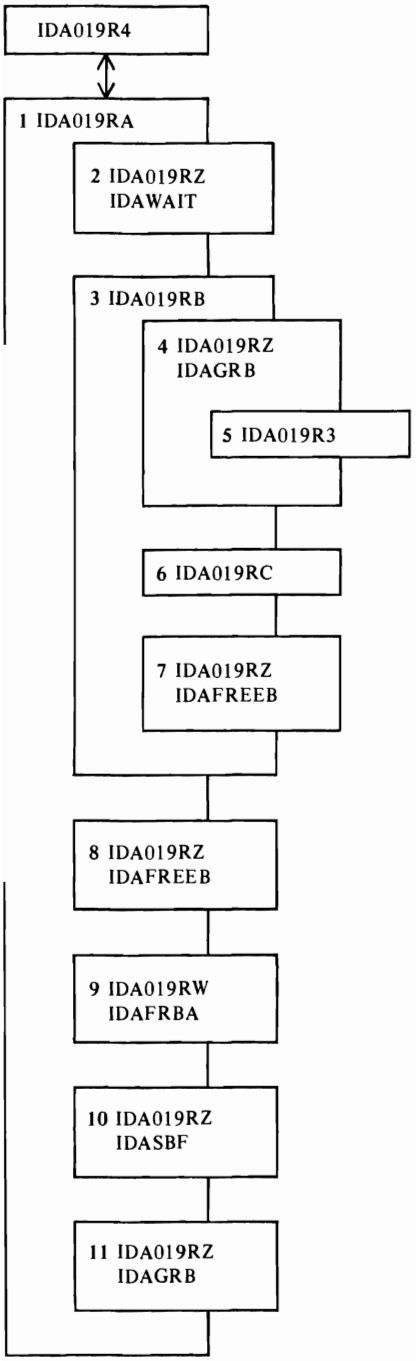



**Figure 22. Obtain the Control Interval Containing a Specified Record and Establish the Position of the Record in the Control Interval (ESDS, KSDS)**

**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.

3 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 IDA019R3 issues the I/O CCW chain required to read the control interval into a buffer, if another buffer doesn't already contain the control interval.

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. Steps **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 determines 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 is the data set is entry-sequenced.

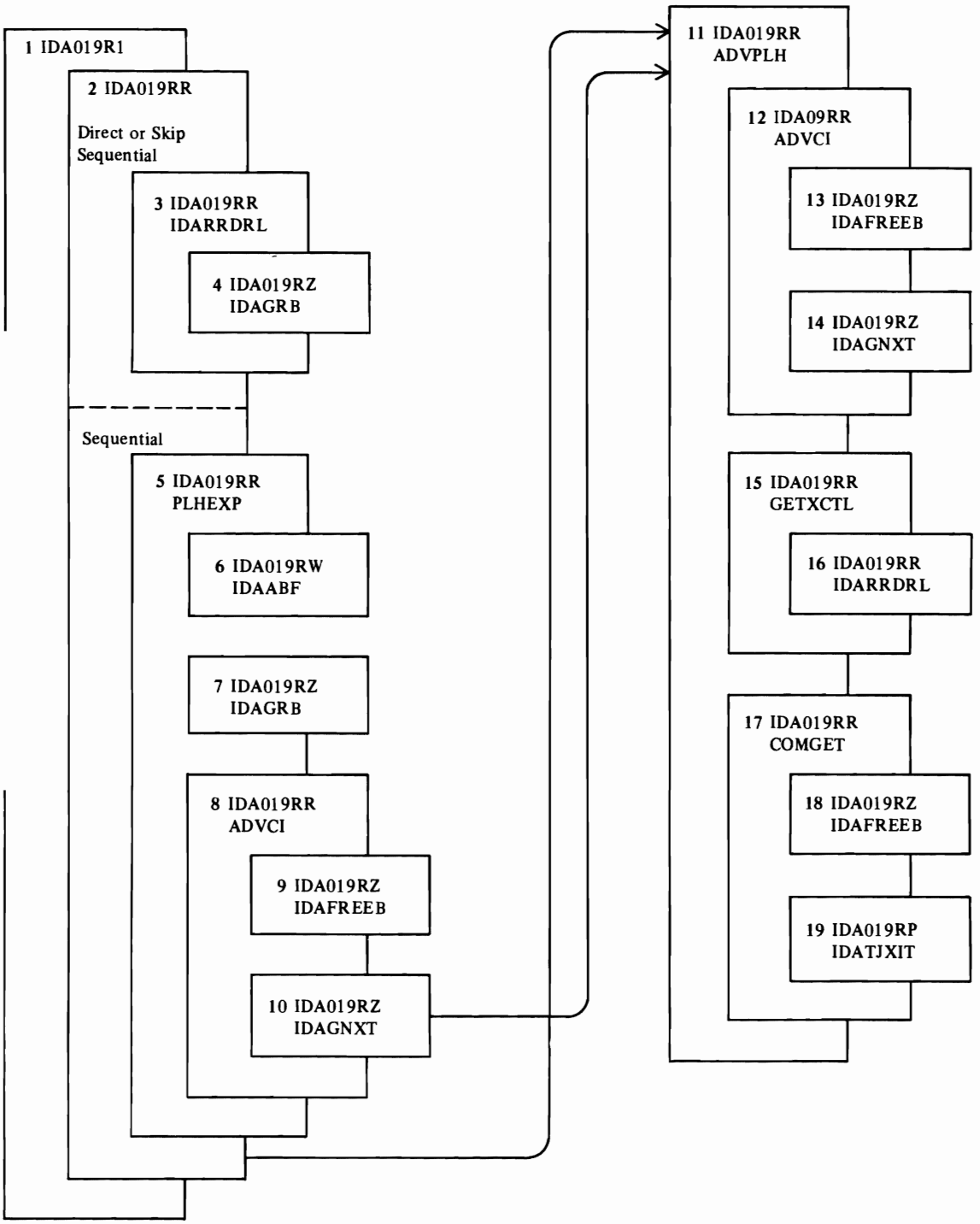



Figure 23. GET Processing (RRDS)

**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.

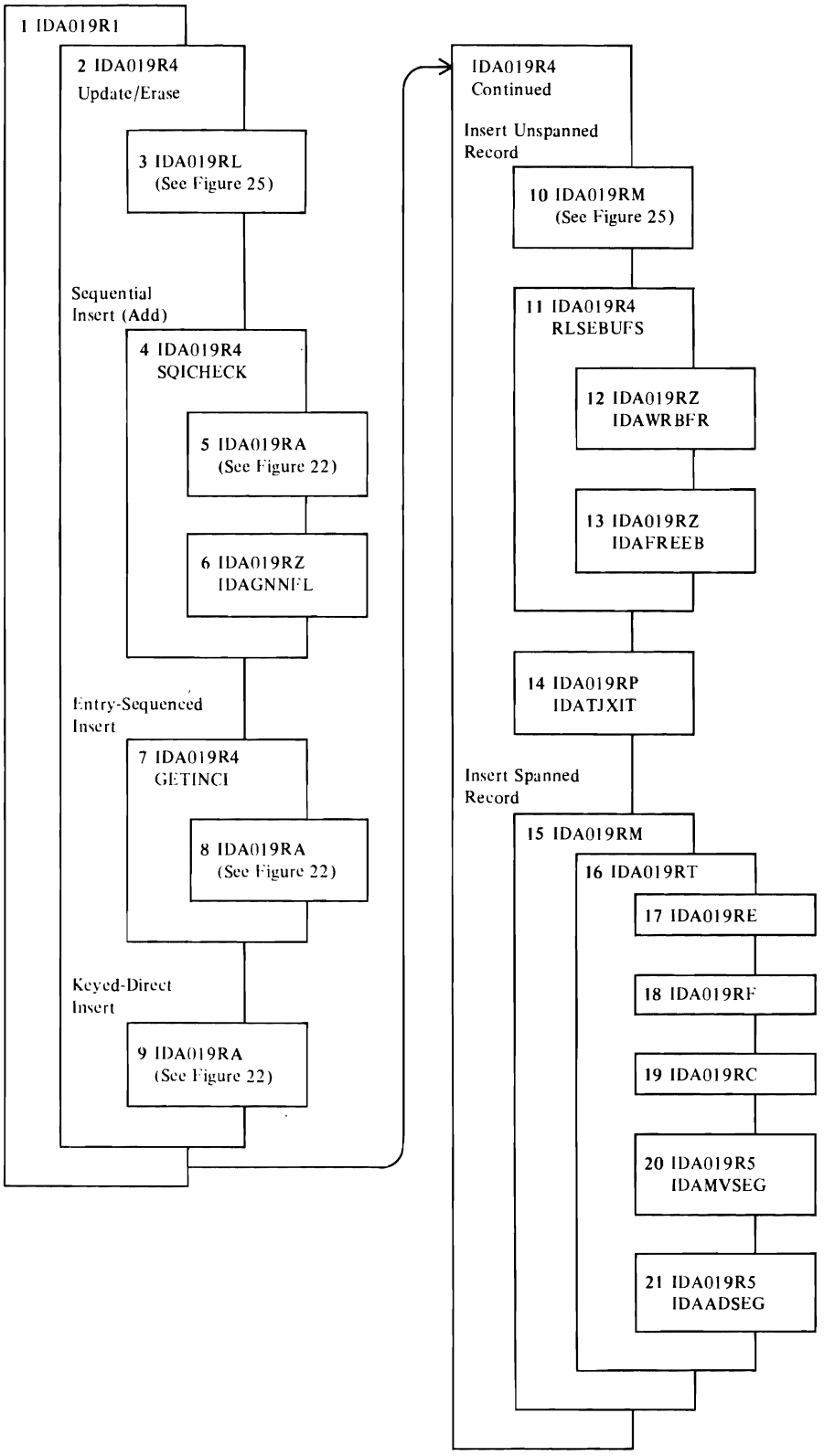



Figure 24. PUT Processing (ESDS, KSDS)

**Notes for Figure 24**

1 IDA019R1 is the common Record Management request module. It verifies that the request is a valid Record Management macro instruction, 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. 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.

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 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. IDA019R3 issues the required I/O CCW chain to rewrite the record on the direct access device.

13 IDAFREEB frees the buffer when it has been rewritten into the data set.

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.

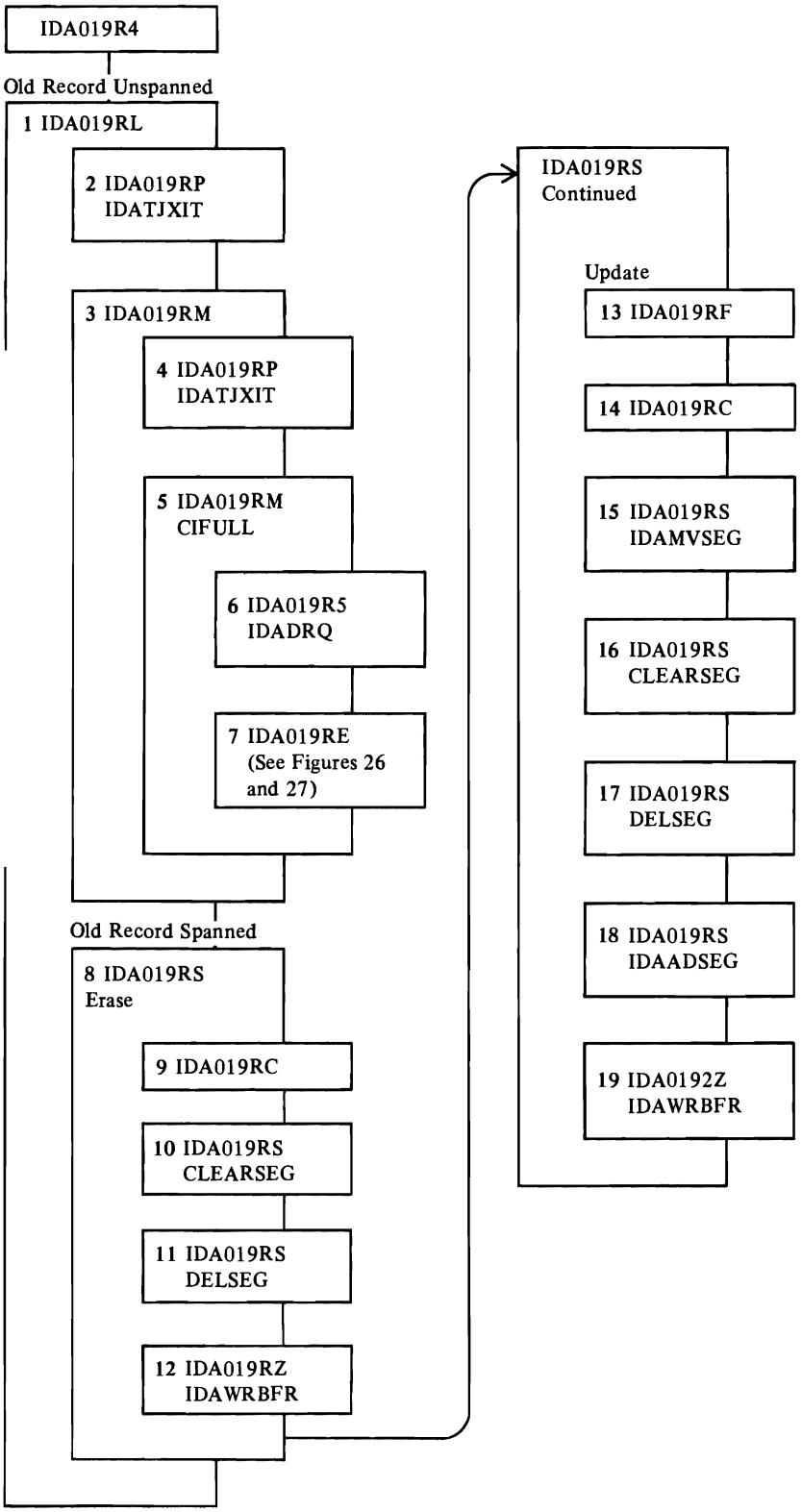



Figure 25. Update/Erase Processing (ESDS, KSDS)

**Notes for Figure 25**

**1** IDA019RL removes an unspanned record from a control interval (ERASE), updates a previously read unspanned record if the 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 journaled. 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.

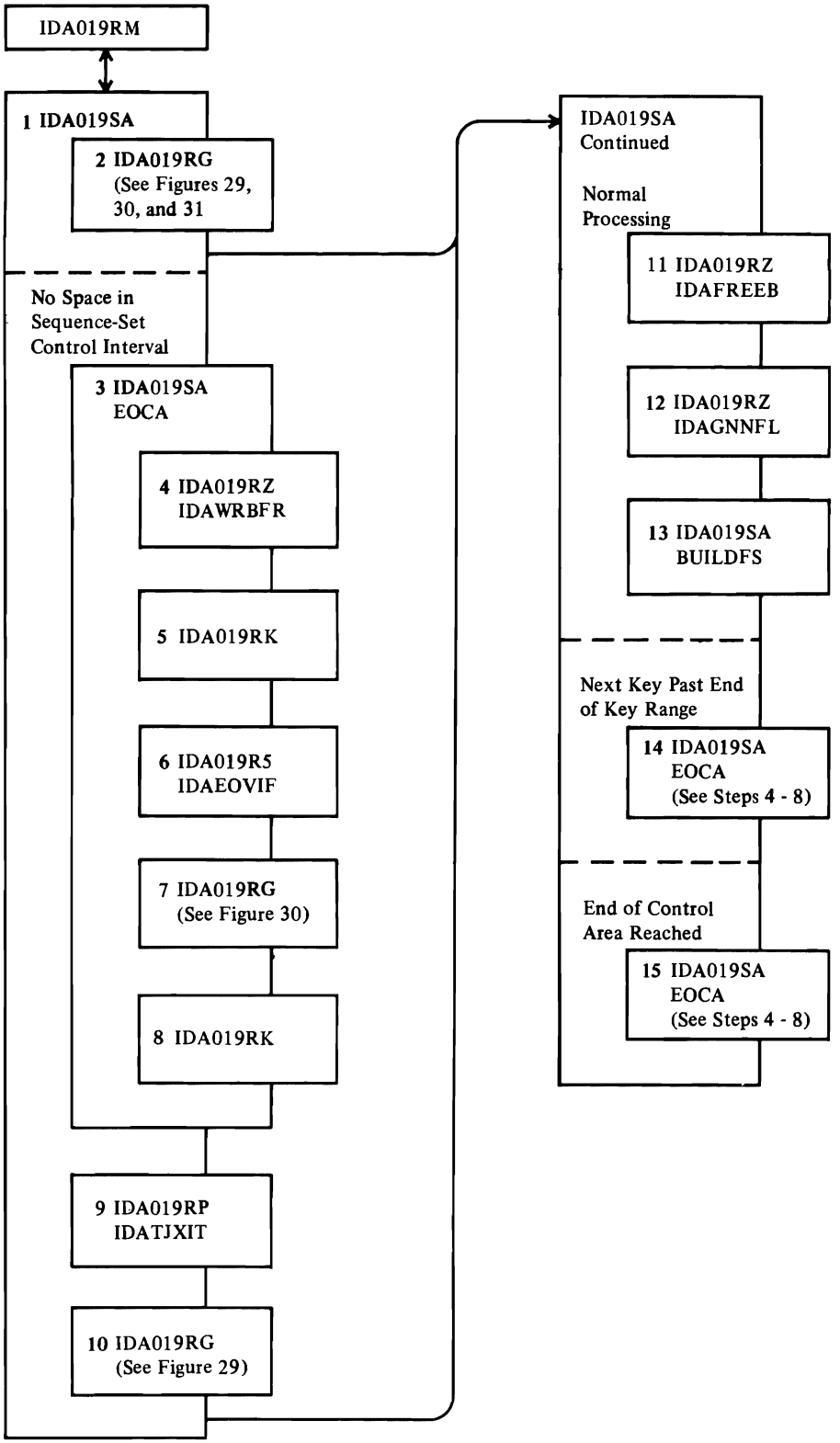



Figure 26. Obtain the Next Control Interval: Create Processing and Entry-Sequenced Data Set Processing

**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 step 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 the VSAM End of Volume routine 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.

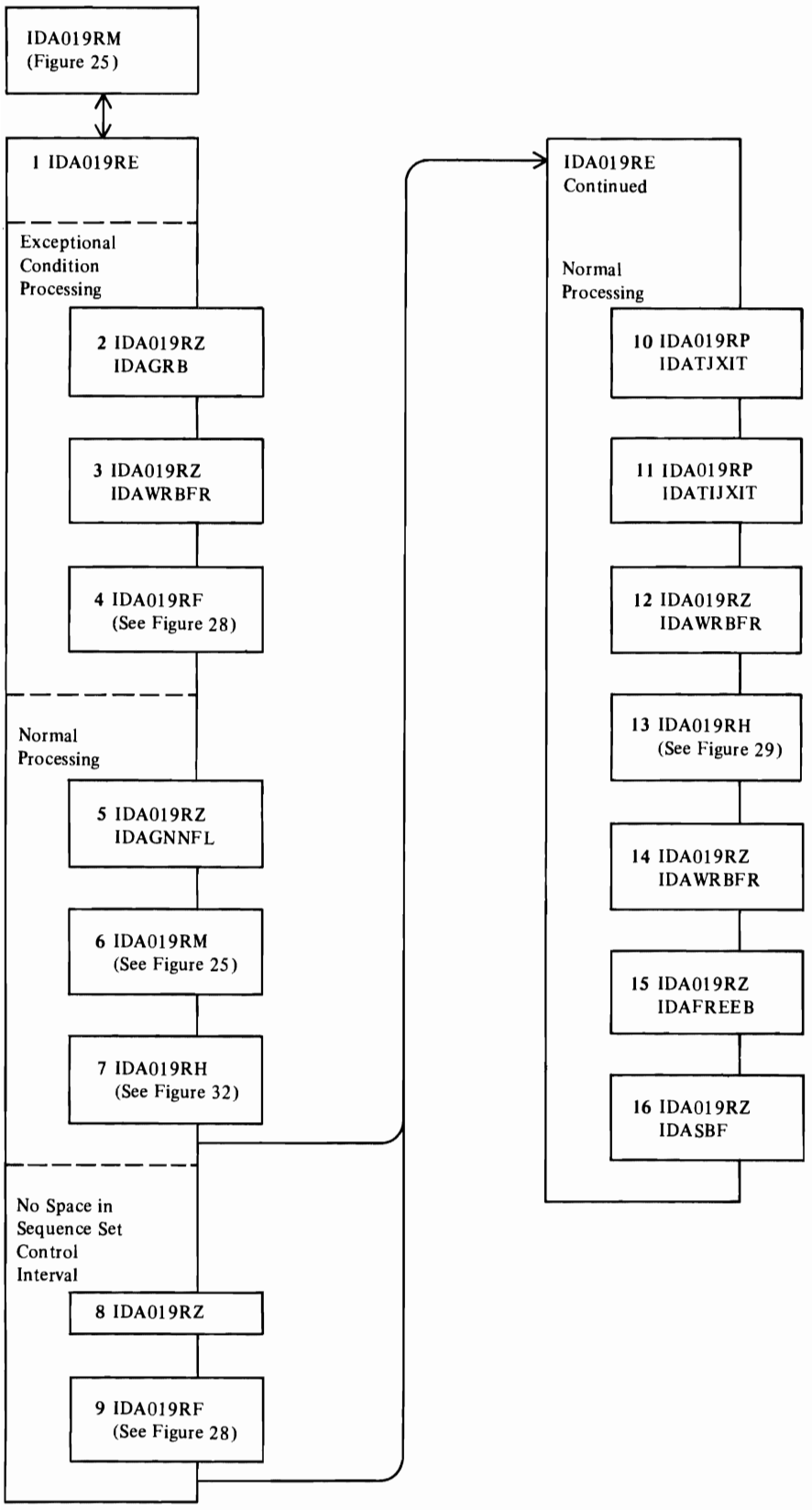



Figure 27. Split a Control Interval: Key-Sequenced Data Set, NonCreate-Time Processing

**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 step **17**).

**14** IDAWRBFR writes the updated (old) control interval into the data set.

**15** IDAFREEB frees the buffer obtained during step **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).

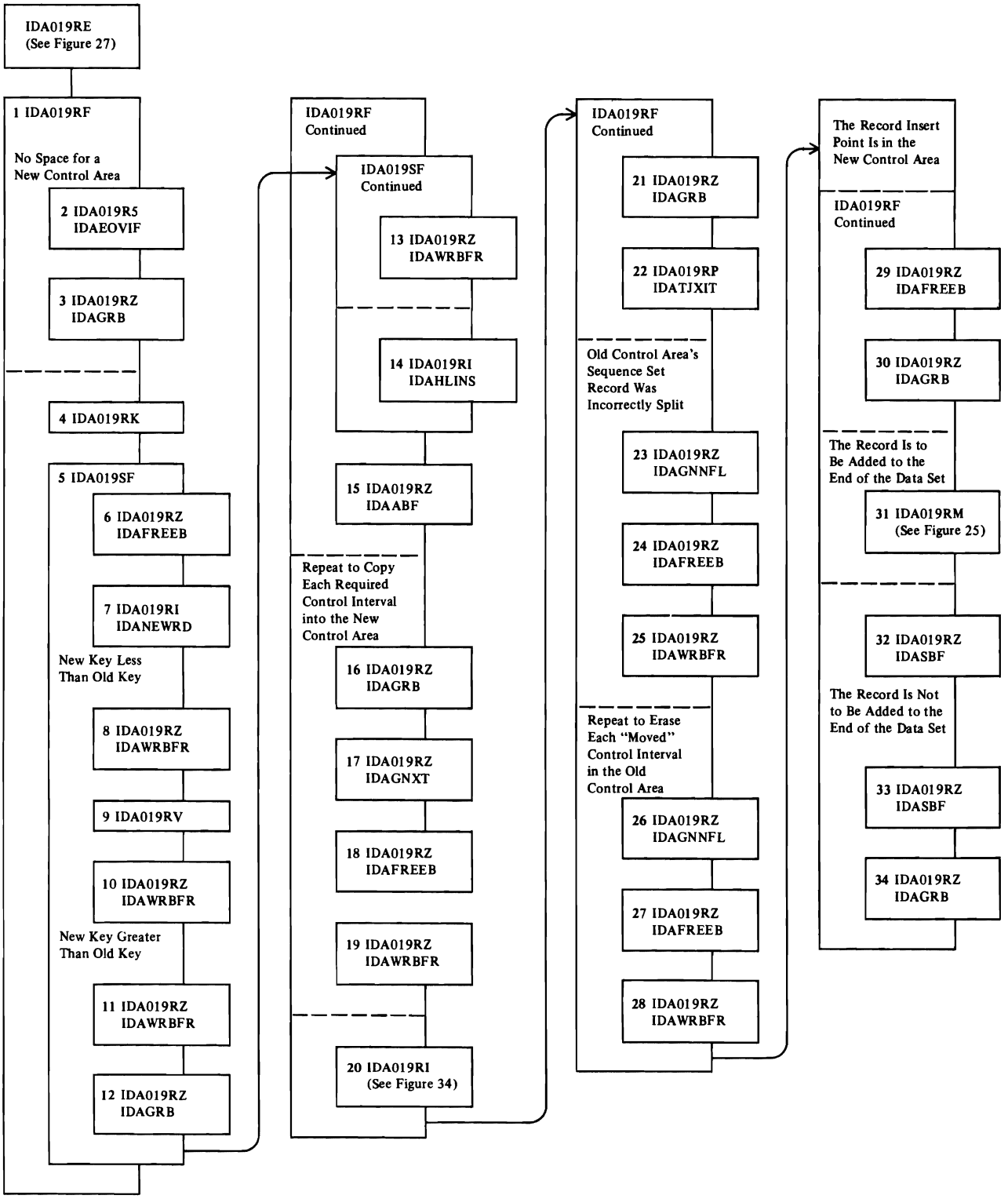



Figure 28. Split a Control Area

**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 VSAM 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 preformats 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** IDAFREEB 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 again to contain another control interval.

Steps **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. Steps **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. Steps **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.

If the record is not to be added to the end of the data set:

**32** 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 step **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.

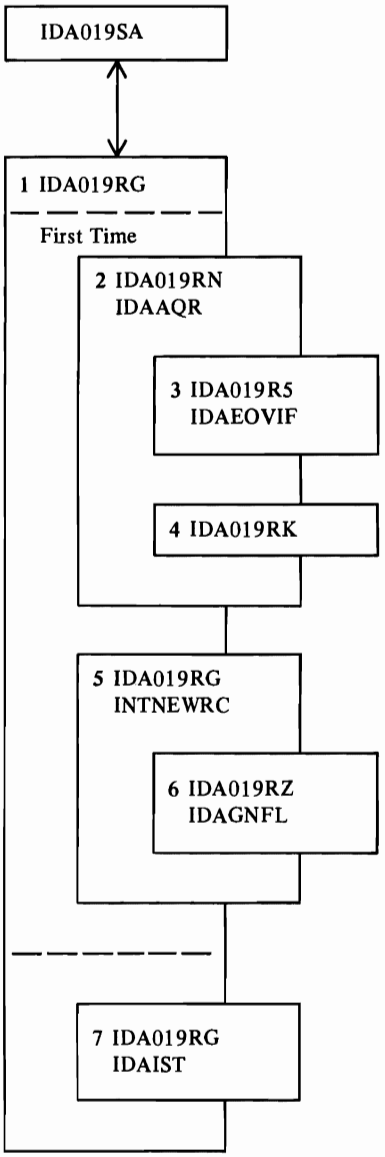



Figure 29. Create-Time Sequence-Set Record Processing: Build an Entry

**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.

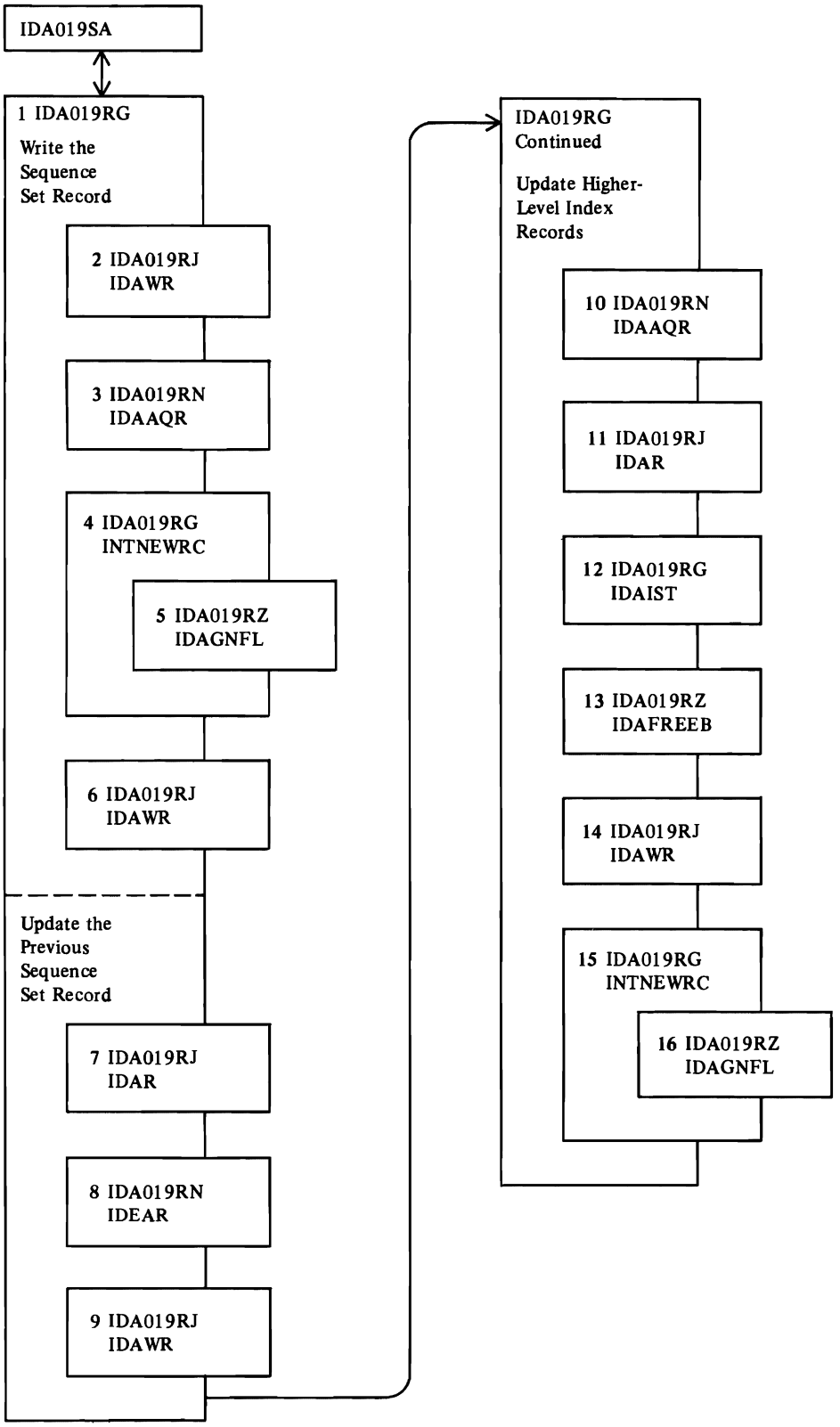



Figure 30. Create-Time Sequence-Set Record Processing: Write the Record (End of Control Area)

**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 sequencesett 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 sequencesett record.

**5** IDAGNFL obtains a buffer for the sequencesett 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 step **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 sequencesett 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 step **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.

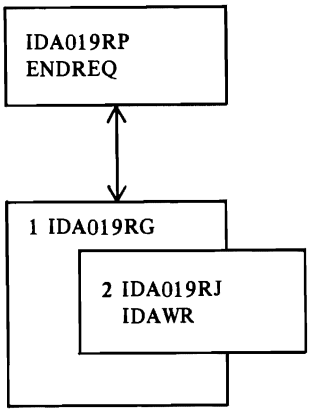



Figure 31. Create-Time Sequence-Set Record Processing: Write the Record (Closing the Data Set)

**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.

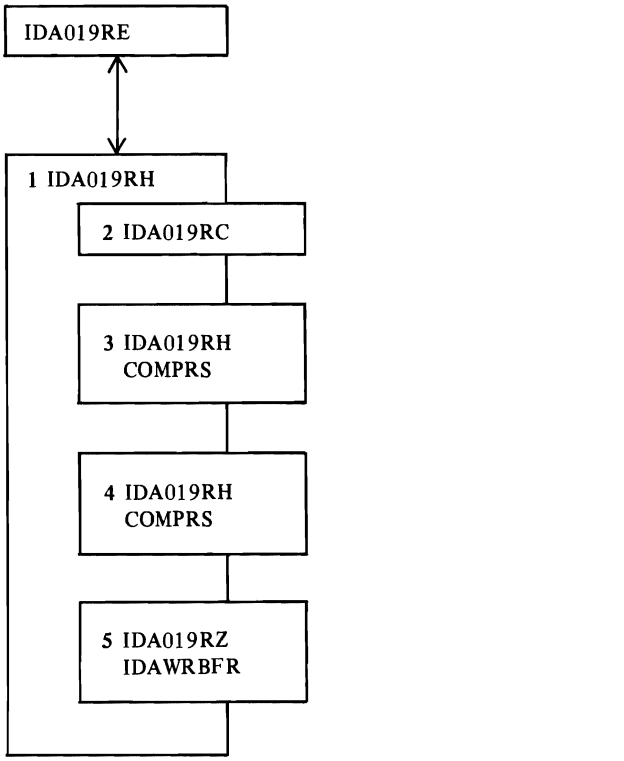



Figure 32. NonCreate-Time Sequence-Set Record Processing

**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.

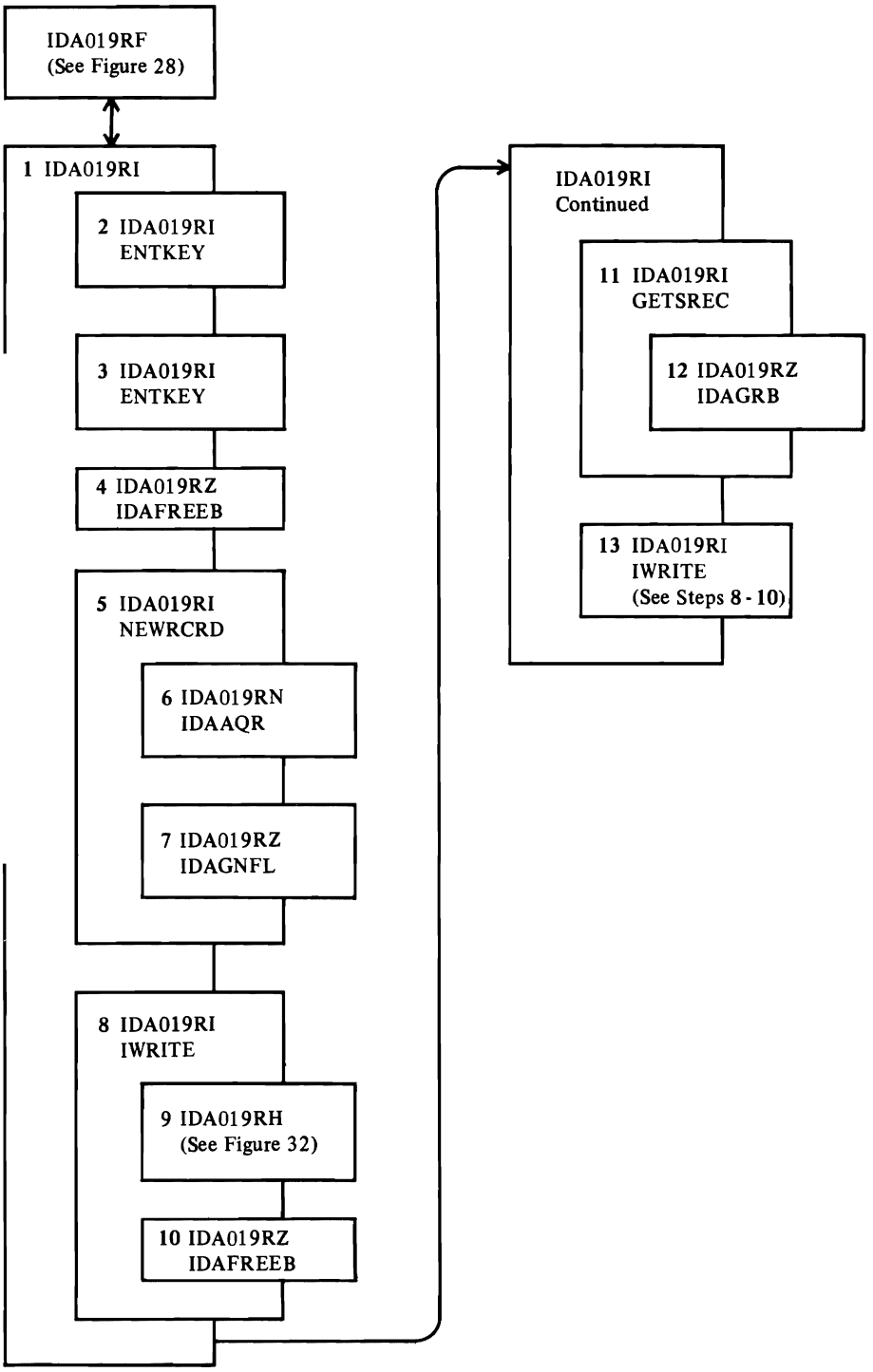



Figure 33. Update the Index: Adding to the End of a Key Range or Data Set

**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. 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 sequencesett 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.

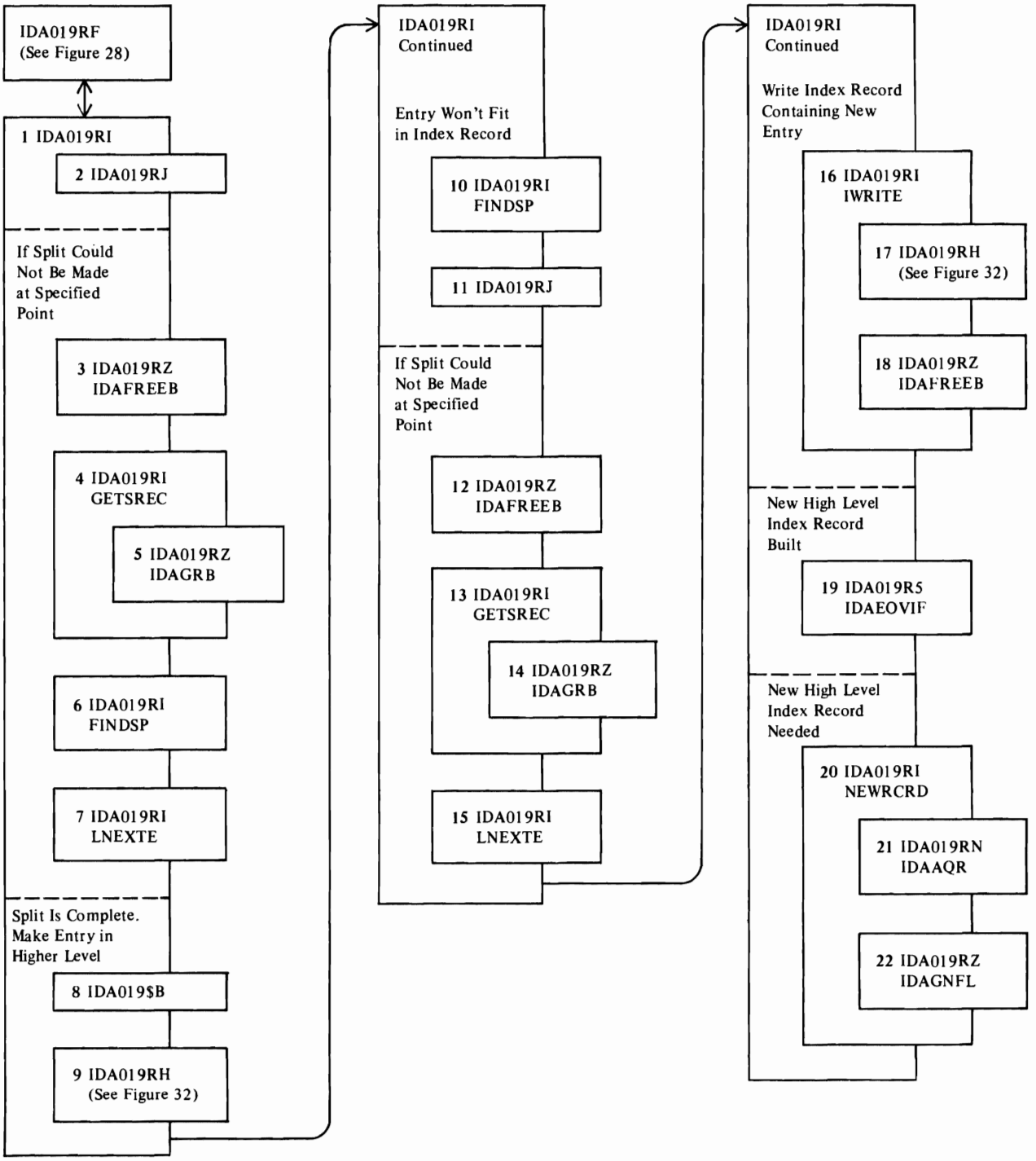



**Figure 34. Update the Index: Splitting a Control Area (Not at the End of a Key Range or Data Set)**

**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 sequencesett record.

If the sequence-set record could not be split at the specified point, steps **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 sequencesett record.

**6** FINDSP scans the sequence-set record to locate the split point.

**7** LNEXTE adjusts the split point by one entry. Step **2** is retried, and steps **3** through **7** repeat, until the sequence-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, steps **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. step **11** is retried, and steps **12** through **15** repeat, until the index record is split. When the split is correct, steps **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 steps **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.

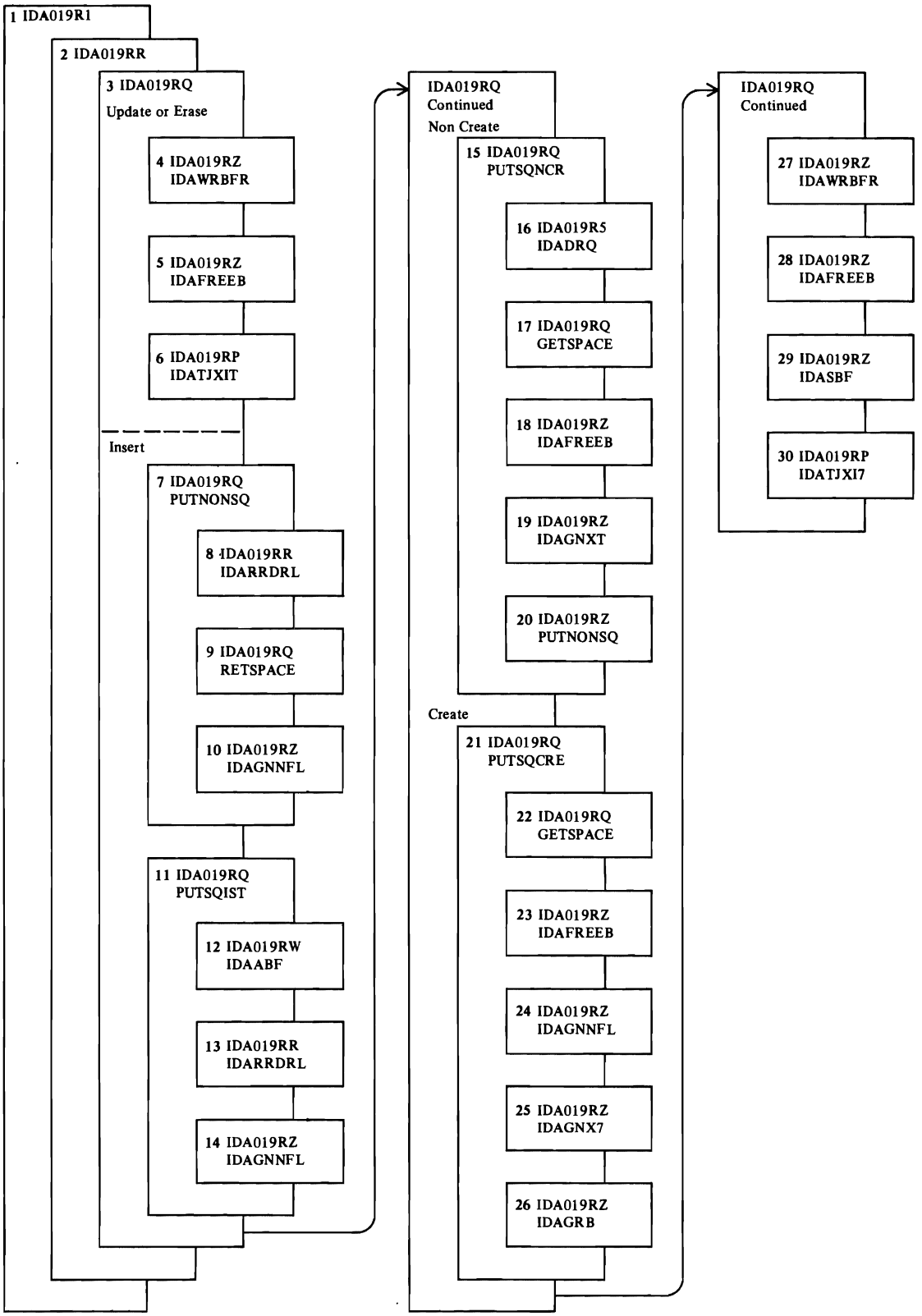



Figure 35. PUT/ERASE Processing

**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 additional 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.

**11** 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.

**Notes for Figure 35 Continued**

**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.

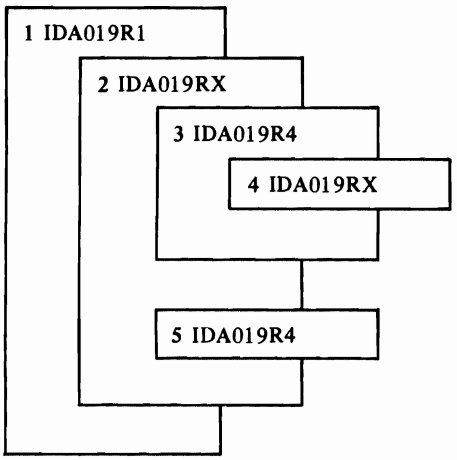



Figure 36. Path Processing

**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.

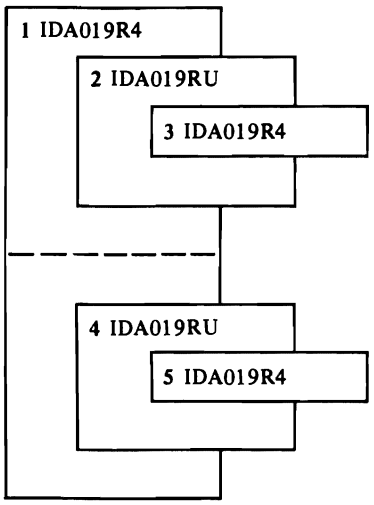



Figure 37. Upgrade Processing

**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.

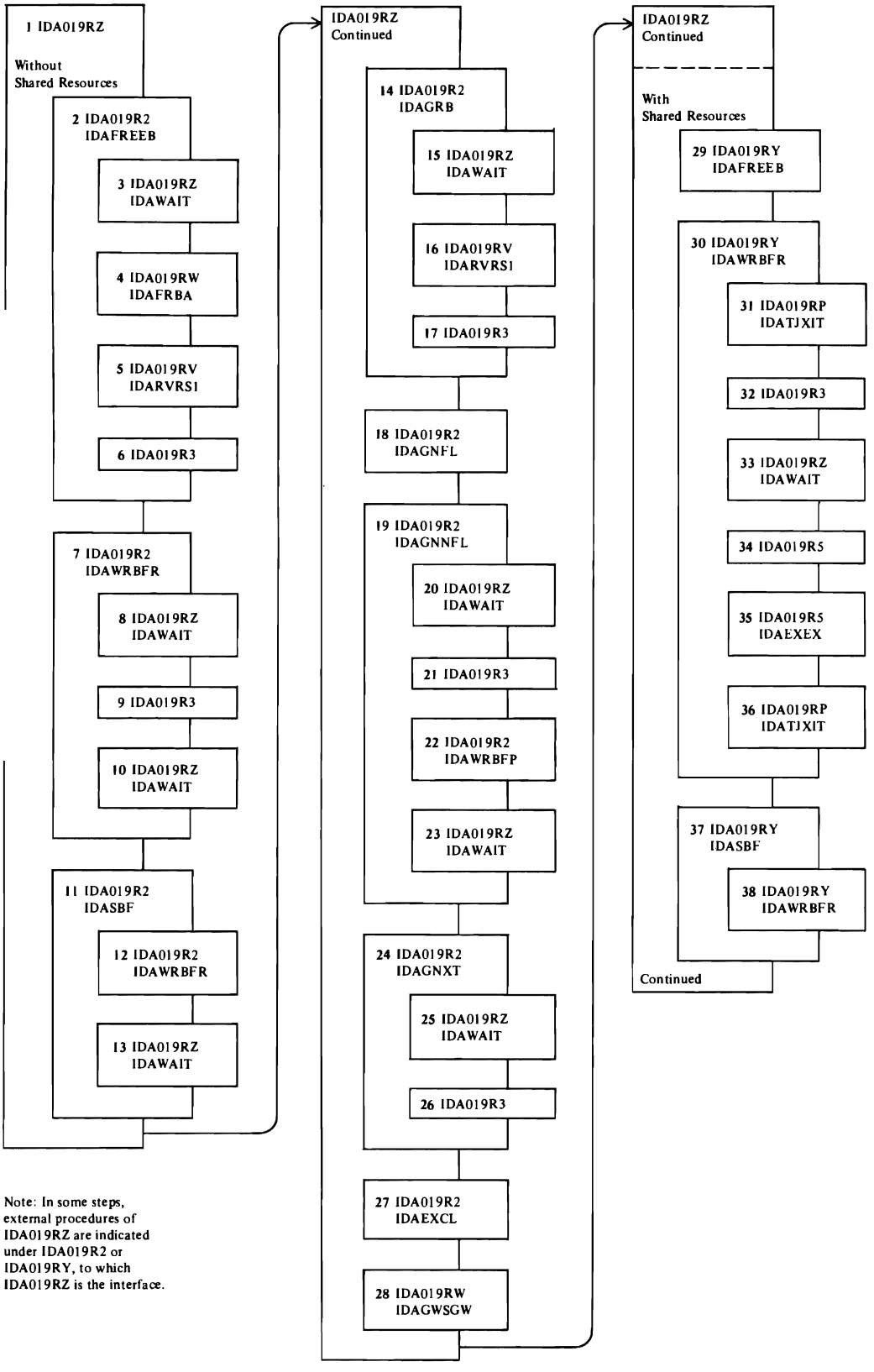



Figure 38 (Part 1 of 3). Buffer Management

**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.

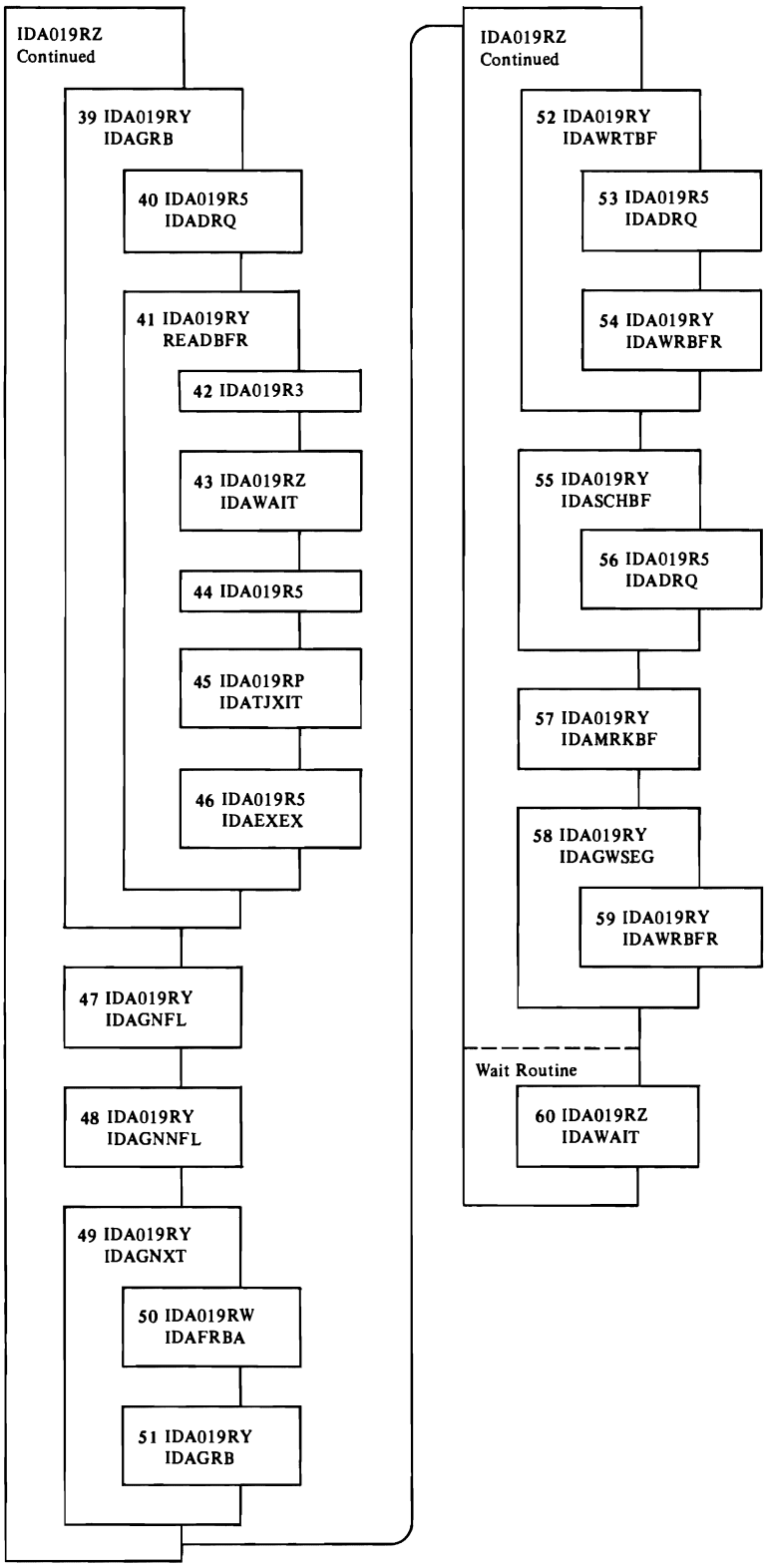



Figure 38 (Part 2 of 3). Buffer Management

**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** If the RPL specifies synchronous and WAITX, exit to the UPAD routine. If the ECB is still not posted for a synchronous request, IDAWAIT issues a WAIT macro for the I/O to finish. For an asychronous request, IDAWAIT sets a flag for the I/O Manager's Asynchronous Routine to pass control to IDAWAIT after the I/O is finished.

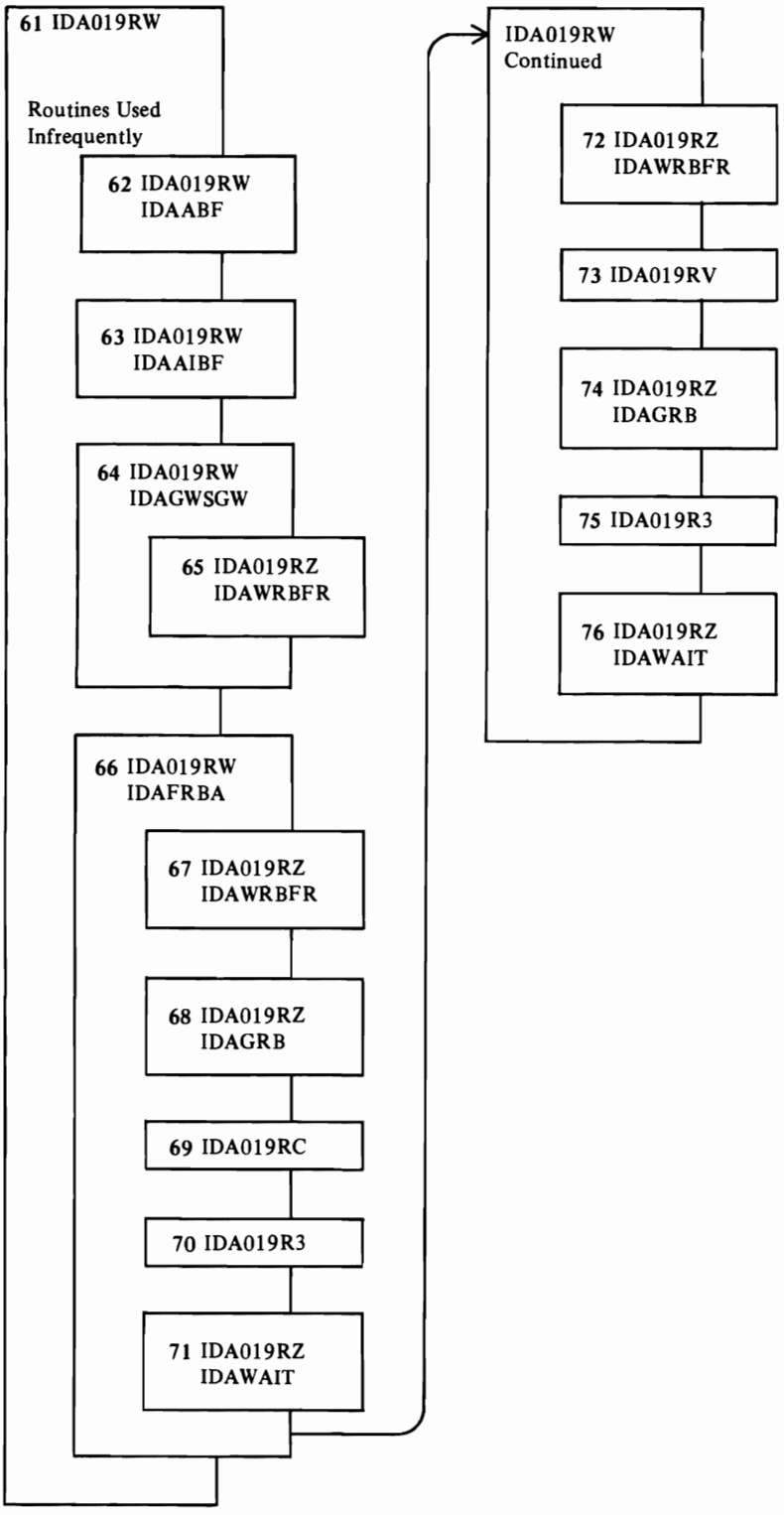



Figure 38 (Part 3 of 3). Buffer Management

**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 the 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 sequence-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.

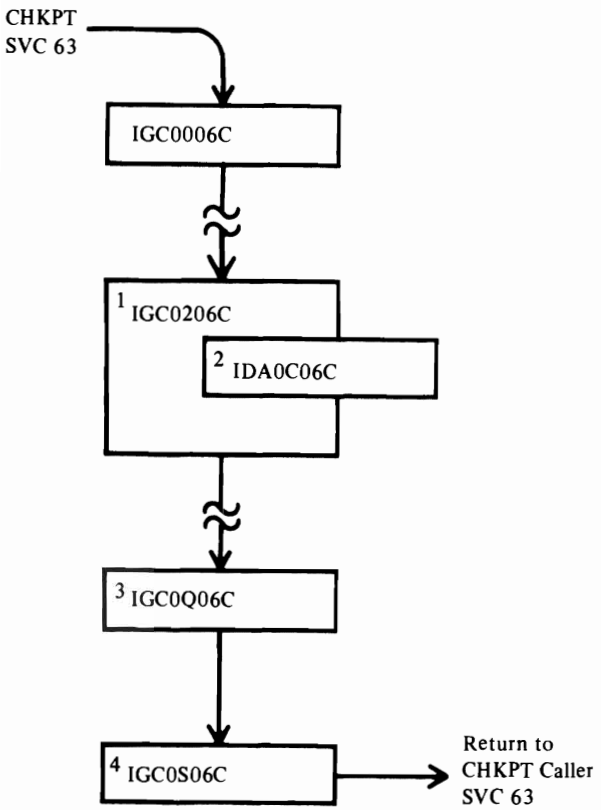



Figure 38.1 Checkpoint Processing

**Notes for Figure 38.1**

1 IGC0006C and IGC0206C are VS2 Release 1.7 checkpoint modules described in *OS/VS2 Release 1.7 Checkpoint/Restart Logic*.

2 IGC0206C loads and branches to IDA0C06C to save VSAM control block information.

3 IGC0Q06C is a VS2 Release 1.7 checkpoint module described in *OS/VS2 Release 1.7 Checkpoint/Restart Logic*. It frees the VSAM VCRWA and all VCRT's if they exist.

4 IGC0S06C is a VS2 Release 1.7 checkpoint module described in *OS/VS2 Release 1.7 Checkpoint/Restart Logic*.

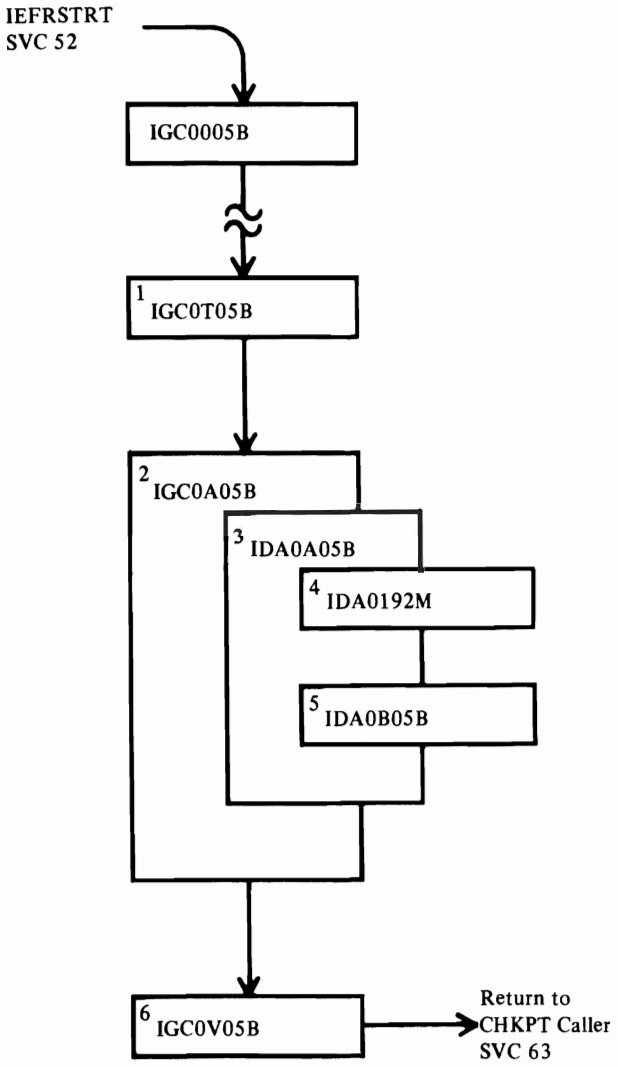



Figure 38.2 Restart Processing

**Notes for Figure 38.2**

1 IGC0005B and IGC0T05B are VS2 Release 1.7 restart modules described in *OS/VS2 Release 1.7 Checkpoint/Restart Logic*.

2 IGC0A05B loads and branches to IDA0A05B.

3 IDA0A05B is the VSAM restart module. It restores VSAM control blocks.

4 IDA0192M, the VSAM Virtual Storage Manager, is called to acquire storage for PFL's.

5 IDA0B05B is the VSAM restart module second load. It does data set repositioning and/or verification.

6 IGC0V05B is a VS2 Release 1.7 restart module described in *OS/VS2 Release 1.7 Checkpoint/Restart Logic*.

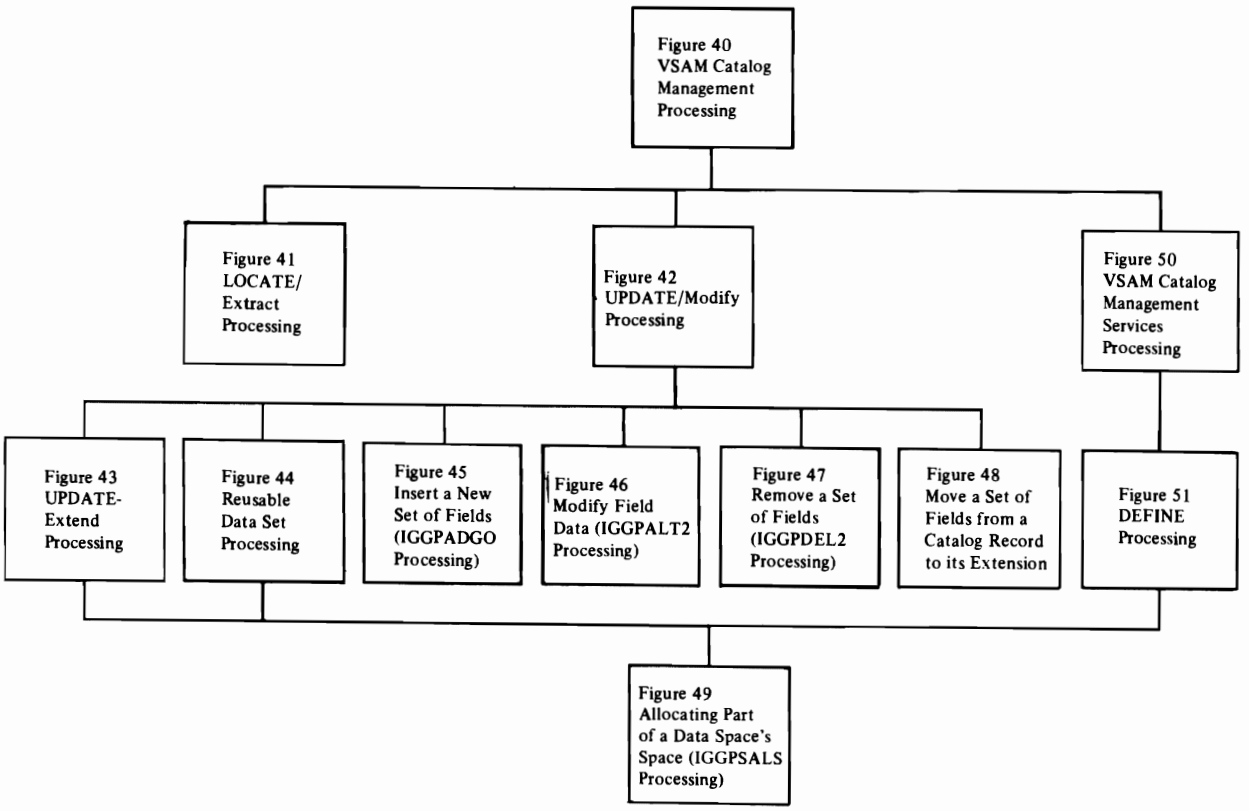



**Figure 39. Catalog Management Program Organization Contents**

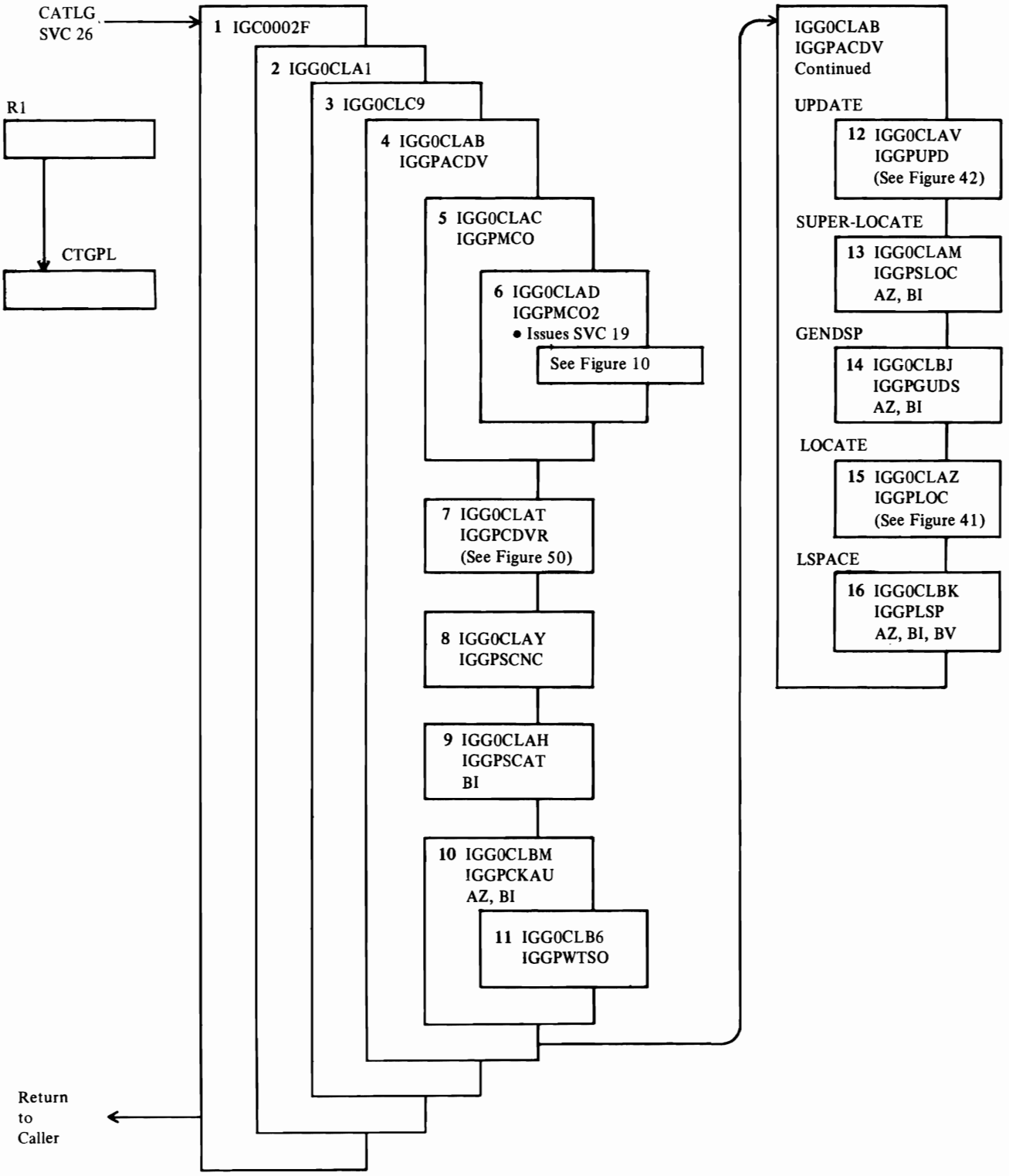



Figure 40. VSAM Catalog Management Processing

**Notes for Figure 40**

1 IGC0002F is an OS/VS Catalog Management module (see *OS/VS Catalog Management Logic* for details).

2 IGG0CLA1 is the VSAM Catalog Management transient load module.

3 IGG0CLC9 builds the CCA for the request and performs initial and final VSAM Catalog Management processing.

4 IGGPACDV is the VSAM Catalog Management Common Processing procedure.

5 When the VSAM master catalog is not open, IGGPMCO is called to open it.

6 IGGPMCO and IGGPMCO2 initialize an ACB to describe the VSAM master catalog, then issue SVC 19 to open it.

7 When the CTGPL indicates a VSAM Catalog Management Services request (DEFINE, ALTER, DELETE, or LISTCAT), the VSAM Catalog Management Services: Common Processing procedure (IGGPCDVR) is called.

8 IGGPSCNC checks and initializes the CTGFLs for the other types of Catalog Management requests (LOCATE and UPDATE).

9 IGGPSCAT retrieves the catalog record identified by the CTGPL (the VSAM object's base catalog record). Extensions to the base record are retrieved as they are needed.

BI: IGGPGET issues GET to retrieve catalog records.

10 IGGPCKAU verifies the caller's authorization to perform the CTGPL's request.

AZ: IGGPEXT locates the password information required by IGGPCKAU.

BI: IGGPGET issues GET to retrieve the object's catalog record that contains its password set of fields (group occurrence).

11 When the user is on a TSO terminal, IGGPWTSO issues requests to the TSO terminal for the required password.

12 When the caller's request is UPDATE, IGGPUPD receives control. The caller may request that his VSAM data set be extended (IGGPUPDE), that it be reset (IGGPRUS), or it be updated (IGGPUPD). If the request is for an update, only fixed-length record fields should be changed.

13 When the caller's request is SUPERLOCATE, IGGPSLOC processes it. SUPERLOCATE obtains the volume serial number of each volume that contains a part of the cluster's data set and index.

AZ: IGGPEXT locates the volume information sets of fields (group occurrences).

BI: IGGPGET issues GET to retrieve catalog records as required.

14 The caller's GENDSP request is processed by either IGGPGDSP (request for a nonunique data space) or IGGPGUDS (request for a unique data space). GENDSP obtains the control interval members of the catalog record(s) of each object (cluster, data set, index, and catalog) contained in a VSAM space identified by a DSNAME.

AZ: IGGPEXT locates the data space group set of fields (group occurrence) that describes the data space and the data set directory entry sets of fields that point to the catalog records of VSAM objects in the volume's data space.

BI: IGGPGET issues GET to retrieve volume catalog records as required.

15 When the caller's request is LOCATE, IGGPLOC processes it. The caller is allowed to retrieve fixed-length and (entire) variable-length catalog record fields.

16 When the caller's request is LSPACE, IGGPLSP processes it.

AZ: IGGPEXT locates the data space group sets of fields (group occurrences) that describe each shared (nonunique) data space on the volume.

BI: IGGPGET issues GET to retrieve volume catalog records as required.

BV: IGGPSMFL writes SMF record type 69—VSAM Data Space Defined or Deleted.

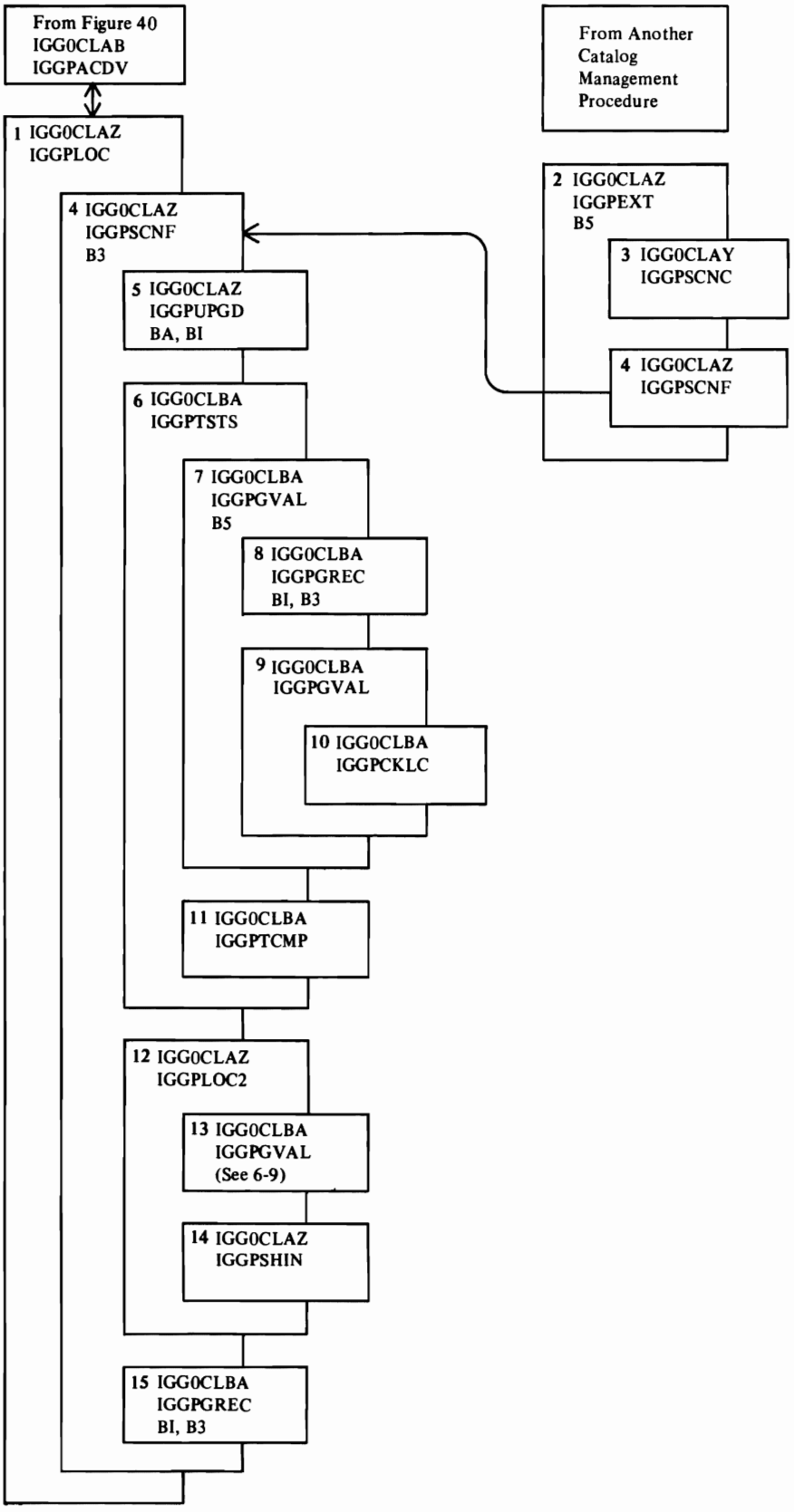



Figure 41. LOCATE/Extract Processing

**Notes for Figure 41**

**1** IGGPLOC retrieves an entire (fixed-length or variable-length) catalog record field's contents for the caller (other than an internal OS/VS2 catalog management procedure).

**2** IGGPEXT retrieves a fixed-length, variable-length, or part of a variable-length catalog record field's contents for the caller (other than an OS/VS2 catalog management procedure).

If the specified filed name indicates a "derived information" field, IGG0CLBS processes the field name.

**3** IGGPSCNC initializes CTGFLs with dictionary information required to find the field in the catalog record and ensures that the CTGFLs are valid.

**4** IGGPSCNF (steps **6** through **13**) processes each CTFGL addressed by the caller's CTGPL to retrieve all catalog information that satisfies the caller's request.

B3: IGGPSMFG makes a copy of the base catalog record in case it is updated later by a modify call.

**5** IGGPUPGD retrieves the associated upgrade entry record if this is an upgrade field name.

BA: IGGPGVAL retrieves the connecting association to the upgrade entry from the current base entry.

BI: IGGPGET issues GET to retrieve connecting entries to the upgrade entry, as well as the upgrade entry itself.

**6** When the CTGFL (addressed by the caller's CTGPL) addresses CTGFLs-for-tests, IGGPTSTS (steps **6** through **10**) processes each CTGFL-for-tests to identify each set of fields (group occurrence) that satisfies the test conditions.

**7** IGGPGVAL retrieves one catalog-record-field's value.

If the specified filed name indicates a "derived information" field, IGG0CLB5 processes the field name.

**8** If more set of fields pointers (group occurrence pointers) are in an extension of the base catalog record, or

If the specified set of fields pointer (group occurrence pointer) contains the control interval number of an extension record:

IGGPGREC retrieves the required extension record.

BI: IGGPGET issues GET to retrieve the catalog record.

B3: IGGPSMFG makes a copy of the catalog record in case it is updated later.

**9** IGGPLVAL locates the field within the catalog record.

**10** IGGPCKLC verifies that the field exists (ie. the requested field is in the catalog record or one of its sets of fields (group occurrences)).

**11** IGGPTCMP compares the catalog record field's value to the caller's test data and, if the compare is OK, saves the sequence number of the catalog-record-field's set of fields pointer (group occurrence pointer).

**12** IGGPLOC2 retrieves catalog-record-field contents to satisfy the caller's request. If the caller's CTGFL specifies a special field (one not in the catalog record) or a combination field-name (a field-name that identifies a group of related fields), IGGPLOC2 processes the field-name and calls IGGPGVAL, as required, to retrieve the requested information.

If the caller provided CTGFLs-for-tests, each catalog record field is retrieved if it is:

- Identified by the CTGFL's (addressed by the CTGPL) field name, and
- Contained in a set of fields (group occurrence) that satisfies all tests associated with the CTGFL (addressed by the CTGPL). The set of fields pointer's (group occurrence pointer's) sequence number is set by step **11**.

If the caller didn't provide CTGFLs-for-tests, the contents of each catalog record field identified by the CTGFL's field name is retrieved.

**13** IGGPGVAL retrieves each catalog-record-field's contents, as required by IGGPLOC2.

**14** IGGPSHIN places the catalog record field's contents into the user-provided work area addressed by CTGPL and increments the required work area length. If there is insufficient space in the work area, only the required work area length is changed.

**15** IGGPGREC retrieves the original base catalog record, if necessary, since a horizontal extension of the base or an associated upgrade entry record may have overlaid it.

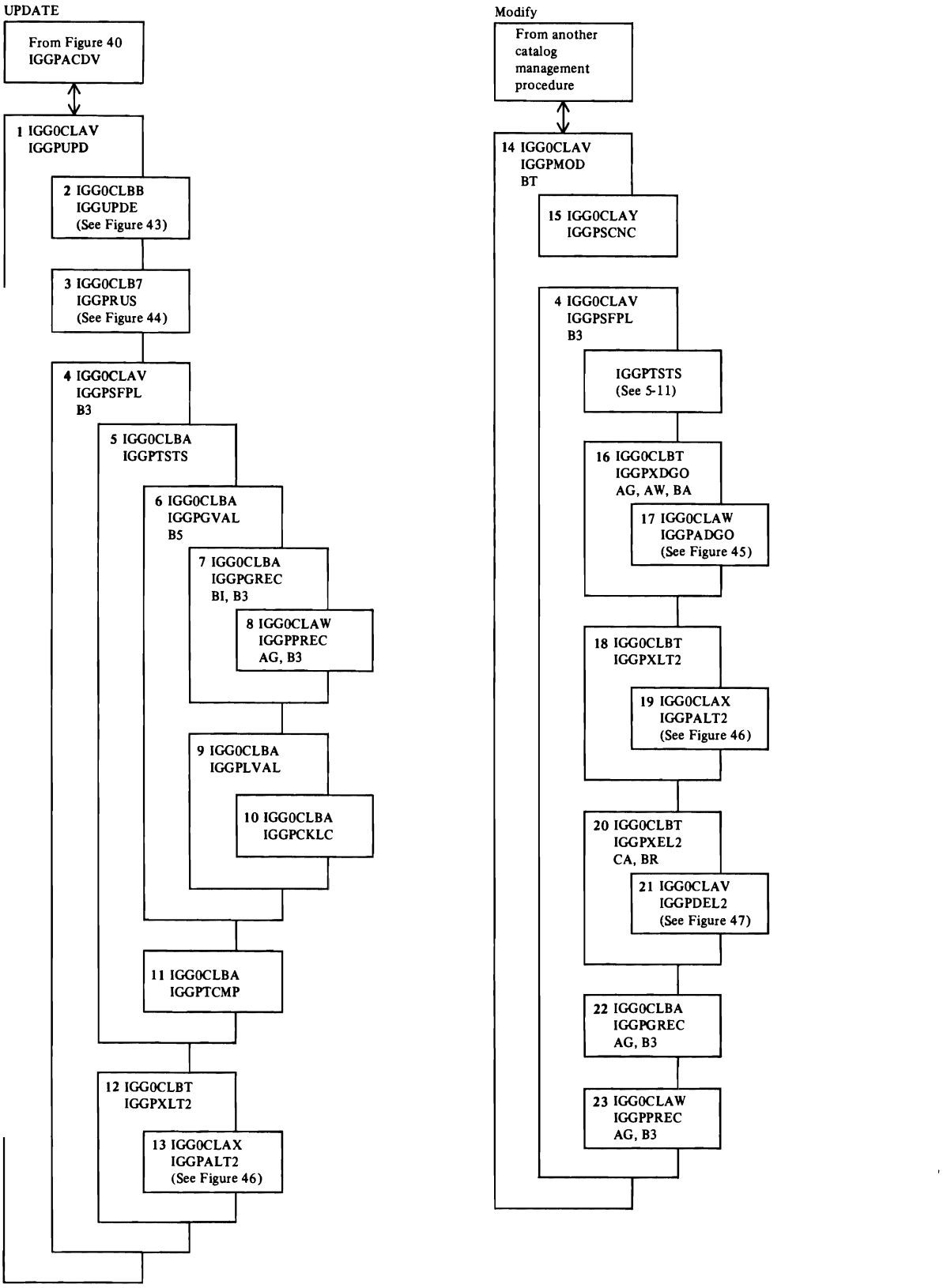



Figure 42. UPDATE/Modify Processing

**Notes for Figure 42**

**1** IGGPUPD modifies a fixed-length catalog record field, obtains more space for a VSAM object, or calls IGGPRUS to reset a VSAM data set.

**2** IGGPUPDE obtains more space for a VSAM object.

**3** IGGPRUS resets a VSAM data set.

**4** IGGPSFPL (steps **4** through **10** ) processes each CTGFL addressed by the caller's CTGPL to modify all catalog record field data specified by the caller's request.

**5** When the CTGFL (addressed by the CTGPL) addresses CTGFLs-for-tests, IGGPTSTS (steps **6** through **10**) processes each CTGFL-for-tests to identify each set of fields (group occurrence) that satisfies the test conditions.

**6** IGGPGVAL retrieves one catalog-record-field's value.

If the specified field name indicates a "derived information" field, IGG0CLBS processes the field.

**7** If more set of fields pointers (group occurrence pointers) are in an extension of the base catalog record, or

If the specified set of fields pointer (group occurrence pointer) contains the control interval number of an extension record,

IGGPGREC retrieves the required extension record.

BI: IGGPGET issues GET to retrieve the catalog record.

B3: IGGPSMFG makes a copy of the catalog record in case it is updated later.

**8** IGGPPREC writes the contents of the buffer (a catalog record) prior to reading another record into it, if the "buffer-must-be-written" indicator is on.

AG: IGGPPUPC issues PUT-update to rewrite an updated catalog record.

AG: IGGPPAD issues PUT-Add to insert a new catalog record into the catalog.

B3: IGGPSMF identifies the copy of the original catalog record (saved by IGGPSMFG) as an updated record.

**Note:** When the catalog record is completely updated, a SMF record type 63—VSAM Data Set Cataloged—is written that contains the entire new catalog record (the base and all extensions) and each part of the original catalog record that was modified (part=logical catalog record=505-byte (or less) base or extension record's contents).

**9** IGGPLVAL locates the field within the catalog record.

**10** IGGPCKLC verifies that the field exists (that is, the requested field is in the record or one of its sets of fields (group occurrences)).

**11** IGGPTCMP compares the catalog record field's value to the caller's test data and, if the compare is OK, saves the sequence number of the catalog-record-field's set of fields pointer (group occurrence pointer).

**12** IGGPXLT2 filters those field names (derived) which do not exist physically in the catalog. It ensures that these fields are not updated; all others are passed to IGGPALT2.

**13** IGGPALT2 replaces a catalog record field's contents with the caller's update data.

If the caller provided CTGFLs-for-tests, each catalog record field is updated if it is:

- Identified by the CTGFL's (addressed by the CTGPL) field name, and
- Contained in a set of fields (group occurrence) that satisfies all tests associated with the CTGFL (addressed by the CTGPL). The set of fields pointer's (group occurrence pointer's) sequence number is available from step **11**.

If the caller didn't provide CTGFLs-for-tests, each set of field's field identified by the CTGFL's field name is updated.

**14** IGGPMOD allows a VSAM catalog management procedure to update catalog record information in the following ways:

A new set of fields (group occurrence) is added to the record (IGGPXDGO processing).

A set of fields (group occurrence) is removed from the catalog record (IGGPXEL2 processing).

A fixed-length field, variable-length field, or part of a variable-length catalog record field's contents is modified (IGGPXLT2 processing).

**15** IGGPSCNC initializes the CTGFLs with dictionary information required to find the field in the catalog record and ensures that the CTGFLs are valid.

**16** IGGPXDGO intercepts field names (derived) which do not exist physically in the catalog. All others are passed to IGGPADGO. It constructs a bit map set of fields when the first data space group is added and updates the associated data space group when data space descriptors are added. Note that derived field names exist only in the volume entry record.

AG: IGGPAOCI obtains a control interval for constructing the bit map set of fields in an extension record.

AG: IGGPPAD adds the newly constructed bit map record.

AW: IGGPPREC updates the base volume entry record which points to the bit map set of fields.

BA: IGGPGVAL retrieves the data space group associated with the space descriptor group to be added.

BA: IGGPGREC retrieves the base volume entry record so that bit map processing can be done.

BR: IGGPBMR updates the bit map to reflect the added space.

**17** When the caller provides set of fields (group occurrence) field data, but doesn't provide CTGFLs-for-tests, IGGPADGO builds a new set of fields (group occurrence) with the caller's field data and adds it to the catalog record.

**18** IGGPXLT2 filters those field names (derived) which do not exist physically in the catalog. It ensures that these fields are not updated; all others are passed to IGGPALT2.

**Notes for Figure 42 Continued**

**19** When the caller provides header-field field data, or when the caller provides set of fields (group occurrence) field data and CTGFLs-for-tests, IGGPALT2 modifies the field's contents (as per step **13** above) and makes all necessary adjustments to the catalog records.

**20** IGGPXEL2 causes the bit map set of fields to be updated when a data space group is to be deleted. All set-of-field names, both derived and nonderived, are passed to IGGPDEL2.

BA: IGGPGVAL retrieves the data space group to be deleted.

BA: IGGPGREC retrieves the base volume entry record so bit map processing can be done.

BR: IGGPBMR updates the bit map to reflect the released space.

**21** When the caller doesn't provide field data, IGGPDEL2 deletes catalog record sets of fields (group occurrences).

If the caller provides CTGFLs-for-tests, all sets of fields (group occurrences) identified by IGGPTSTS (see step **10**) are deleted.

If the caller didn't provide CTGFLs-for-tests, only those sets of fields (group occurrences) that contain the field identified by the CTGFL's (addressed by the CTGPL) field name are deleted.

**22** IGGPGREC retrieves the original base catalog record for processing the next CTGFL, since a horizontal extension record may have overlaid it.

BI: IGGPGET issues GET to retrieve the catalog record.

B3 IGGPSMFG makes a copy of the catalog record in case it is updated later.

**23** IGGPPREC flushes any catalog buffers that must be written.

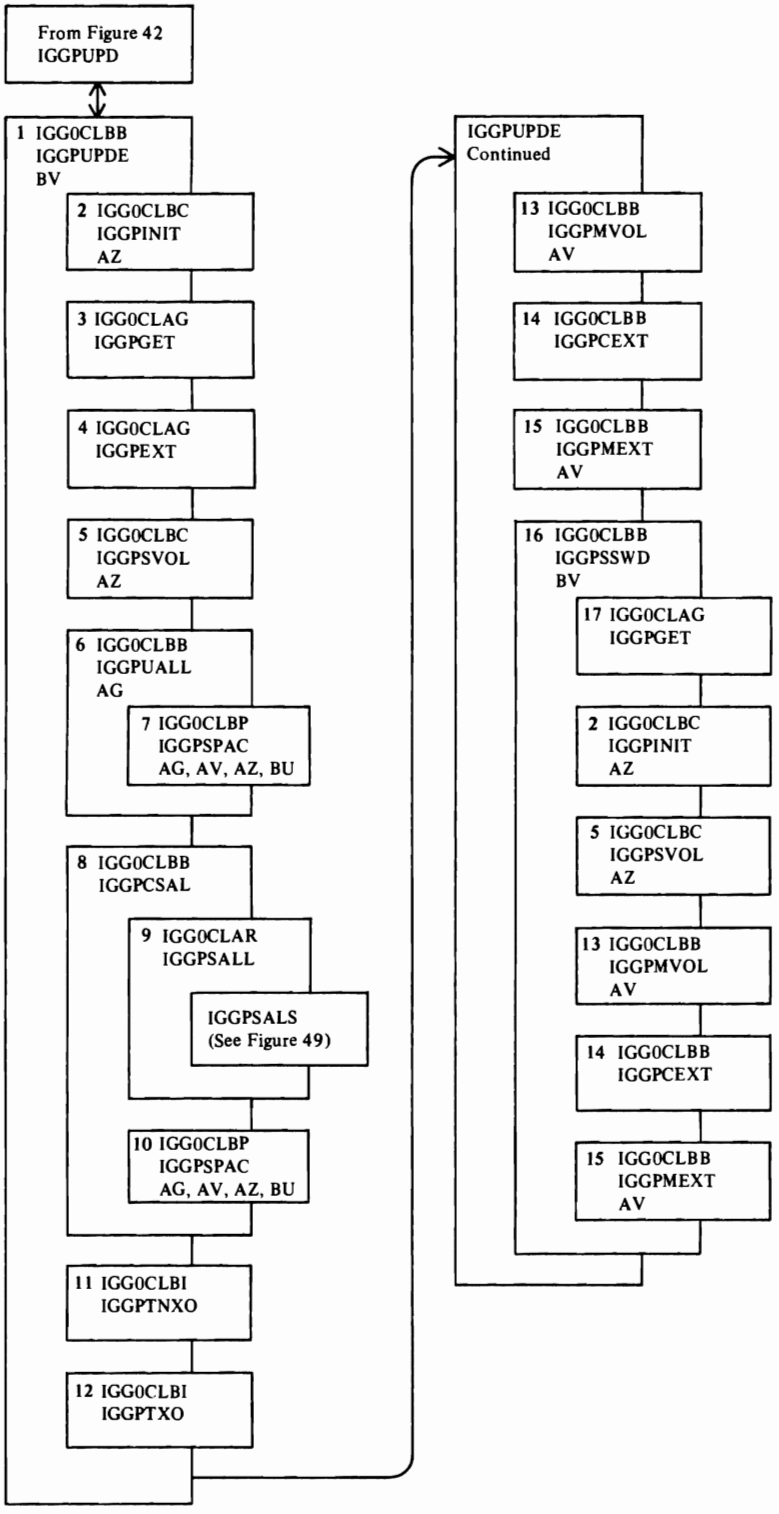



Figure 43. UPDATE-Extend Processing

**Notes for Figure 43**

1 IGGPUPDE obtains more space for the VSAM object.

BV: IGGPSMFA writes SMF record type 63—VSAM Data Set Cataloged or Altered—for the data set catalog record.

BV: IGGPSMFL writes SMF record type 69—VSAM Data Space Defined, Extended, or Deleted—if additional space was obtained by DADSM, either to create a new VSAM data space or to extend an existing VSAM data space.

2 IGGPINIT initializes a CTGPL and CTGFLs and calls IGGPEXT.

AZ: IGGPEXT retrieves the AMDSB set of fields.

3 IGGPGET retrieves the base index record for key-sequenced data sets that have sequence set with data.

4 IGGPEXT retrieves the index AMDSB for key-sequenced data sets that have sequence set with data to ensure that the maximum number of extents for the index component will not be exceeded.

5 IGGPSVOL finds the volume information set of fields that describes the volume that contains the VSAM object to be extended.

AZ: IGGPEXT locates the volume information set of fields.

6 When the object is in a non-shared (unique) data space, the object is allowed to reside in only one data space (16 extents maximum) per volume.

If the data space exists and contains less than 12 extents, IGGPUALL calls IGGPSPAC to obtain another extent(s) for the data space. IGGPSPAC will return a maximum of 5 extents to satisfy the request.

If the data space already contains 16 extents, no space is allocated to the object from the specified volume.

If the data space doesn't exist (the volume is a candidate volume for the object), IGGPUALL calls IGGPSPAC to build a data space on the volume for the object.

AG: IGGPGET issues GET to retrieve the volume catalog record.

7 IGGPSPAC calls OS/VS DADSM to obtain space for a VSAM data space (EXTEND) or to create a new VSAM data space (ALLOCATE).

AG: IGGPISCI ensures that a catalog or CRA extend will not occur while the catalog volume entry is being modified.

AG: IGGPPUPC updates the base catalog volume entry record after the new timestamp value has been set in it.

AV: IGGPMOD updates the volume catalog record fields.

AZ: IGGPEXT retrieves volume catalog record fields.

BU: IGGPF4DQ, IGGPF4RD, and IGGPF4WR update the physical volume's format-4 DSCB with new timestamp values.

8 When the object is in a shared (nonunique) data space, the object is allowed to reside in many data spaces (123 extents maximum per volume). **Note:** An extent for an object in a shared data space is a contiguous amount of tracks in one of the data space's extents (obtained by OS/VS DADSM).

IGGPCSAL attempts to obtain more space for the object from any shared data space on the volume.

9 IGGPSALL calls IGGPSALS to attempt to obtain the required amount of space from the free space of one of the shared (nonunique) data spaces on the volume.

**Note:** All of the requested amount of space can be obtained from more than one shared (nonunique) data space. If the amount of space necessary to satisfy the request is not contiguous, IGGPSALS obtains (a maximum of five) contiguous amounts of tracks to satisfy the request.

10 When all shared (nonunique) data spaces on the volume have been examined, and none can satisfy the request, IGGPSPAC attempts to obtain another extent(s), at least large enough to satisfy the request, for any shared (nonunique) data space. If all shared data spaces have 12 or more extents, IGGPSPAC builds a new data space on the volume (if the volume contains enough contiguous free space to satisfy a data space's primary allocation requirements). If IGGPCSAL called IGGPSPAC to obtain more space from OS/VS DADSM, IGGPCSAL calls IGGPSALL again to suballocate part of the newly obtained space to the object.

11 IPPGTNXO computes the sum of the beginning CCHHs (converted to relative track numbers) for the newly acquired extents (recoverable catalogs only).

12 IGGPTXO updates the data set directory with the new sum computed in step **11**.

13 IGGPMVOL updates the volume information set of fields to describe the object's newly obtained space.

AV: IGGPMOD modifies the volume information set of field's fixed-length fields and to modify statistical information in the AMDSB.

14 IGGPCEXT builds extent descriptors to insert in the volume information set of fields. The extent descriptors describe each contiguous amount of newly allocated space. IGGPCEXT computes the RBAs and CCHH values from the information returned by IGGPSALL:

CCHH NN Desc#

where:

CCHH is the starting cylinder and track number of the extent,

NN is the number of tracks in the extent, and

Desc# is the data space descriptor's sequence number.

**Notes for Figure 43 Continued**

**15** IGGPMEXT inserts the extent descriptors into the volume information set of fields.

AV: IGGPMOD adds the extent descriptor(s) to the volume information set of fields (group occurrence).

**16** When the space is obtained for a sequence-set-with-data object (a data set or key range), IGGPSSWD allocates part of the newly obtained space to the object's sequence set and modifies the object's index catalog records to reflect this.

Initial IGGPSSWD processing:

IGGPGET retrieves the index catalog record. The index entry is then processed in a manner similar to the data entry except that space has already been acquired (hence, neither IGGPUALL nor IGGPCSAL is called).

BV: IGGPSMFA writes SMF record type 63—VSAM Data Set Cataloged or Altered—for the index catalog record.

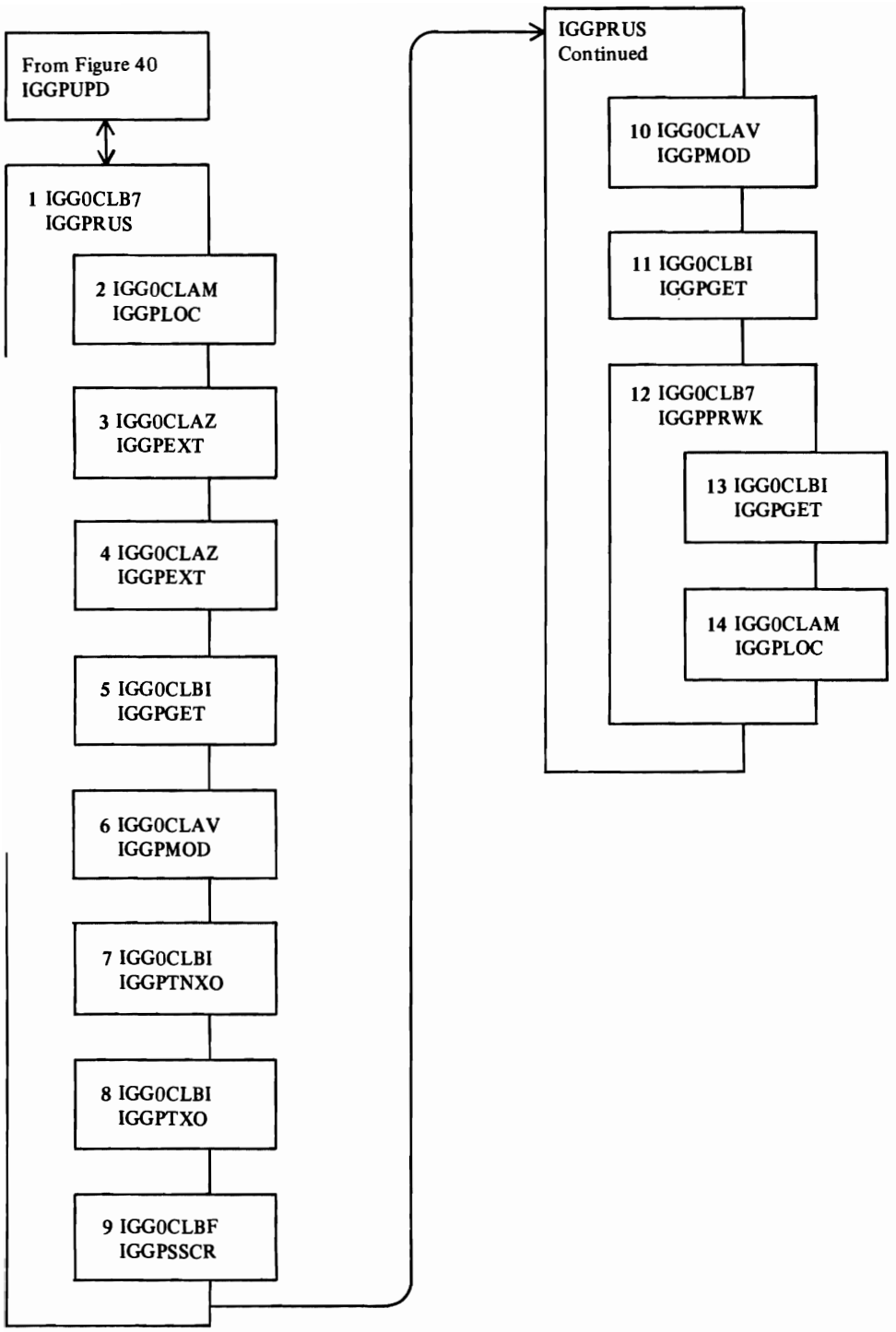



Figure 44. Reusable Data Set Processing

**Notes for Figure 44**

This figure describes the processing done when a caller wants to reset (reuse) a VSAM data set. The reusable attribute may be applied to a VSAM key-sequenced, entry-sequenced, and relative record data set, and to an alternate index. The attribute allows the data set to have its high-used RBA set to 0 at OPEN time. Reusable data sets may be multi-volumed and they must be suballocated (nonunique). Also, they cannot have key ranges and they are restricted to a maximum of 16 physical extents per volume. A reusable base cluster may not have alternate indexes; however, reusable alternate indexes can be associated with a nonreusable base cluster.

**1** IGGPRUS checks the data set catalog record to verify that it is a type D record and that the data set has the correct attributes for resetting. Appropriate error codes are returned to the caller if data set does not have the proper attributes.

**2** If the data set is not reusable (but is empty), IGGPRUS calls IGGPLOC before returning to caller. No error codes are returned.

**3** If the data set can be reset, IGGPEXT retrieves the AMDSB set of fields. If the data set attributes in the AMDSB indicate a key-sequenced data set, the user's work area (CTGWKA) is checked to ensure that it contains the control interval number of the index record in the catalog. If not, return to caller with the appropriate error code.

**4** IGGPEXT retrieves each volume information set of fields pertaining to the data set.

**5** IGGPGET retrieves the corresponding volume records from the catalog.

**6** Each volume information set of fields is updated so as to retain only primary extents. If no primary extents remain after updating, the volume becomes a candidate volume. IGGPMOD returns the updated volume information set of fields to the catalog.

**7** If a recoverable catalog is involved, IGGPTNXO calculates for each volume processed the sum of the relative track addresses of extents remaining in the reset data set.

**8** IGGPIXO uses the value from step **7** to update the Data Set Directory set of fields for each volume processed.

**9** If the resetting of the data set results in extents being freed on the volume being processed, IGGPSSCR updates the Space Map set of fields.

**10** IGPMOD updates the catalog with the AMDSB set of fields and the base data set catalog record.

**11** If a key-sequenced data set is being reset, IGGPGET retrieves the index catalog record and resetting of the index section of the data set proceeds as described above for the data component.

**12** If a key-sequenced data set has been reset, IGGPFRWK retrieves the data set catalog record required by IGGPLOC.

**13** IGGPLOC processes the user's LOCATE request.

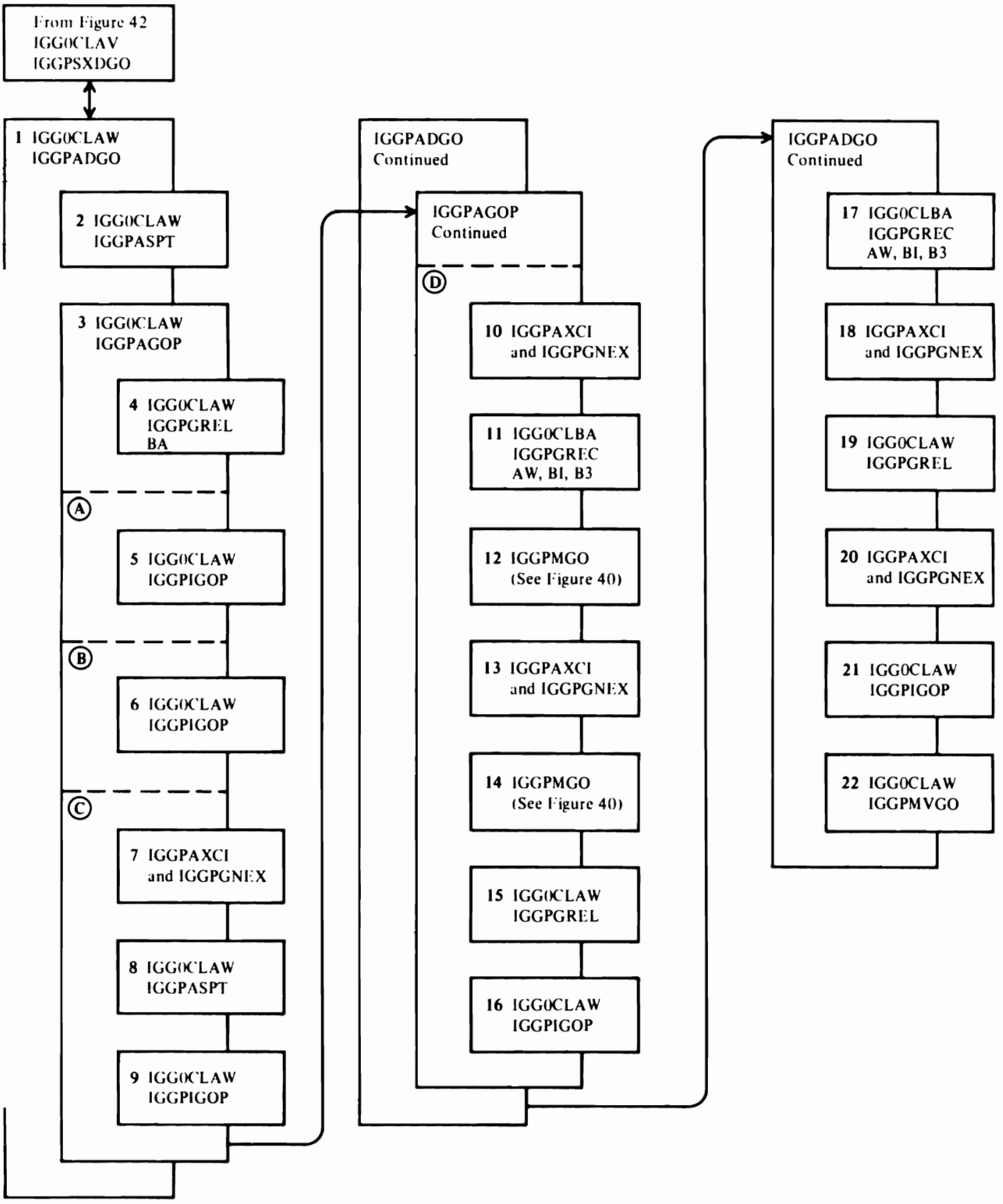



Figure 45. Insert a New Set of Fields (IGGPADGO Processing)

**Notes for Figure 45**

**1** IGGPADGO inserts a new set of fields (group occurrence) into the object's catalog record and adjusts other catalog records as required. In the notes for this figure, the hypothetical set of fields to be added is called ADDSET.

**Note:** If ADDSET can fit in the base catalog record, its set of fields pointer (group occurrence pointer) in the base catalog record contains ADDSET's displacement from the beginning of the base catalog record's sets of fields (group occurrences).

Otherwise, ADDSET is put in an extension record and two set of fields pointers (group occurrence pointers) are used to locate it. Its set of fields pointer in the base catalog record contains the extension record's control interval number. ADDSET's set of fields pointer in the extension record contains its displacement from the beginning of the extension records sets of fields.

**2** If the base catalog record doesn't contain an "available-space pointer" (a pointer to an extension record that contains free space), IGGPASPT builds an "available-space pointer" (a set of fields pointer with sequence number = 0 and type code = 0) and calls IGGPIGOP to insert it into the base catalog record.

**3** IGGPAGOP ensures that there is a set of fields pointer (group occurrence pointer) in the base catalog record for ADDSET and determines where the set of fields can be inserted. If the base catalog record (or one of its extensions) contains a deleted set of fields pointer (with the same type code as ADDSET), IGGPAGOP activates it and assigns it to ADDSET. If not, IGGPAGOP builds a set of fields pointer and inserts it into the base catalog record (or one of its extensions).

**4** IGGPGREL identifies the set of fields pointer (group occurrence pointer) in the base catalog record (or one of its extensions) that has the same type code as ADDSET and either:

- Is marked as deleted, or
- Has the highest sequence number with ADDSET's group code.

BA: IGGPGREC retrieves any horizontal extensions of the base entry record.

IGGPAGOP then ensures that the base catalog record contains a set of fields pointer (group occurrence pointer) that will point to ADDSET:

**A** If IGGPGREL found a set of fields pointer (group occurrence pointer) marked deleted which points to an extension record:

**5** IGGPIGOP activates the set of fields pointer. IGGPAGOP is finished; ADDSET is to be inserted into the extension record pointed to by the set of fields pointer.

**B** If ADDSET and its set of fields pointer (group occurrence pointer) can fit in the base catalog record:

**6** IGGPIGOP builds a set of fields pointer for ADDSET and inserts it into the base catalog record following the set of fields pointer identified by IGGPGREL (see step **10**). IGGPAGOP is finished; ADDSET is to be inserted into the base catalog record.

**C** If the base catalog record cannot contain a new set of fields pointer (group occurrence pointer), even if all sets of fields are moved out of the base catalog record:

**7** IGGPAXCI and IGGPGNEX obtain an extension record for the base catalog catalog record.

If the record area in which the extension record is to be built already contains a record to be written, IGGPGREC flushes the record.

**8** IGGPASPT builds an available space pointer and inserts it into the base catalog record's extension.

**9** IGGPIGOP builds a set of fields pointer for ADDSET and inserts it into the base catalog record's newly obtained extension record. IGGPAGOP is finished; ADDSET is to be inserted into the base catalog record's newly obtained extension.

**D** If the base catalog record can contain a new set of fields pointer (group occurrence pointer) by moving the sets of fields (group occurrences) in the base catalog record into an extension record:

Move the set of fields out of the base catalog record into an extension record and adjust the base catalog record as necessary:

**10** If the base catalog record's available-space pointer doesn't point to an extension record, IGGPAXCI and IGGPGNEX obtain an extension record to contain the sets of fields.

**11** If the base catalog record's available-space pointer points to an extension record, IGGPGREC retrieves the extension record.

AW: IGGPPREC writes any record in the record area that must be flushed before the extension record can be retrieved.

BI: IGGPGET retrieves the extension record.

B3: IGGPSMFG ensures that a copy of the original extension record exists.

**12** IGGPMGO moves *all* sets of fields (except type code 1 or 2) into the extension record from the base catalog record. IGGPMGO adjusts each set of fields pointer in the base catalog record to point to the extension record containing its set of fields.

**13** If the extension record obtained by step **11** isn't able to contain all of the base record's sets of fields, IGGPAXCI and IGGPGNEX obtain another extension record to contain the remaining sets of fields. IGGPAGOP updates the base catalog record's available-space pointer to point to the newly obtained extension record.

**14** IGGPMGO moves the rest of the base catalog record's sets of fields into the extension record (obtained in step **13**).

**15** IGGPGREL re-identifies the set of fields pointer (group occurrence pointer) in the base catalog record for ADDSET's use.

**16** IGGPIGOP activates or builds a new set of fields pointer for ADDSET. IGGPAGOP is finished; the record into which ADDSET is to be inserted is the last extension record obtained by **11** or **13**.

**Notes for Figure 45 Continued**

If ADDSET is not to be added to the base catalog record and if the (deleted, now activated) set of fields pointer (see step **3**) points to an extension extension record:

**17** IGGPGREC retrieves the extension record into which ADDSET is to be inserted. (See Figure 42, steps **7** through **8**, for details.)

**18** If ADDSET and its set of fields pointer cannot fit in the extension record, IGGPAXCI and IGGPGNEX obtain another extension reocrd.

**19** IGGPGREL determines where (in the extension record) ADDSET's set of fields pointer (group occurrence pointer) should be be inserted.

If ADDSET is not to be added to the base catalog record, and if ADDSET's set of fields pointer (in the base catalog record) doesn't point to an extension record:

**20** IGGPAXCI and IGGPGNEX obtain an extension record to contain ADDSET.

**21** IGGPIGOP builds a new set of fields pointer (group occurrence pointer) for ADDSET and inserts it into the extension record that will contain ADDSET.

**22** IGGPMVGO moves ADDSET into the record that is to contain it.

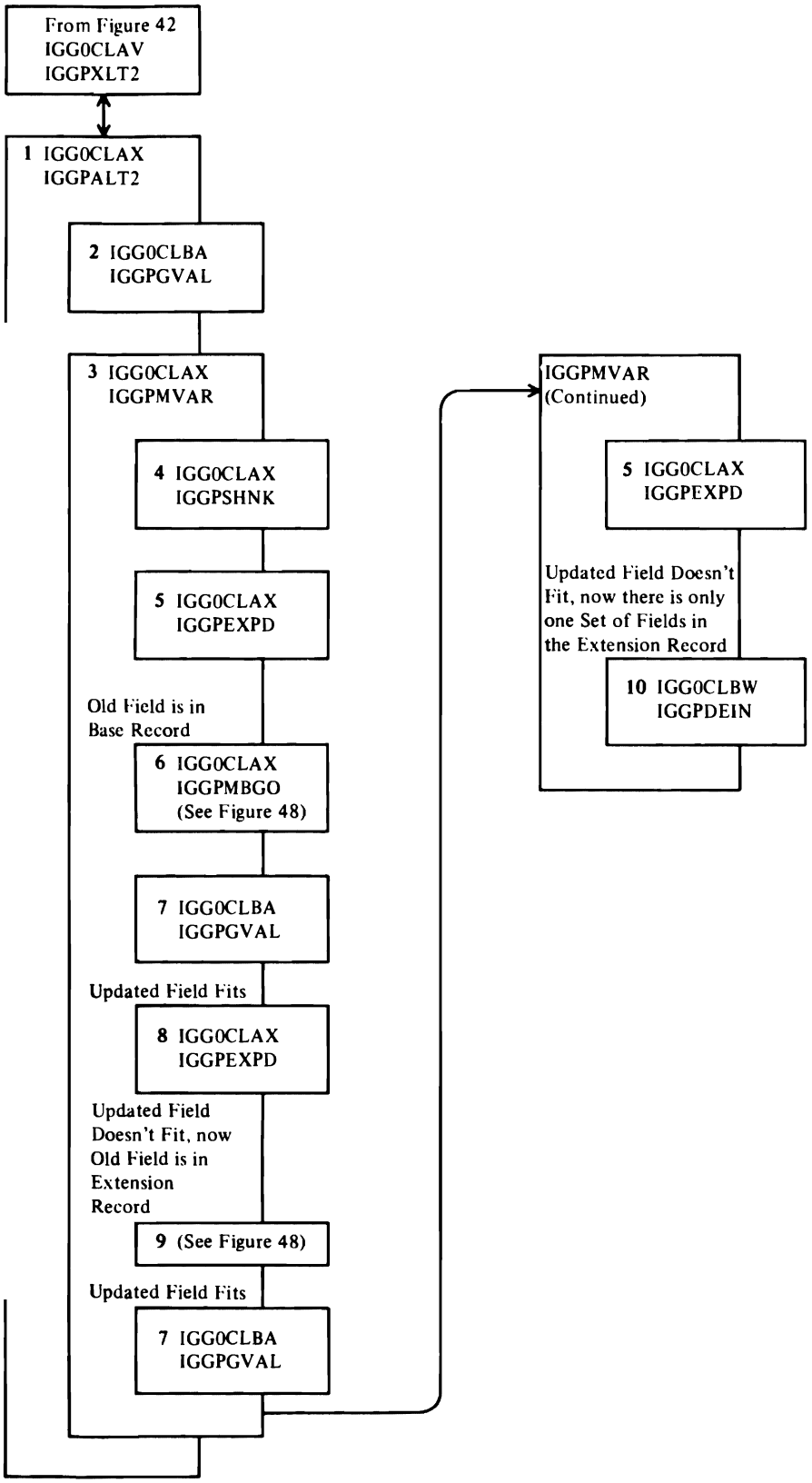



Figure 46. Modify Field Data (IGGPALT2 Processing)

**Notes for Figure 46**

**1** IGGPALT2 modifies the contents of a catalog record. It repeats the following sequence to process each field name identified by a combination name.

**2** IGGPGVAL retrieves the field to be modified (See Figure 42, steps **5** through **10**, for details.)

**3** When the field is fixed-length, IGGPALT2 replaces the field's contents with the caller's update data.

When the field is variable-length with length change, IGGPMVAR replaces the field's contents with the caller's update data and adjusts other catalog records as required.

**Note:** IGGPMVAR can only be used with IGGPMOD callers, since IGGPUPD callers do not modify the contents of variable-length fields.

**4** When the variable-length field's new length is less than or equal to its old length, IGGPSHNK:

1. Adjusts the rest of the record so that all the record's free space is contiguous, and increases the amount of free space.
2. Adjusts other set of fields pointers (group occurrence pointers) to reflect displacement changes resulting from the free space adjustment.
3. Replaces the catalog record field's contents with the caller's update data.

**5** When the variable-length field's new length is greater than its old length, and when the entire new field's contents can be contained in the catalog record, IGGPEXPD:

1. Adjusts the rest of the record so that the larger field is inserted, and decreases the amount of free space.
2. Adjusts other set of fields pointers (group occurrence pointers) to reflect displacement changes resulting from the insertion.
3. Replaces the catalog record field's contents with the caller's update data.

When the entire field's new contents cannot be contained in the catalog record, and that record is the base catalog record:

**6** IGGPMBGO moves the field's set of fields (group occurrence) into an extension record.

**7** IGGPGVAL locates the field in the extension reocrd.

**8** If the entire field's new contents can be contained in the extension record, IGGPEXPD updates the field's contents as described in step **5**.

When the entire field's new contents cannot be contained in the catalog record; when the record is an extension record; and when the extension record contains two or more sets of fields (group occurrences):

**9** Sets of fields (group occurrences) are moved out of the extension record into another extension record (as described in Figure 48) until:

- The entire field's new contents can be contained in its set of fields' extension record, or
- The field's set of fields (group occurrence) is the only set of fields in an extension record.

When the entire field's new contents cannot be contained in the catalog record; when the record is an extension record; and when the extension record contains only the field's (to be updated) set of fields (group occurrence):

**10** IGGPDEIN updates the field's contents and adjusts other catalog records as required.

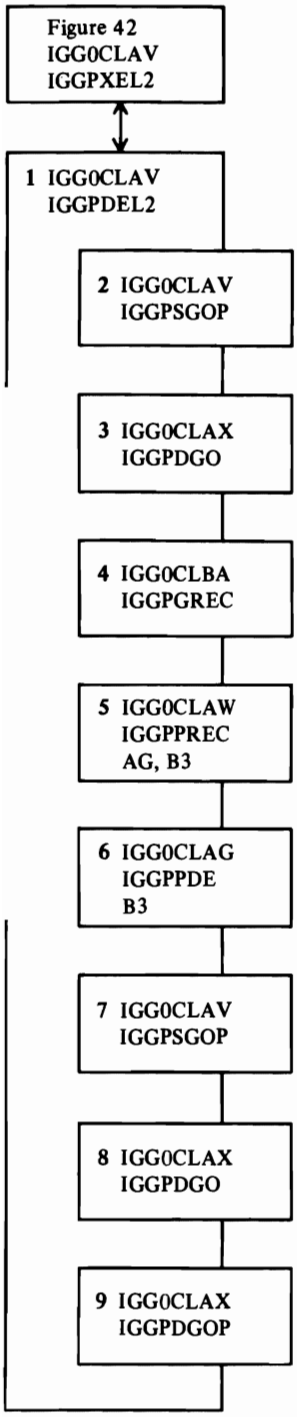



Figure 47. Remove a Set of Fields (IGGPDEL2 Processing)

**Notes for Figure 47**

**1** IGGPDEL2 removes a set of fields (group occurrence) and adjusts other catalog records as required. When IGGPXEL2 calls IGGPDEL2, IGGPSFPL has determined the sequence number of the set of fields pointer (group occurrence pointer) to be deleted. In this figure, the set of fields to be deleted is called DELSET.

**2** IGGPSGOP locates DELSET's set of fields pointer (group occurrence pointer) in the base catalog record or one of its horizontal extensions.

When DELSET is in the base catalog record or one of its horizontal extensions:

**3** IGGPDGO deletes DELSET. IGGPDEL2 marks its set of fields pointer (group occurrence pointer) as deleted and zeros its displacement value. DELSET is now deleted; IGGPDEL2 is finished and returns to IGGPSFPL.

When DELSET is in an extension record:

**4** IGGPGREC retrieves the extension record and as many of its extensions as required to delete DELSET. (See Figure 42, steps **6** through **7**, for details.)

When DELSET is the only set of fields in the extension record:

IGGPDEL2 marks DELSET's set of fields pointer (group occurrence pointer) in the base catalog record as deleted and zeros its pointer to the extension record.

**5** IGGPPREC updates the object's base catalog record. catalog record.

**6** IGGPPDE issues PUT-update to rewrite the extension as a free catalog control interval.

B3: IGGPSMF identifies the copy of the original catalog record (saved by IGGPSMFG) as an updated record.

DELSET is now deleted; IGGPDEL2 processes each additional vertical extension that contains part of the set of fields (using steps **4** through **6**) and, when DELSET is completely deleted, returns to IGGPSFPL.

When DELSET is not the only set of fields (group occurrence) in the extension record:

**7** IGGPSGOP finds DELSET's set of fields pointer (group occurrence pointer) in the extension record.

**8** IGGPDGO deletes DELSET in the extension record.

**9** IGGPDGOP deletes DELSET's set of fields pointer (group occurrence pointer) in the extension record.

IGGPDEL2 marks DELSET's set of fields pointer (group occurrence pointer) in the base catalog record as deleted, and zeros its pointer to the extension record. DELSET is now deleted; IGGPDEL2 is finished and returns to IGGPSFPL.

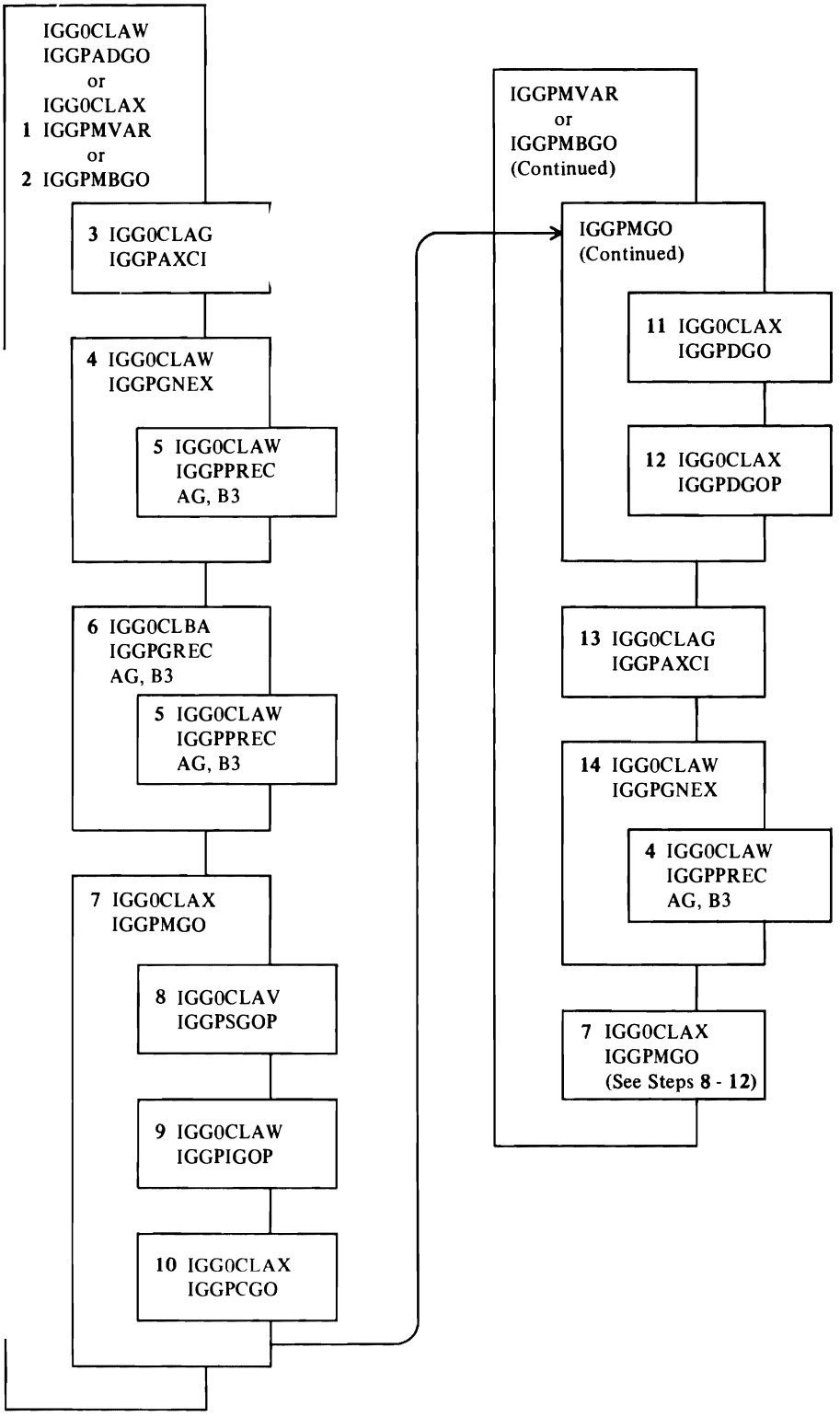



Figure 48. Move a Set of Fields from a Catalog Record into its Extension

**Notes for Figure 48**

This figure describes the sequence required to move a set of fields (group occurrence) from one record (OLDREC) to another (NEWREC). The acronyms "OLDREC" and "NEWREC" are introduced to help you understand this process; these acronyms do not appear in the VSAM code as symbolic names or comments.

**1** IGGPMVAR executes this sequence to move each set of fields (group occurrence), except the one that contains the expanded variable-length field, from one extension record to another. IGGPMVAR moves one set of fields at a time, the record's last set of fields, and returns to its caller.

**2** IGGPMBGO executes this sequence to move the expanding set of fields (group occurrence) in the base catalog record into an extension record.

**3** When the base catalog record's available-space pointer (identified by type code = 0 and sequence number = 0) is zero, IGGPAXCI obtains a free control interval.

**4** IGGPGNEX initializes it as an extension record (NEWREC). IGGPGNEX then updates the available-space pointer to contain NEWREC's control interval number.

**5** If the "buffer must be written" indicator is on, IGGPREC writes the contents of the buffer (a catalog record) prior to reading another record into it.

AG: IGGPPUPC rewrites an updated catalog record.

AG: IGGPPAD writes a new (extension) catalog record.

B3: IGGPSMF identifies the copy of the original catalog record (saved by IGGPSMFG) as an updated record.

**6** When the base catalog record's available-space pointer points to an extension record with free space, IGGPGREC retrieves the extension record (NEWREC).

AG: IGGPGET issues GET to retrieve the catalog record.

B3: IGGPSMFG makes a copy of the catalog record in case it is updated later.

**7** When there is enough free space in NEWREC to contain OLDREC's set of fields (group occurrence), IGGPMGO moves the set of fields (group occurrence) from OLDREC to NEWREC.

**8** IGGPSGOP searches NEWREC to locate the position of the new set of fields' pointer (group occurrence pointer).

**9** IGGPIGOP inserts a new set of fields pointer (group occurrence pointer) in NEWREC to contain the displacement of the new set of fields.

**10** IGGPCGO copies the contents of the set of fields (group occurrence) into NEWREC and reduces the amount of NEWREC's free space.

**11** IGGPDGO deletes the set of fields (group occurrence) in OLDREC and increases the amount of OLDREC's free space.

**12** If OLDREC is not the base catalog record, IGGPDGOP deletes the set of fields pointer (group occurrence pointer) in OLDREC.

**13** If the caller in step **1** or **2** determines that NEWREC (determined at steps **3** or **5**) cannot contain the set of fields (group occurrence), another extension record is built. IGGPAXCI obtains a free control interval and IGGPGNEX initializes it as an extension record (NEWREC). The base catalog record's available-space pointer is updated.

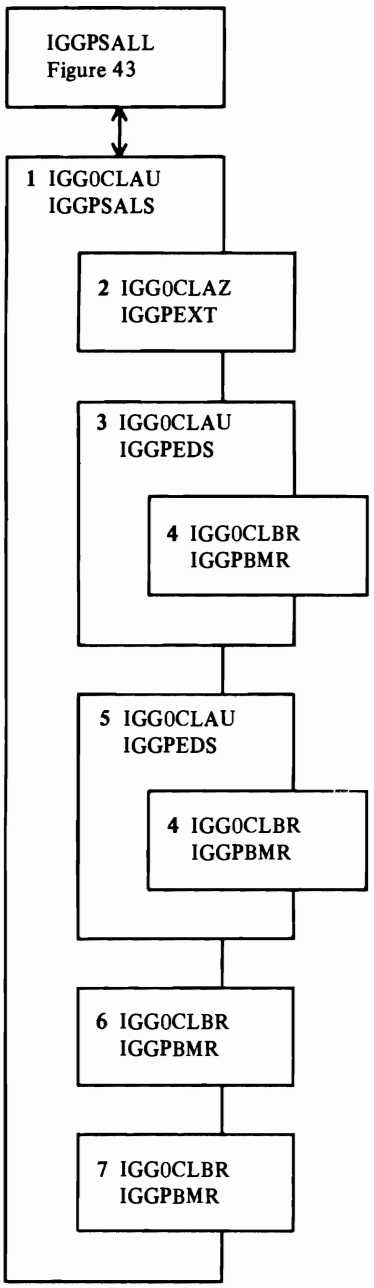



Figure 49. Allocating Part of a Data Space's Space (IGGPSALS Processing)

**Notes for Figure 49**

1 IGGPSALS assigns tracks to a VSAM object that can reside in a nonunique data space (a data space that can contain more than one VSAM object).

2 IGGPEXT retrieves data space group sets of fields from the volume catalog record. Each Data Space Group set of fields contains one or more (up to 16) extent descriptors that describe the data space's extents. IGGPSALS builds a table that contains extent descriptors in the form:

S#TTNN

where

S# is the sequence number of the extent descriptor in its data space group set of fields.

TT is the extent's starting track number, and is converted by another routine to CCHH for a seek address.

NN is the number of tracks in the extent.

When the extent table is full, or when there are no more data space group sets of fields to process, IGGPSALS calls IGGPEDS to process each entry in the table.

3 IGGPEDS scans the data space extent table to find the entry with the smallest TT value. This entry is processed, then IGGPEDS scans the table again to find the entry with the next higher TT value. All entries in the table are found, then processed, from the lowest TT value to the highest.

4 IGGPBMR scans the space map set of fields, starting at bit position TT and ending at bit position TT+NN-1, attempting to find a contiguous amount of unallocated tracks large enough to satisfy the minimum allocation unit for the request (usually a control interval).

IGGPBMR returns to IGGPEDS with either a "no extent found" indicator or a TTNN value. IGGPEDS analyzes the TTNN value to determine:

1. If the extent exactly satisfies the caller's allocation request, no further extent table processing is done.
2. If the extent is larger than the caller's allocation request, but is smaller than any previously obtained extent, the (smaller) extent's TTNN and its data space's sequence number is saved. Processing the data space extent table continues.
3. If the extent is smaller than the caller's allocation request, its TTNN and data space sequence number is put in the "small extent table" if:
   - There are fewer than the maximum number of entries in the small extent table (five, or a caller-specified maximum less than five), or
   - The extent's NN value is larger than the smallest NN value in the small extent table. Processing the data space extent table continues.

5 When the data space extent table is partially full, IGGPEDS processes each entry as described in steps **3** and **4**.

6 IGGPBMR adjusts the bits in the space map set of fields if the caller's allocation request is satisfied with one extent.

7 If the caller's allocation request is satisfied with more than one extent, all entries in the small extent table are sorted on decreasing NN value, so that space is allocated from the least number of extents. IGGPBMR adjusts the bits in the space map set of fields for each extent required to exactly satisfy the caller's request. Each bit in the space map set of fields identifies a track on the volume as either allocated to a VSAM object or unallocated.

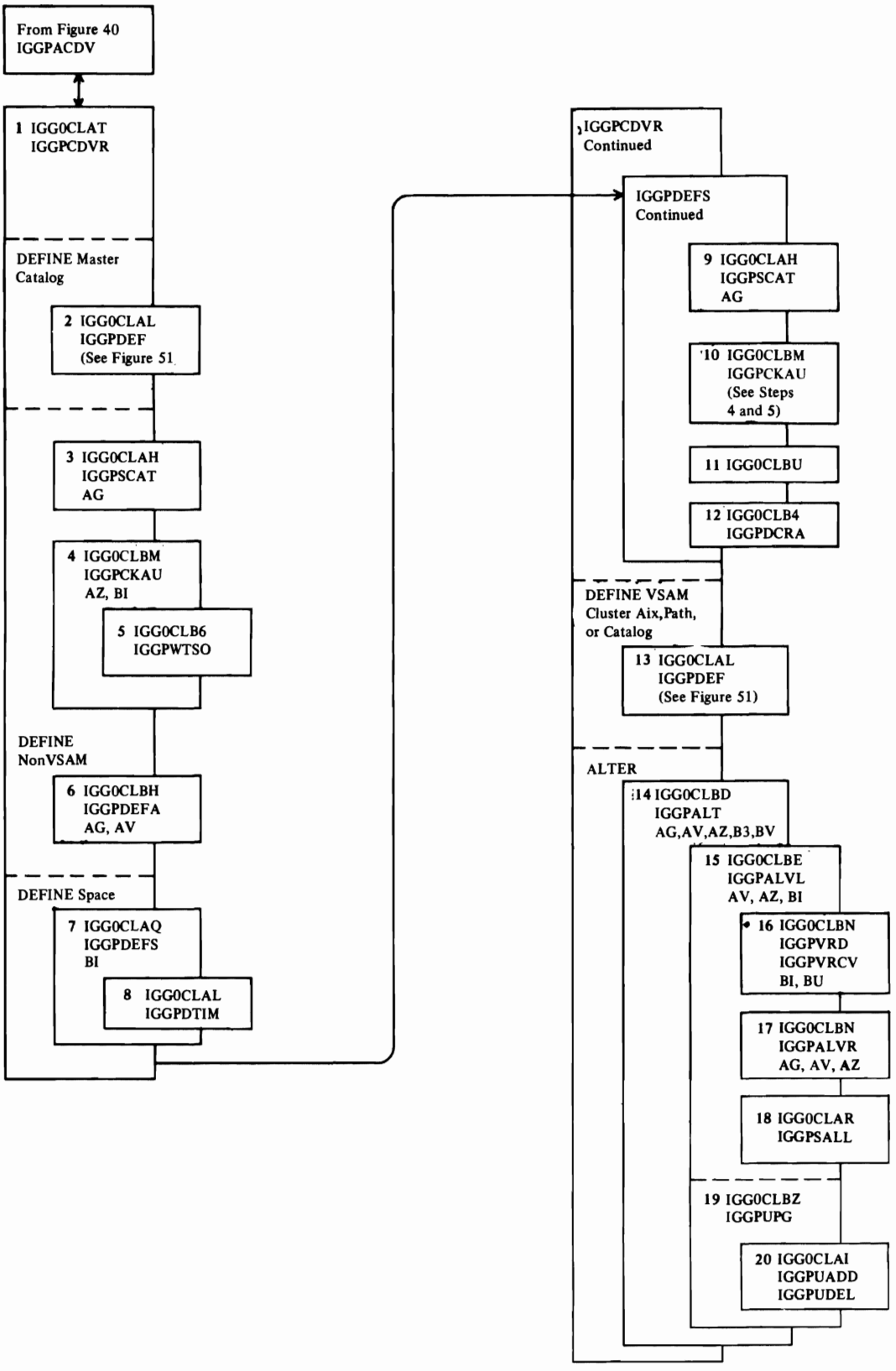



Figure 50 (Part 1 of 2). VSAM Catalog Management Services Processing

**Notes for Figure 50**

1 IGGPCDVR is the VSAM Catalog Management Services: Common Processing procedure.

2 If the request is to create the VSAM master catalog, IGGPDEF processes it.

3 IGGPSCAT searches the catalog for a duplicate name (if the request is DEFINE), or to retrieve the object's catalog record (if the request is ALTER, DELETE, or LISTCAT). If the request is DEFINE SPACE, step 3 is not performed.

4 IGGPCKAU verifies the caller's authorization to perform the request. If the request is LISTCAT, step 4 is bypassed.

AZ: IGGPEXT locates the password information required by IGGPCKAU.

BI: IGGPGET issues GET to retrieve the catalog's cluster catalog record (control interval number 2) to determine if the catalog is password protected. If the catalog is not password protected, all security verification processing is bypassed.

5 When the user is on a TSO terminal, IGGPWTSO issues requests to the TSO terminal for the required password if the password is not supplied with the input parameters (part of the Access Method Services job syntax language).

6 When the caller's request is DEFINE NONVSAM, IGGPDEFA processes it.

AG: IGGPAOCI obtains a free control interval to contain the nonVSAM catalog record.

AV: IGGPMOD inserts information into the newly created nonVSAM catalog record.

BV: IGGPSMFA writes SMF record type 63—VSAM Data Set Cataloged.

7 When the caller's request is DEFINE SPACE, IGGPDEFS processes it.

AG: IGGPPAD issues PUT-Add to add records to the catalog as required.

BI: IGGPGET issues GET to retrieve the volume catalog record.

8 IGGPDTIM obtains time-of-day data.

9 IGGPSCAT retrieves the the catalog record identified by the CTGPL (the VSAM object's base catalog record). Extensions to the base catalog record are retrieved as they are needed.

BI: IGGPGET issues GET to retrieve catalog records.

10 IGGPCKAU verifies the caller's authorization to create a VSAM data space.

AZ: IGGPEXT locates the password information required by IGGPCKAU.

BI: IGGPGET issues GET to retrieve the volume-owner's (a VSAM catalog) catalog record that contains the password set of fields (group occurrence).

11 IGG0CLBU processes the format-4 DSCB, when the DEFINE SPACE creates the volume's first data space:

IGGPF4RD reads the format-4 DSCB.

IGGPF4DQ dequeues the volume identified by the format-4 DSCB.

IGGPF4WR writes (or updates, if it exists) the format-4 DSCB.

12 IGGPDCRA defines a catalog recovery area on the volume when the volume's first data space is created and the owning catalog is recoverable.

13 When the caller's request is to create a VSAM cluster or a VSAM user's catalog, IGGPDEF processes it.

14 When the caller's request is ALTER, IGGPALT processes processes it.

AG: IGGPPUPC issues PUT-update to rewrite a catalog catalog record.

AG: IGGPPAD issues PUT-Add to insert a catalog record into the VSAM catalog.

AG: IGGPPDE issues PUT-update to rewrite the record as a free catalog catalog record.

AV: IGGPMOD modifies the contents of catalog record fields.

AZ: IGGPEXT locates catalog record fields.

BI: IGGPGET issues GET to retrieve a catalog record.

BV: IGGPSMFA writes SMF record type 63.

15 IGGPALVL modifies volume catalog record information.

AV: IGGPMOD modifies the contents of catalog record fields.

AZ: IGGPEXT locates catalog record fields.

BI: IGGPGET issues GET to retrieve the volume catalog record.

16 IGGPVRD mounts the required volume(s) and IGGPVRCV removes VSAM ownership from the volume(s).

BI: IGGPGET retrieves the volume catalog record.

BU: IGGPF4RD reads the format-4 DSCB.

BU: IGGPF4DQ dequeues the volume identified by the format-4 DSCB.

BU: IGGPF4WR writes (updates) the format-4 DSCB.

17 IGGPALVR updates the volume catalog record.

AG: IGGPPUPC rewrites the updated volume catalog catalog record.

AV: IGGPMOD modifies fields in the volume catalog record.

AZ: IGGPEXT locates fields in the volume catalog record.

BI: IGGPGET retrieves the volume catalog record.

18 IGGPSALL assigns a specified volume as a candidate volume to a VSAM object.

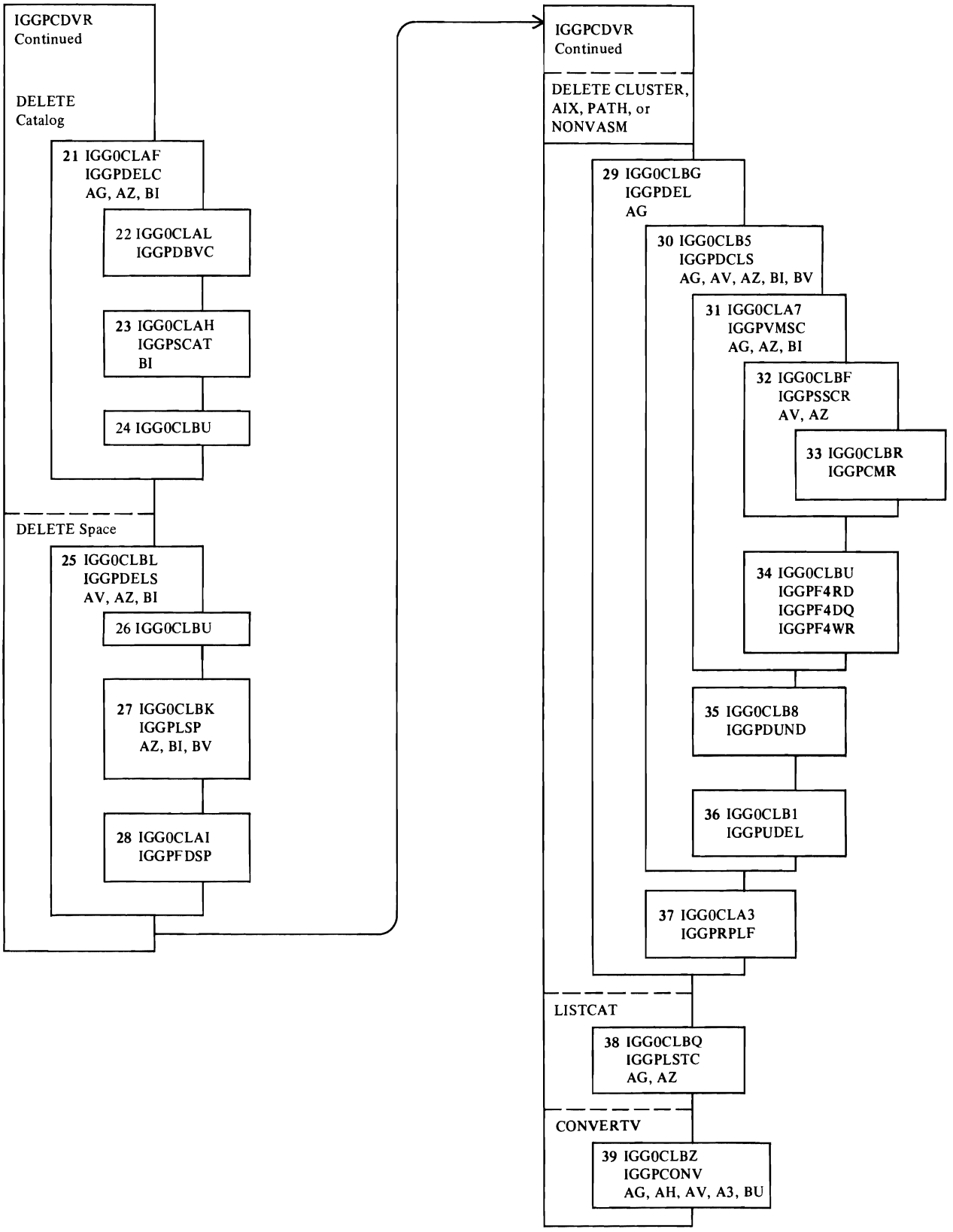



Figure 50 (Part 2 of 2). VSAM Catalog Management Services Processing

**Notes for Figure 50 Continued**

**21** When the caller's request is DELETE CATALOG, IGGPDELC processes it.

AG: IGGPPDE issues PUT-update to rewrite the record as a free catalog catalog record.

AG: IGGPCCCR checkpoints the catalog record before it is deleted.

AG: IGGPRPLF dequeues the catalog (releases exclusive control over the catalog so that other requests can process its records).

AZ: IGGPEXT locates catalog record fields.

BI: IGGPGET issues GET to retrieve the catalog records.

**22** IGGPDBVC verifies that the caller's work area (to return the name of the catalog in) is in the caller's address space.

**23** IGGPSCAT searches the catalog being deleted to verify that the catalog is empty (contains no cataloged objects).

BI: IGGPGET issues GET to retrieve the catalog records that describe the catalog.

**24** IGG0CLBU processes the format-4 DSCB:

IGGPF4RD reads the format-4 DSCB.

IGGPF4WR writes the updated format-4 DSCB.

**25** When the caller's request is DELETE SPACE, IGGPDELS processes it.

AG: IGGPPDEC issues PUT-update to rewrite an extension record as a free catalog record.

AV: IGGPMOD modifies the contents of the volume catalog record's fields.

AZ: IGGPEXT locates a catalog record field.

BI: IGGPGET issues GET to retrieve the volume catalog record.

**26** If the volume is completely empty, IGG0CLBU processes the format-4 DSCB:

IGGPF4RD reads the format-4 DSCB.

IGGPF4WR writes the updated format-4 DSCB.

**27** When the volume catalog record is not deleted, IGGPLSP determines the amount of available space there is on the volume.

AZ: IGGPEXT locates each shared (nonunique) data space group set of fields (group occurrence) in the volume catalog record.

BI: IGGPGET issues GET to retrieve the volume catalog record.

BV: IGGPSMFL writes SMF record type 69—VSAM Data Space Defined or Deleted.

**28** IGGPFDSP scratches all VSAM data space and makes it available to the OS/VS system.

**29** When the caller's request is DELETE CLUSTER, ALTERNATE INDEX, PATH, PAGESPACE, or NONVSAM, IGGPDEL processes it.

AZ: IGGPEXT locates a catalog record's field.

BI: IGGPGET issues GET to retrieve a catalog record.

**30** IGG0CLB5 contains the following DELETE procedures that are called by IGGPDEL:

IGGPDCLS deletes data, index, cluster, and alternate index catalog records.

IGGPDEAX explicitly deletes an alternate index.

IGGPDIAX implicitly deletes one or more alternate indexes.

IGGPDEPT explicitly deletes a path.

IGGPDIPT implicitly deletes one or more paths.

IGGPDUPG deletes upgrade association set of fields.

Each of the IGG0CLB5's procedures call the following procedures as needed:

AG: IGGPPDE issues PUT-delete to delete a catalog record.

AV: IGGPMOD modifies the contents of catalog record fields.

AZ: IGGPEXT locates a catalog record field.

BI: IGGPGET issues GET to read a catalog record.

BV: IGGPSMFS writes SMF record type 67 to the SMF data set.

**31** IGG0CLA7 contains the following DELETE procedures that are called by IGGPDEL:

IGGPVMSC deletes all space information for the object in the volume catalog record.

IGGPDEMV locates the object's volume information sets of fields (group occurrences).

IGGPDVMV ensures that all required volumes are mounted.

IGGPDUSC returns the data space (issues SCRATCH to delete the format-1 identifier DSCB) from the volume's VTOC.

IGGPMCRA verifies that the primary CRA volume is either mounted or mountable.

IGGPDF4T updates the format-4 DSCB and the volume catalog record timestamp.

Each of IGG0CLA7's procedures call the following procedures as needed:

AG: IGGPPUPC issues PUT-update to rewrite an updated catalog record.

AV: IGGPMOD modifies the data space group set of fields (group occurrences).

AZ: IGGPEXT locates a catalog record field.

BI: IGGPGET issues GET to read a catalog record.

**Notes for Figure 50 Continued**

**32** IGGPSSCR returns the space allocated to a catalog or cluster that is contained in a shared (nonunique) data space.

AZ: IGGPEXT locates the data space group set of fields (group occurrence) that describes the data space.

AV: IGGPMOD modifies the data space group set of fields (group occurrence).

**33** IGGPBMR adjusts the space map set of fields to show the newly allocated tracks.

**34** IGG0CLBU contains the following external procedures used by the DELETE procedures:

IGGPF4RD reads a format-4 DSCB from the VTOC.

IGGPF4DQ issues the VTOC DEQ macro.

IGGPF4WR writes the format-4 DSCB in the VTOC.

**35** IGGPGUND cleans up extension records.

AG: IGGPPDEC issues PUT-update to rewrite an extension record as a free catalog record.

**36** IGG0CLB1 contains the following external procedure used by DELETE procedures:

IGGPUDEL deletes upgrade association sets of fields from the Y catalog record.

**37** IGG0CLA3 contains the following external procedures used by DELETE procedures:

IGGPRPLF releases the serialability of the catalog resource after performing the erase process.

IGGPRPLM acquires the serialability of the catalog resource after performing the erase process.

**38** When the caller's request is LISTCAT, IGGPLSTC processes it. IGGPCKAU is called each time a record is retrieved, and verifies the user's authorization to retrieve the record.

AZ: IGGPEXT locates a catalog record field.

BI: IGGPGET issues GET to retrieve the catalog record.

**39** When the caller's request is CONVERTV, IGGPCONV processes it.

AG: IGGPPUPC writes the catalog record.

AG: IGGPRCCR updates the catalog control record to indicate the next free control interval.

AH: IGGPSCAT searches the master catalog for the user catalog entry.

AV: IGGPMOD updates device type fields in data and index records in the catalog.

A3: IGGPRPLF releases the master catalog from exclusive control.

BU: IGGPF4RD reads the format-4 DSCB.

BZ: IGGPGALO gets the catalog record whose volume information is to be updated.

BZ: IGGPVALI checks the validity of the CTGPL and CTGFLs.

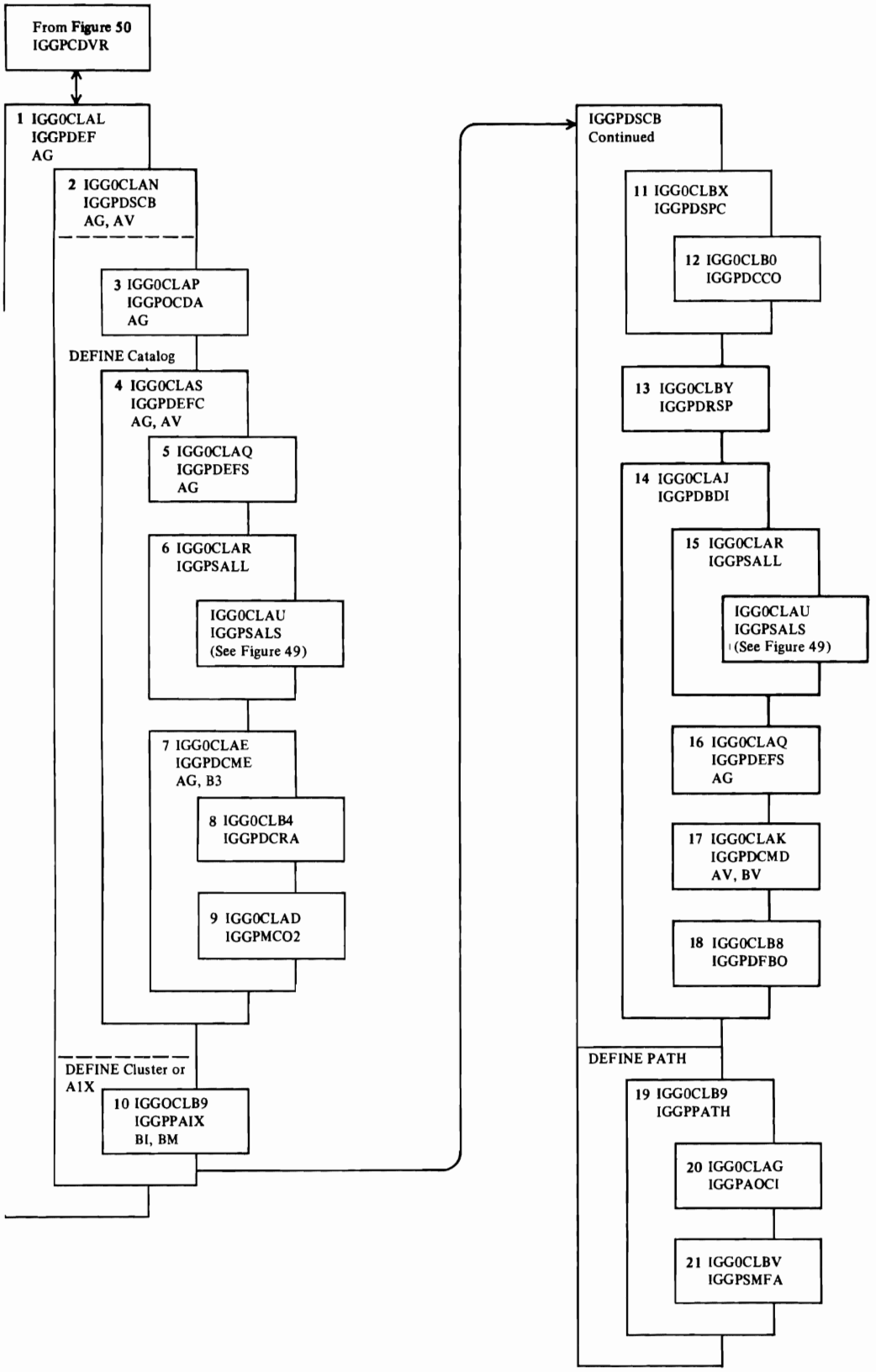



Figure 51. DEFINE Processing

**Notes for Figure 51**

**1** IGGPDEF verifies user-provided values, propagates certain parameters, and generates a dsname for the data set and index of a cluster, as necessary.

**2** IGGPDSCB is a continuation of IGGPDEF.

**3** IGGPDCDA checks the validity of user-supplied parameters and determines the default values of other required parameters.

**4** IGGPDEFC defines a VSAM master or user catalog.

**5** IGGPDEFS builds a data space. The data space is nonunique (contains parts of more than one VSAM object).

**6** IGGPSALL and IGGPSALS assigns part of the data space's tracks to the catalog—a key-sequenced, key-range data set.

**7** IGGPDCOC builds a temporary ACB and temporarily opens the catalog so that the catalog's self-describing records (control interval numbers 0 through 13) can be written.

**8** IGGPDCRA builds a catalog recovery area if the catalog is recoverable.

**9** IGGPMCO2 opens the VSAM master catalog, if it is being defined, after its self-describing records have been written.

**10** IGGPPAIX checks the validity of the user-supplied input parameters and the password for the alternate index.

**11** IGGPDSPC determines the physical requirements of the cluster—its control interval size, blocksize, and number of blocks per track—based on the device that will contain the cluster.

**12** IGGPDCCO checks data and index device characteristics.

**13** IGGPDRSP determines the cluster's space allocation quantities.

**14** IGGPDBDI partially builds the catalog records required for the data set and index and obtains the primary space allocation for the data set and index.

**15** IGGPSALL assigns candidate volumes to the data set and index and, if they reside in a nonunique data space (contains more than one VSAM object), IGGPSALS allocates space from the data space.

**16** IGGPDEFS obtains a unique data space for the data set or index. The data space can contain only one VSAM object.

**17** IGGPDCMD completes the catalog record construction.

AV: IGGPMOD modifies the contents of catalog record fields.

**18** If an error occurred during the cluster's creation, IGGPDFBO resets any allocated tracks to an unallocated status and rewrites any partially built catalog records as free catalog records.

**19** IGGPPATH verifies user-supplied input parameters and the password for the path.

**20** IGGPAOCI obtains one catalog control interval to contain the path record.

**21** IGGPSMFA writes SMF record type 63 after a VSAM path is successfully defined.

# Catalog Management I/O Functions

This section contains a detailed explanation of the catalog management I/O procedures.

When a catalog record is retrieved, updated (rewritten), written (added to the catalog), or deleted, one of the following catalog management procedures initiates the I/O operation:

- IGGPGET—Retrieves the catalog record
- IGGPPUPC—Updates (rewrites) a catalog record
- IGGPPAD—Adds a record to the catalog
- IGGPPDE—Deletes a catalog record
- IGGPRAG—Retrieves a CRA record
- IGGPRAPU—Updates a CRA record
- IGGPRAPA—Adds a CRA record
- IGGPRAPD—Deletes a CRA record

Before a new catalog record can be written into the catalog, a catalog's control interval is assigned to contain the new record's information. This assignment is made by:

- IGGPAOCI—assigns more than one contiguous (if possible) control intervals to the caller. This function is usually called during a DEFINE procedure.
- IGGPAXCI—assigns one control interval to the caller. This function is usually called when an extension record is being built.

Other catalog management I/O procedures are called by the above-mentioned procedures to perform special functions.

All catalog and catalog recovery area I/O functions are described in Figures 51.1–51.19.

## *IGGPGET—catalog record retrieval*

- Retrieves the requested control interval and places it into the caller-specified record area.
- Initializes the RAB (one of six record area blocks in the CCA) and its record segment pointers.

**Normal keyed GET**

1. IGGPXIO retrieves the true name record, using keyed direct GET.
2. IGGPGET converts the catalog record's control interval number (in the true name catalog record) to an RBA.
3. IGGPXIO retrieves the catalog record's control interval, using addressed direct GET.

**Normal keyed GET-next**

– Same as "normal keyed GET", except that "key greater than or equal" is specified when IGGPXIO retrieves the true name catalog record.

**Normal GET by control interval number**

1. IGGPGET converts the catalog record's control interval number (specified by the caller) to an RBA.
2. IGGPXIO retrieves the catalog record's control interval, using addressed direct GET.

**GET requests issued while a catalog is being opened or created**

– When the catalog is being created, IGGPGET "retrieves" the record by locating the DEFINE work area that contains the requested record.

– When the catalog is being opened, IGGPGET issues EXCP and WAIT to retrieve the requested record.

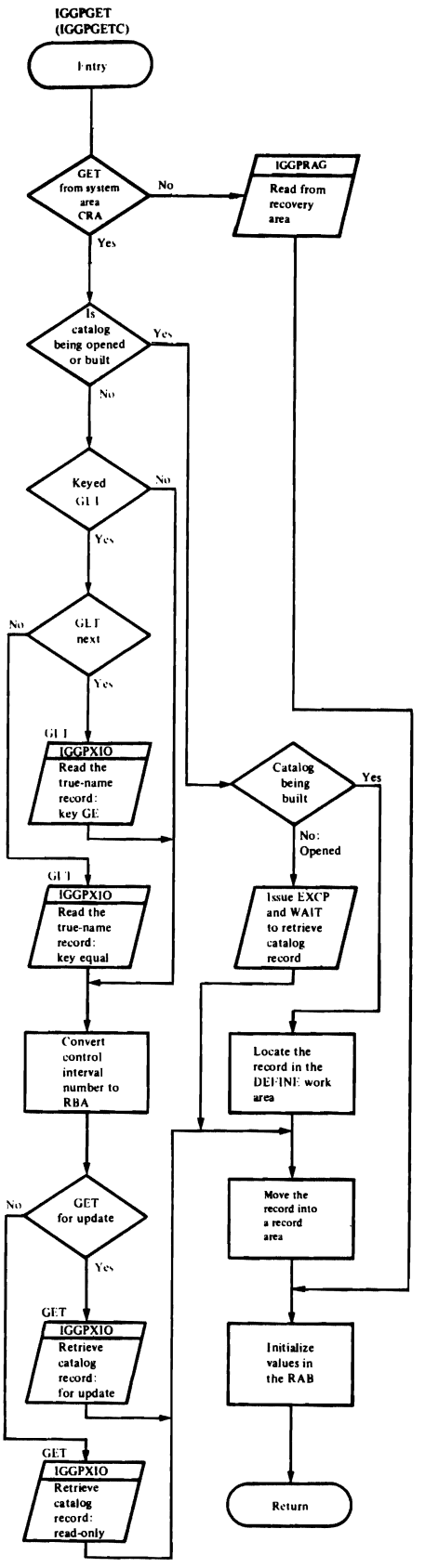



Figure 51.1. Retrieve a Catalog Record (IGGPGET)

## *IGGPPUPC—catalog record write (update)*

1. If the RPL used to retrieve the updated record has been reused, IGGPTRPL calls IGGPXIO to retrieve the catalog record again. This verifies the catalog record's position in the catalog and re-establishes the RPL's GET-for-update status.
2. IGGPXIO writes the catalog record into the catalog, using addressed direct PUT-for-update.

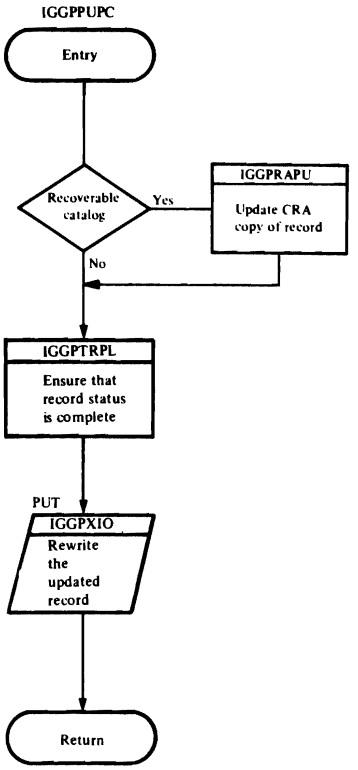



Figure 51.2. Write (Update) a Catalog Record (IGGPPUPC)

### *IGGPPAD—catalog record write (add)*

1. IGGPXIO retrieves the free control interval that will be replaced by the new record, to verify the control interval's position in the catalog and to re-establish the RPL's GET-for-update status.
2. IGGPXIO writes the new catalog record into the catalog, using addressed direct PUT-for-update.
3. If a true name catalog record is required, IGGPPAD builds it, then calls IGGPXIO to write it into the catalog using keyed direct PUT.

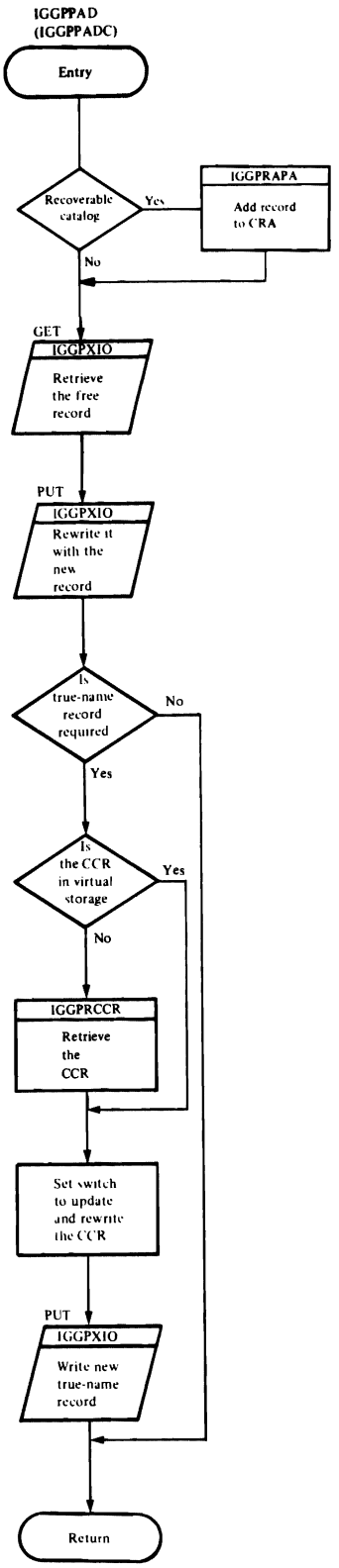



**Figure 51.3. Write (Add) a Catalog Record (IGGPPAD)**

## *IGGPPDE—catalog record deletion*

1. If a true name catalog record exists for the record being deleted:
   - IGGPXIO retrieves the true name catalog record, using keyed direct GET, to establish the record's GET-for-update status.
   - IGGPXIO erases the true name record, using keyed ERASE.
2. If the catalog control record (CCR) is not already in a catalog management record area, IGGPRCCR retrieves it.
3. If the RPL used to retrieve the record (to be deleted) has been reused, IGGPTRPL calls IGGPXIO to retrieve the catalog record again to verify the record's position in the catalog and to re-establish the RPL's GET-for-update status.
4. IGGPPDE builds a free catalog record. Its control interval number is the same as the record to be deleted. The control interval number of the free catalog record is put into the catalog's free-control-interval chain.
5. IGGPXIO deletes the record by replacing it with the free catalog record, using addressed direct PUT-for-update.
6. IGGPCCCR updates and rewrites the catalog control record (CCR).

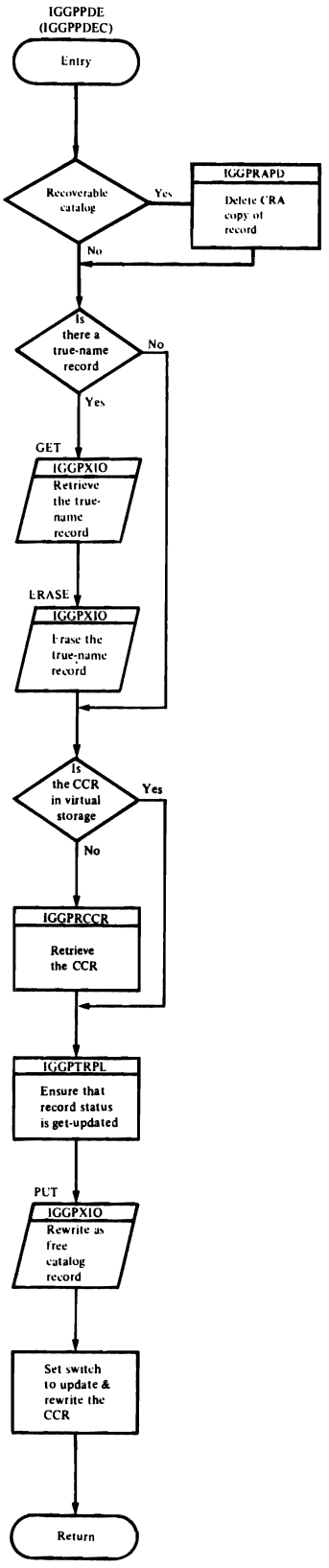



Figure 51.4. Delete a Catalog Record (IGGPPDE)

### *IGGPAOCI—assigns contiguous (if possible) catalog control intervals to the caller*

1. If the catalog control record (CCR) is not already in a catalog management record area, IGGPRCCR retrieves it.
2. IGGPAOCI assigns the requested number of catalog control intervals to the caller. The control intervals will be used to contain new catalog records (built during a DEFINE process). One of the following methods is used (in the order listed) to obtain the control intervals:
   A. If enough previously unassigned control intervals are available in the catalog's extent, IGGPANCI preformats and assigns the requested number of control intervals to the caller.
   B. If enough free control intervals (not necessarily contiguous) are available, IGGPAOCI removes the requested number of control intervals from the free control interval chain and assigns them to the caller.
   C. IGGPANCI obtains more space (via Catalog Extend) for the catalog, then preformats and assigns the requested number of control intervals to the caller.
3. IGGPCCCR updates and rewrites the catalog control record (CCR).

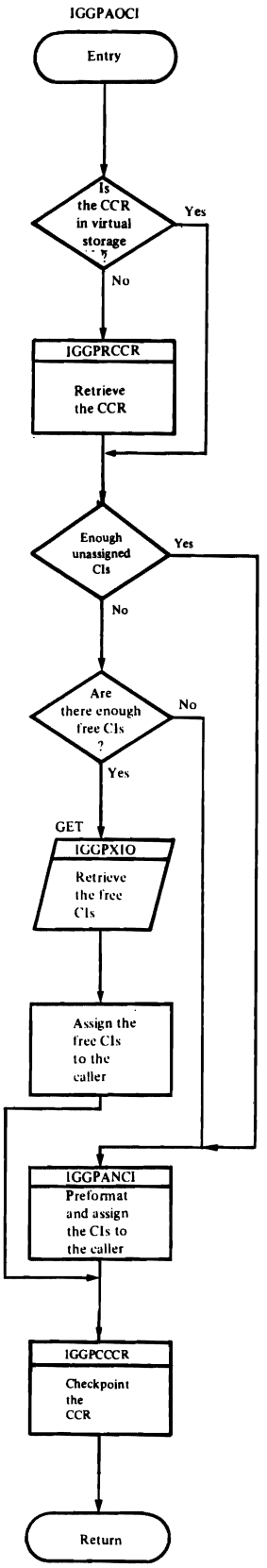



Figure 51.5. Assign Catalog Control Intervals to the Caller (IGGPAOCI)

### *IGGPAXCI—assigns one catalog control interval to be used as an extension record*

1. If the catalog control record (CCR) is not already in a catalog management record area, IGGPRCCR retrieves it.
2. IGGPAXCI assigns one control interval to the caller. One of the following methods is used (in the order listed) to obtain the control interval:
   A. If a free control interval is available, IGGPAXCI removes it from the free control interval chain and assigns it to the caller.
   B. IGGPANCI preformats and assigns a previously unassigned control interval to the caller. This might mean that IGGPANCI first obtains more space for the catalog (via Catalog Extend).
3. IGGPCCCR updates and rewrites the catalog control record (CCR).

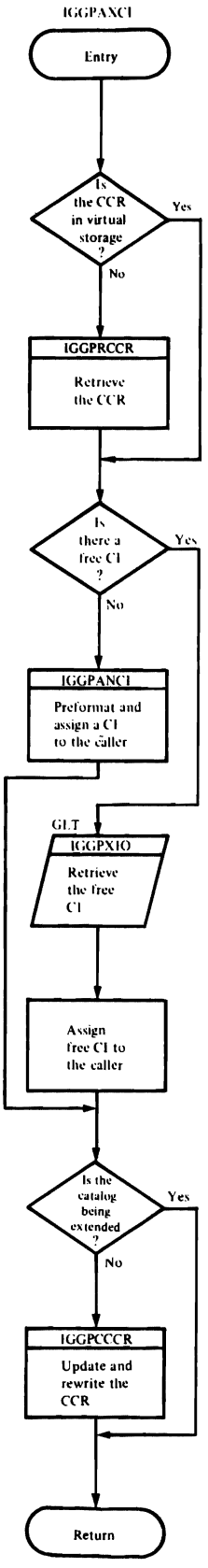



Figure 51.6. Assign a Catalog Control Interval for an Extension Record (IGGPAXCI)

### *IGGPCCCR—updates and rewrites the catalog control record (CCR)*

1. The CCR is read to establish its set-for-update status.
2. IGGPCCCR updates the CCR control fields to reflect the current catalog record usage.
3. IGGPCCCR calls IGGPXIO to rewrite the CCR, using addressed PUT-for-update.

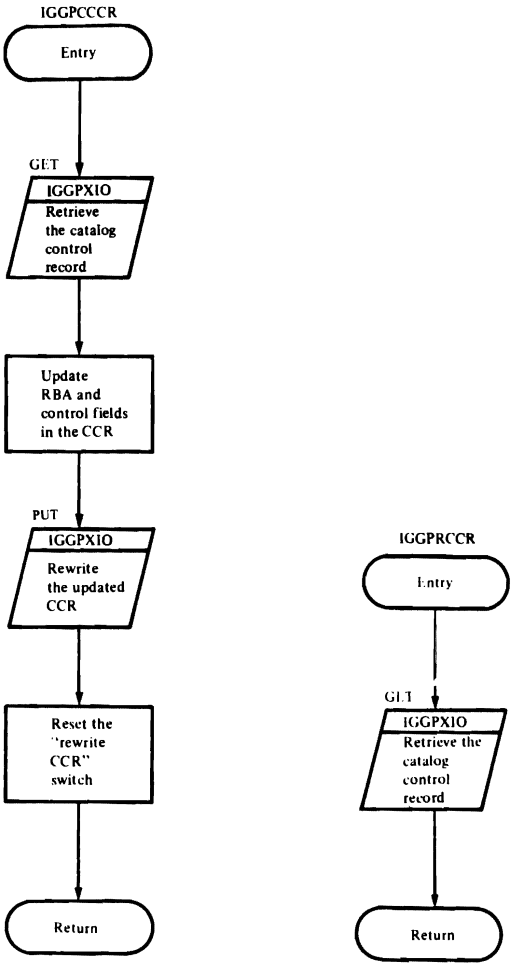



Figure 51.7. Update and Rewrite the CCR (IGGPCCCR)

## *IGGPXIO—calls VSAM record management*

1. IGGPXIO initializes an RPL.
2. IGGPXIO issues GET, PUT, or ERASE—a VSAM record management request macro instruction.
3. If an error occurs, IGGPXIO exits on error to IGGPIORA to convert the RPL error code to an appropriate catalog management error code. IGGPIORA then returns to the routine that called the catalog I/O function that was processing when the error occurred.

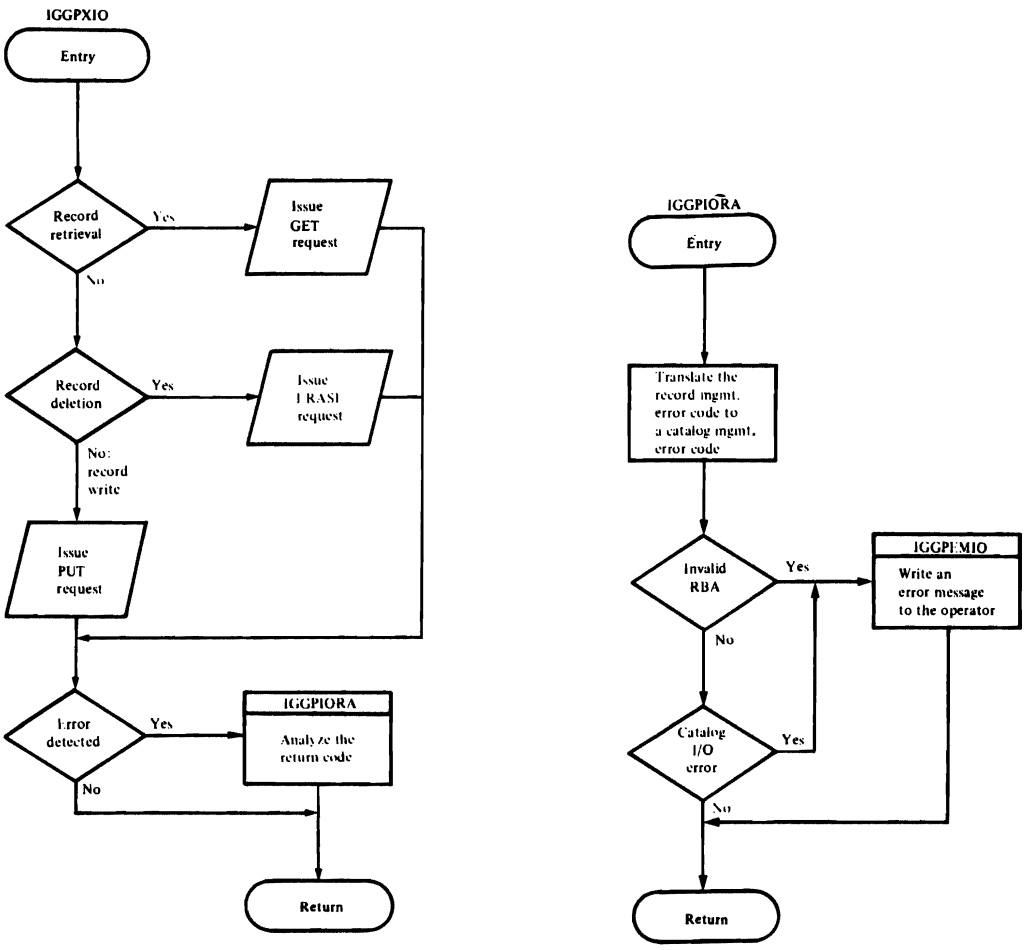



Figure 51.8. Call VSAM Record Management for Catalog Request (IGGPXIO)

***IGGPISCI—ensures that there are enough free control intervals, so that the catalog won't be extended (obtain more space) at the same time a volume catalog record is being processed***

1. If the catalog control record (CCR) is not already in a catalog management record area, IGGPRCCR retrieves it.
2. If there are not sufficient free control intervals available, IGGPISCI calls IGGPANCI to obtain more space for the catalog (via Catalog Extend). IGGPANCI then preformats and assigns a previously unassigned control interval to IGGPISCI.
3. IGGPXIO retrieves the control interval to extablish its GET-for-update status, using addressed GET-for-update.
4. IGGPXIO rewrites the control interval as a free control interval, using addressed PUT-update.
5. IGGPCCCR updates and rewrites the catalog control record (CCR).

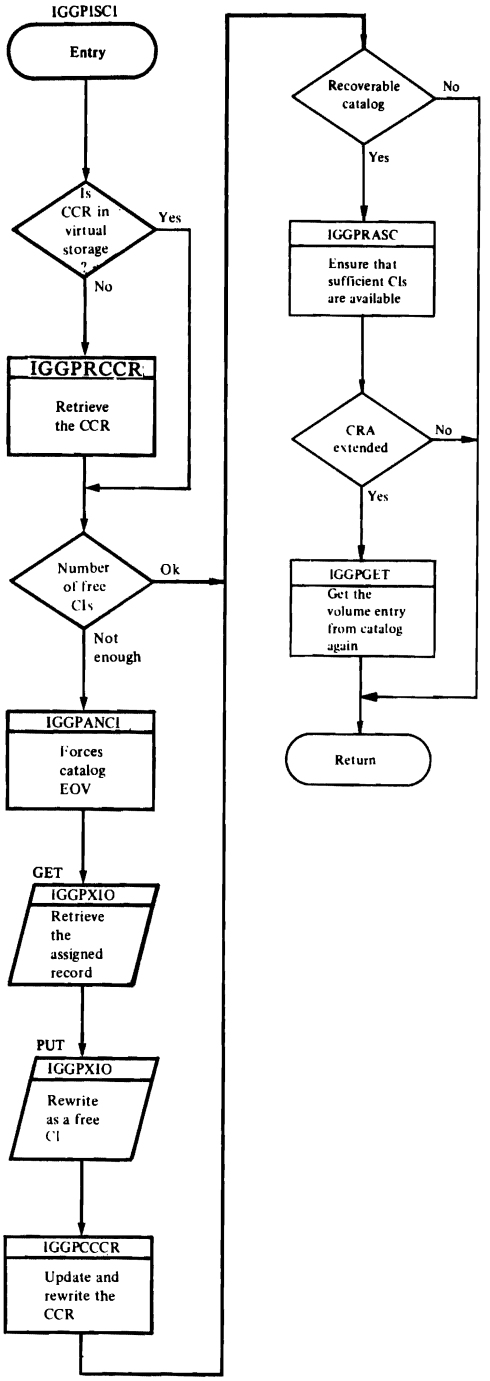



Figure 51.9. Ensure Availability of Catalog Control Intervals (IGGPISCI)

***IGGPRCCR—reads the Catalog Control Record (CCR) and updates control fields and RBAs from information contained within the CCR.***

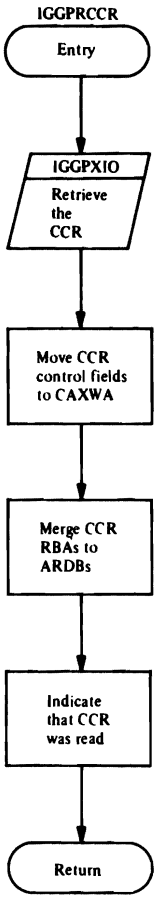



Figure 51.10. Read the CCR and Update Control Fields and RBAs (IGGPRCCR)

## *IGGPRAPU—CRA record write (update)*

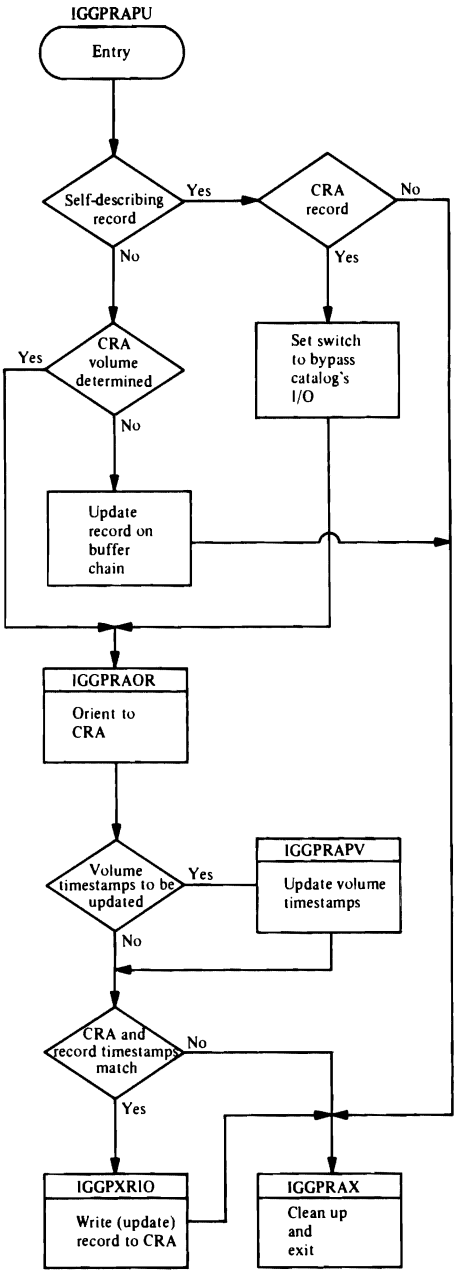



Figure 51.11. Write (Update) a CRA Record (IGGPRAPU)

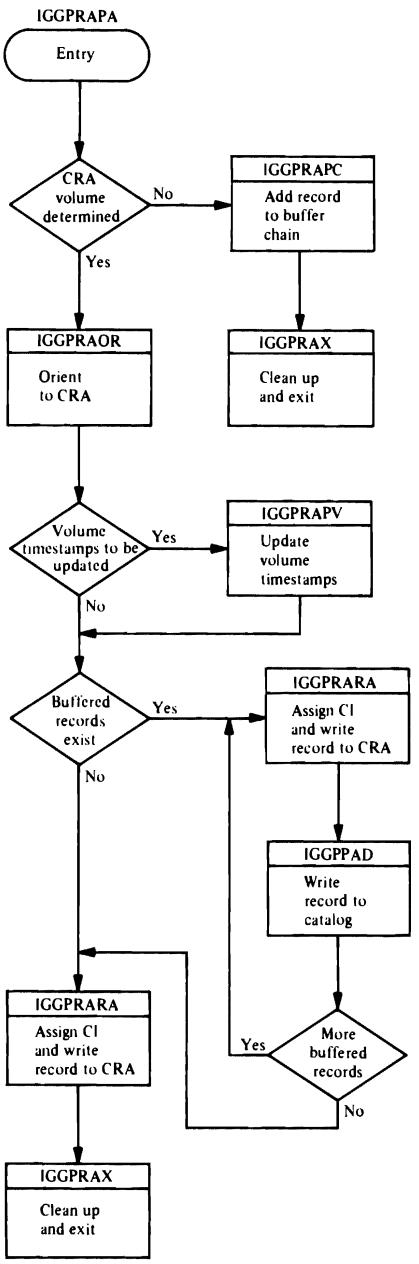



Figure 51.12. Write (Add) a CRA Record (IGGPRAPA)

## *IGGPRAPD—CRA record delete*

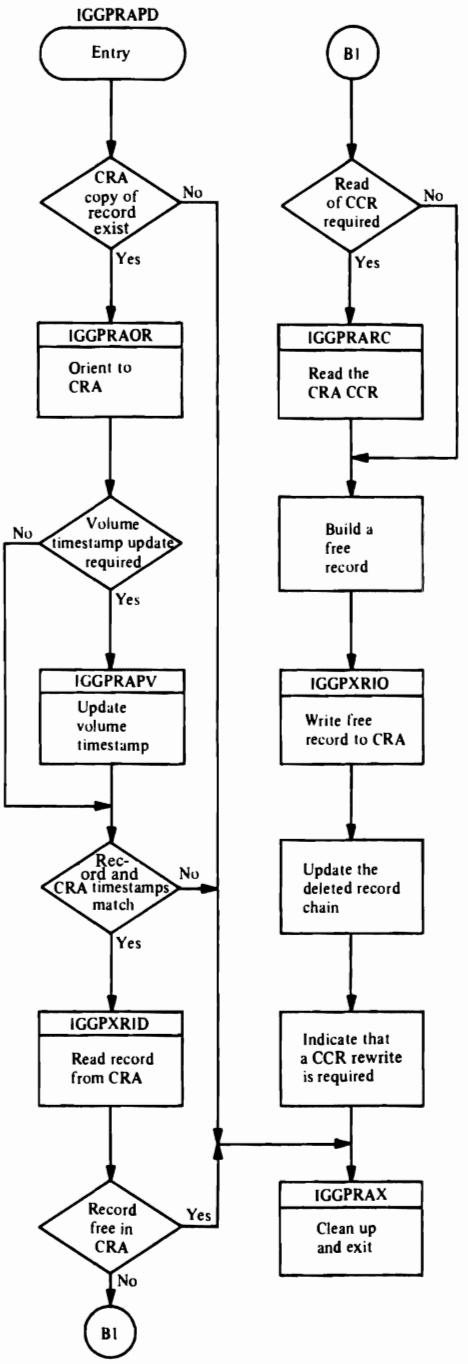



Figure 51.13. Delete a CRA Record (IGGPRAPD)

## *IGGPRAOR—orient to CRA*

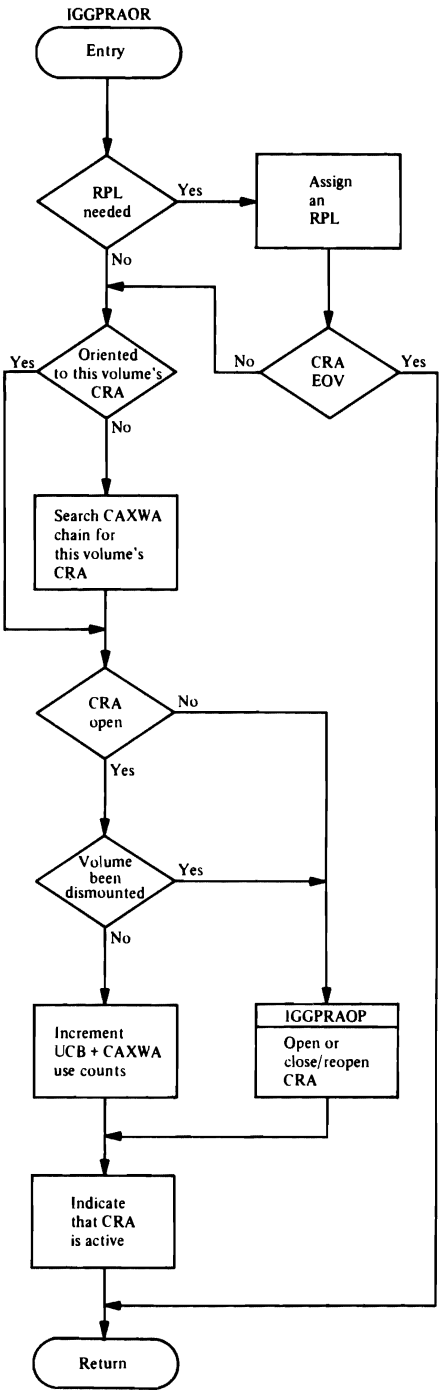



Figure 51.14. Orient to the CRA (IGGPRAOR)

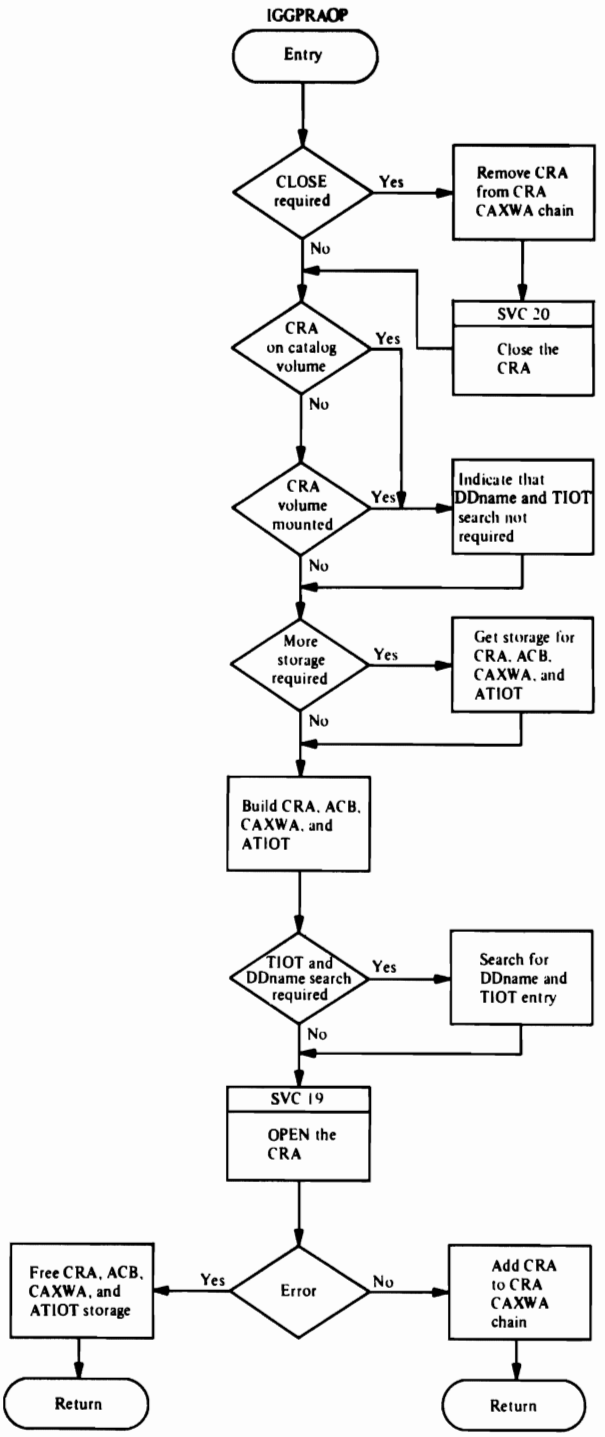



Figure 51.15. Open a CRA (IGGPRAOP)

## *IGGPRARA—assign control interval numbers to new CRA records and write them to the CRA*

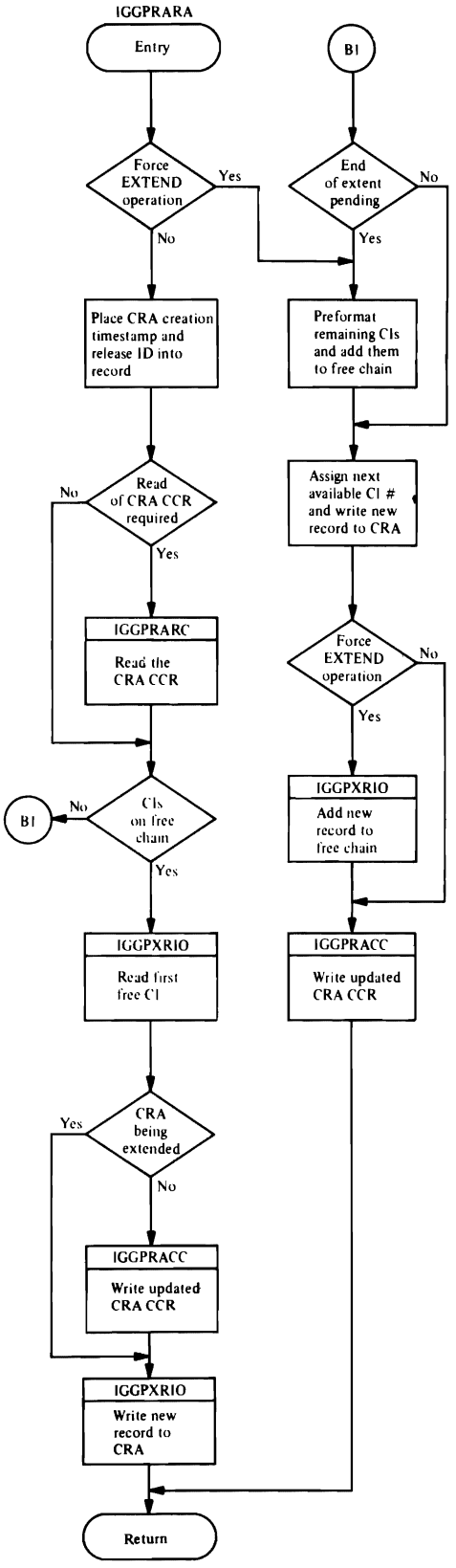



**Figure 51.16. Assign Control Interval Numbers to CRA Records (IGGPRARA)**

***IGGPRASC—ensures that there are enough free control intervals in a CRA so that the CRA won't be extended at the same time the CRA's volume catalog record is being extended***

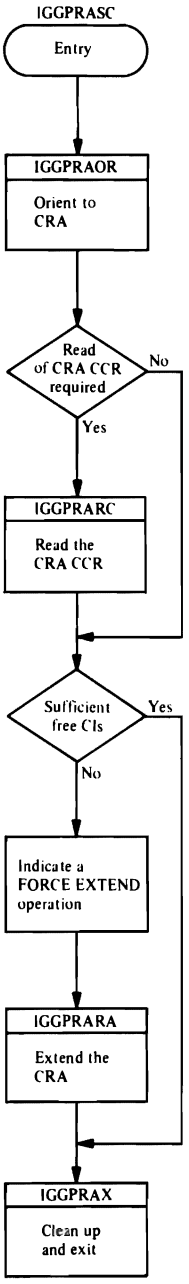



Figure 51.17. Ensure Availability of CRA Control Intervals (IGGPRASC)

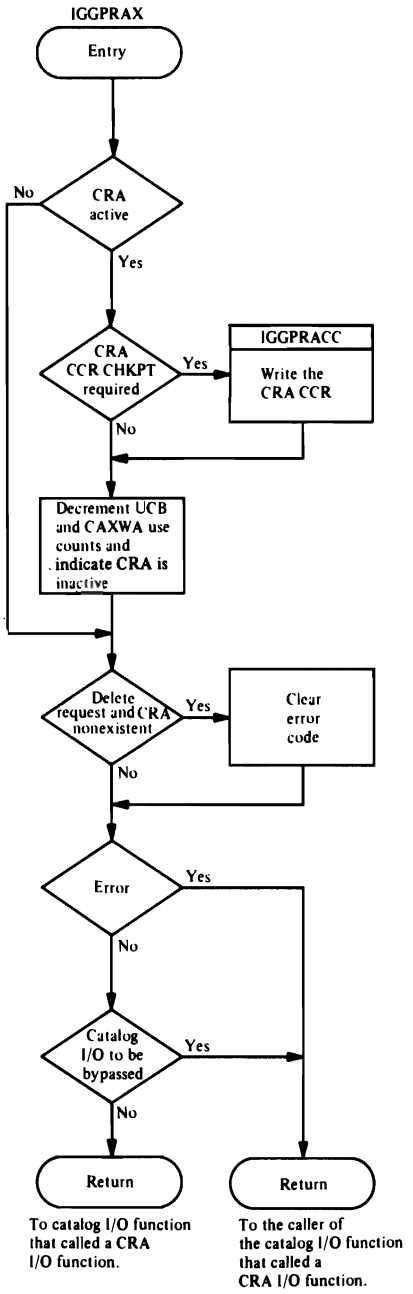



Figure 51.18. Return from CRA I/O Function (IGGPRAX)

### *IGGPXRIO—calls VSAM record management*

1. IGGPXRIO initializes an RPL.
2. IGGPXRIO issues GET or PUT—a VSAM record management request macro instruction.
3. If an error occurs, IGGPXRIO returns on error to IGGPRAEA to convert the RPL error code to an appropriate catalog management error code.

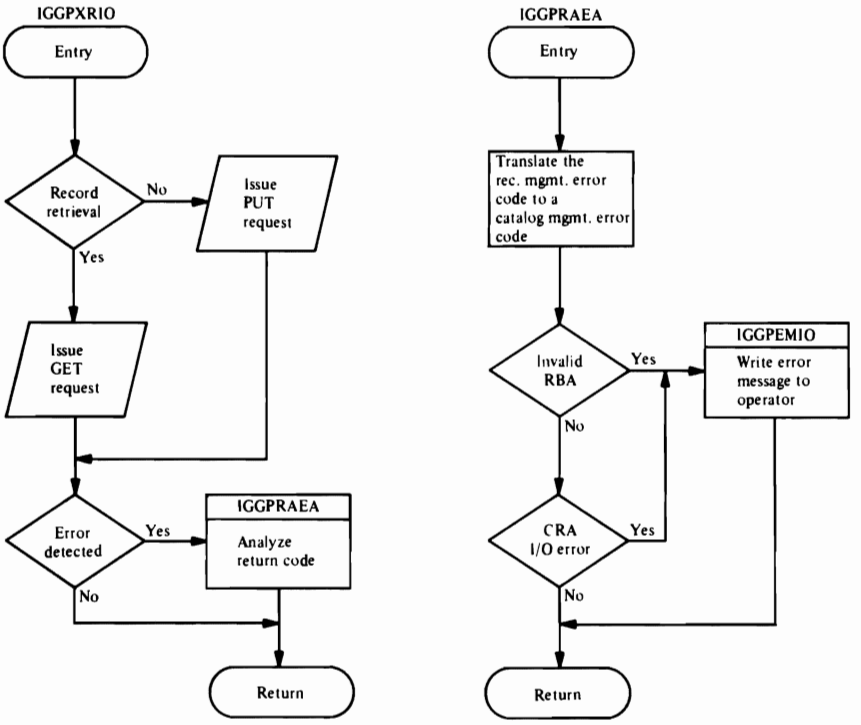



Figure 51.19. Call VSAM Record Management for CRA Request (IGGPXRIO)

# DIRECTORY

The directory section contains lists of items and cross-reference information related to each item in the list. The lists include:

- Module Directory
- Module Packaging
- External Procedure Directory
- Procedure Calls Directory
- Procedure Called-By Directory

## Module Directory

The module directory is organized alphabetically by symbolic module name. It lists the module's descriptive name, the module's procedure names (external entry points), the component to which the module belongs, and the method of operation diagrams and program organization compendium figures in which the module is referenced.

The component identifier codes are:

| | |
|---|---|
| II | ISAM Interface |
| CBM | Control Block Manipulation |
| RM | Record Management |
| O | Open |
| C | Close |
| EOV | End of Volume |
| CM | Catalog Management |
| C/R | Checkpoint/Restart |

| Module Name | Descriptive Name | External Procedure Names | Component | Method of Operation Diagrams | Program Organization Figures |
|---|---|---|---|---|---|
| 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 | — |
| IDA019C1 | Control Block Manipulation | IDA019C1 | CBM | CA1, CB1, CB2 | — |
| IDA019RA | Direct Record Locate | IDA019RA | RM | BC1, BE1, BH1, BJ1 | 20, 21, 22, 24 |
| IDA019RB | Index Search | IDA019RB | RM | BC1, BH1, BH8, BJ1, BM1 | 22, 34 |
| IDA019RC | Search Compressed Index Block | IDA019RC | RM | BC1, BH1, BH2, BH6, BI1, BJ1, BM1 | 22, 24, 25, 32, 38 |
| 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, 27, 28, 33, 34 |
| IDA0A05B | Restart | IDA0A05B | C/R | AH1, AH2 | 38.2 |

| Module Name | Descriptive Name | External Procedure Names | Component | Method of Operation Diagrams | Program Organization Figures |
|---|---|---|---|---|---|
| IDA0B05B | Restart | IDA0B05B | C/R | AI1 | 38.2 |
| IDA0C06C | Checkpoint | IDA0C06C | C/R | AG1 | 38.1 |
| 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, BH8, BH9 | 28, 33, 34 |
| | | IDAHLINS | RM | BH4 | 28 |
| | | IDANEWRD | RM | BH4 | 28 |
| IDA019RJ | Split Index Record | IDA019RJ | RM | BH4, BH7, BH8, BH9 | 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 | 26, 28, 29 |
| IDA019RL | Data Modify | IDA019RL | RM | BH2, BI1 | 24, 25 |
| IDA019RM | Data Insert | IDA019RM | RM | BE1, BF1, BH1, BH2, BH3 | 24, 25, 26, 27, 28 |
| | | IDACHKKR | RM | — | — |
| IDA019RN | Indexing Subroutines | IDA019RN | RM | — | — |
| | | IDAAQR | RM | BG3, BG4, BG5, BH8, BH9 | 29, 30, 33, 34 |
| | | IDAER | RM | BG5 | 30 |
| IDA019RO | Verify | IDA019RO | RM | BM1 | — |
| IDA019RP | ENDREQ and JRNAD | IDA019RP | RM | BK1, BK2 | — |
| | | IDAENDRQ | RM | BK1, BK2 | 31 |
| | | IDATJXIT | RM | BN1, BN3 | 20, 21, 23, 24, 25, 26, 27, 35, 38 |
| | | IDAUPXIT | RM | BS3 | 38 |
| IDA019RQ | Relative Record Subroutines | IDA019RQ | RM | BO1, BO2 | 35 |
| IDA019RR | Relative Record Driver | IDA019RR | RM | BC1, BD1, BJ1 | 23, 35 |
| | | IDARRDRL | RM | BJ1, BO1 | 23, 35 |
| IDA019RS | Spanned Record Data Modify | IDA019RS | RM | BH2, BI1 | 25 |
| | | IDAADSEG | RM | BH1, BH2 | 24, 25 |
| | | IDAMVSEG | RM | BH1, BH2 | 24, 25 |
| IDA019RT | Spanned Record Data Insert | IDA019RT | RM | BE1, BF1, BH1, BH3 | 24 |
| | | IDADARTV | RM | BC1, BD1 | 20, 21 |

| Module Name | Descriptive Name | External Procedure Names | Component | Method of Operation Diagrams | Program Organization Figures |
|---|---|---|---|---|---|
| | | IDAJRNSR | RM | — | — |
| | | IDASPNPT | RM | — | — |
| IDA019RU | Alternate-Index Upgrade Driver | IDA019RU | RM | BC1, BD1, BE1, BH1, BI1, BR1 | 37 |
| | | IDAXGPLH | RM | BB1 | — |
| IDA019RV | Locate Previous Sequence-Set Record | IDA019RV | RM | — | 28, 38 |
| | | IDAADVPH | RM | BD1 | 21 |
| | | IDARVRS1 | RM | — | 38 |
| IDA019RW | Buffer management, Part 2 | IDA019RW | RM | — | 22, 38 |
| | | IDAABF | RM | BH4 | 23, 28, 35, 38 |
| | | IDAAIBF | RM | — | 38 |
| | | IDAFRBA | RM | BN3 | 22, 38 |
| | | IDAGWSGW | RM | — | 38 |
| IDA019RX | Path Processing Driver | IDA019RX | RM | BQ1 | 36 |
| | | IDAGETWS | RM | — | — |
| | | IDARELWS | RM | — | — |
| | | IDARXBD | RM | — | — |
| IDA019RY | Shared Resources Buffer Management | IDA019RY | RM | BP1, BP2, BP3, BS1, BS2 | 38 |
| IDA019RZ | Buffer Management Interface | IDA019RZ | RM | — | 27, 38 |
| | | IDAEXCL | RM | — | 38 |
| | | IDAFREEB | RM | BC1, BD1, BG1, BG5, BH4, BH5, BM1, BN1, BN2, BN3, BO1, BO2, BS2 | 20, 21, 22, 24, 26, 27, 28, 30, 33, 34, 35, 38 |
| | | IDAGNFL | RM | BG3, BG4, BH3, BH8 | 29, 30, 33, 34, 38 |
| | | IDAGNNFL | RM | BE1, BF1, BG1, BH4, BH5, BN2, BO1 | 24, 26, 27, 28, 35, 38 |
| | | IDAGNXT | RM | BC1, BD1, BH4, BN1, BO1 | 20, 21, 23, 28, 35, 38 |
| | | IDAGRB | RM | BC1, BE1, BH1, BH3, BH4, BH5, BH9, BH10,BJ1, BM1, BN1, BO1, BS1 | 21, 22, 23, 27, 28, 33, 34, 35, 38 |
| | | IDAGWSEG | RM | — | 38 |
| | | IDAMRKBF | RM | — | 38 |
| | | IDASBF | RM | BE1, BH5, BK1 BN1, BN2 | 22, 27, 28, 35, 38 |
| | | IDASCHBF | RM | — | 38 |
| | | IDAWAIT | RM | BS1, BS3 | 22, 38 |

| Module Name | Descriptive Name | External Procedure Names | Component | Method of Operation Diagrams | Program Organization Figures |
|---|---|---|---|---|---|
| IDA019RZ | (Continued) | IDAWRBFR | RM | BE1, BG2, BH3, BH4, BH5, BH8, BH9, BK1, BK2, BN2, BN3, BO1, BO2 | 20, 24, 25, 26, 27, 28, 32, 35, 38 |
| | | IDAWRTBF | RM | — | 38 |
| IDA019R1 | Decode and Validate | IDA019R1 | RM | AD6, BB1, BK1, BK2, BL1 | 20, 21, 23, 24, 35, 36 |
| IDA019R2 | Buffer Management, Part 1 | IDA019R2 | RM | BS1, BS2 | 38 |
| IDA019R3 | I/O Management | IDA019R3 | RM | BS1, BT1 | 20, 22, 38 |
| IDA019R4 | Keyed/Addressed Request Driver | IDA019R4 | RM | BC1, BD1, BE1, BF1, BH3, BQ1, BR1 | 20, 21, 22 24, 25, 36 37 |
| IDA019R5 | I/O Error Analysis | IDA019R5 | RM | BB2, BK1, BL1 | 38 |
| | | IDADRQ | RM | BP1, BP3, BS1 | 35, 38 |
| | | IDAEOVIF | RM | BE1, BG2, BN2, BO1 | 26, 28, 29, 34 |
| | | IDAERROR | RM | — | — |
| | | IDAEXEX | RM | — | 38 |
| | | IDAEXITR | RM | BK1, BL1 | — |
| | | IDARSTRT | RM | — | — |
| IDA019R6 | Channel-End and Abnormal-End Appendages | IDA019R6 | RM | BS3, BT1 | — |
| IDA019R7 | Asynchronous Routine | IDA019R7 | RM | BT1 | — |
| IDA019R8 | Control-Interval Processing | IDA019R8 | RM | BM1, BN1, BN2, BN3 | — |
| IDA019R9 | Page-Fix and Start-I/O Appendages | IDA019R9 | RM | BT1 | — |
| IDA019SA | Control-Interval Initialization: Create Entry-Sequenced Data Set | IDA019SA | RM | BE1, BF1, BG1, BG2, BG3, BK2 | 26, 29, 30 |
| IDA019SB | Dynamically Build Channel Program Area for Shared Resouces | IDA019SB | RM | — | — |
| IDA019SF | Control-Area Split: Spanned Records | IDA019SF | RM | BH4 | 28 |
| IDA019S1 | Improved Control-Interval Processing Driver | IDA019S1 | RM | BN1, BN3 | — |
| IDA019S3 | Improved Control-Interval Processing: I/O Management | IDA019S3 | RM | BN1, BN3 | — |
| IDA0192A | VSAM Open String | IDA0192A | O | AC1, AC2, AC7 | 8, 9 |
| IDA0192B | Open a Cluster | IDA0192B | O | AC4, AC5, AC6 | 9 |
| IDA0192C | Catalog Interface | IDA0192C | O | AC2, AC4, AC5, AC6, AD5, AD6, AE1 | 9, 13, 15, 16, 17 |
| IDA0192D | Stage/Destage (ACQUIRE/ RELINQUISH) | IDA0192D | O/C/EOV | AD5 | 9, 13, 15, 17 |
| IDA0192F | Open Base Cluster, Path, and Upgrade Alternate Index | IDA0192F | O | AC3, AC4, AC5, AC6 | 9 |
| IDA0192G | Data Space Security Verification | IDA0192G | O | — | 16 |
| IDA0192I | ISAM Interface: Open Processing | IDA0192I | II | AC1, AC7 | — |

| Module Name | Descriptive Name | External Procedure Names | Component | Method of Operation Diagrams | Program Organization Figures |
|---|---|---|---|---|---|
| IDA0192M | Virtual-Storage Management | IDA0192M | O/C/EOV | — | 9, 11, 18 |
| IDA0192P | VSAM Open/Close/EOV: Problem Determination | IDA0192P | O | AD5 | 9, 13, 15, 17 |
| IDA0192S | VSAM Open/Close/EOV: SMF Record Build | IDA0192S | O | AC7 | 9, 13, 15, 17 |
| IDA0192V | Volume Mount and Verify | IDA0192V | O | AC3 | 9, 17 |
| IDA0192W | Channel Program Area Build | IDA0192W | O | AC1, AC4, AC5 AC6 | 9, 11 |
| IDA0192Y | String Build and Shared-Resource Processor | IDA0192Y | O/C | AC1, AC5, AF1 | 9, 11, 18 |
| IDA0192Z | Control Block Build | IDA0192Z | O | AC4, AC5, AC6 | 9 |
| IDA0200B | Close a Cluster | IDA0200B | C | AD2, AD3, AD5 | 13 |
| IDA0200S | ISAM Interface: Close Processing | IDA0200S | II | AD1, AD6 | 12 |
| IDA0200T | VSAM Close String | IDA0200T | C | AD1, AD2, AD3, AD4, AD6 | 13 |
| IDA0231B | Close (TYPE=T) a Cluster | IDA0231B | C | — | 15 |
| IDA0231T | VSAM Close (TYPE=T) String | IDA0231T | C | — | 15 |
| IDA0557A | VSAM End of Volume | IDA0557A | EOV | AE1, AE2 | 17 |
| IEFAB410 | Private (VSAM User's) Catalog Open | IEFAB410 | O | — | — |
| IEFAB411 | Private (VSAM User's) Catalog Close | IEFAB411 | C | — | — |
| IEFNB902 | AMP Parameter Interpreter | IEFNB902 | — | — | — |
| IFG0191X | Catalog Open ACB Processor, Load 1 | IFG0191X | CM | — | 10 |
| IFG0191Y | Catalog Open ACB Processor, Load 2 | IFG0191Y | CM | — | 9, 10 |
| IFG0192A | VSAM Open/Close/EOV String Load | IFG0192A | O/C/EOV | — | 8, 9, 12, 13, 15, 17 |
| IFG0192B | Invalid ACB Processing | IFG0192A | O/C/EOV | — | — |
| IFG0192Z | VSAM Catalog Open: ACB Processor | IFG0192Z | O | — | — |
| IFG0200N | Catalog Close/EOV ACB processor | IFG0200N | CM | — | 13, 14 |
| | | IFG0550Y | CM | — | 17 |
| IGG0CLAA | SUPERLOCATE: Upgrade Support | IGGPSLEN<br>IGGPSLIV | CM<br>CM | DG1, DG2,<br>DG3, DG4 | —<br>— |
| IGG0CLAB | Catalog Management Driver | IGGPACDV | CM | DB1, DD1, DE1, DH1 | 40 |
| IGG0CLAC | Master Catalog ENQ/CVOL Search | IGGPMCO | CM | — | 40 |
| IGG0CLAD | Master Catalog and Catalog Recovery Area Open | IGGPMCO2 | CM | EE2 | 40, 51 |
| | | IGGPCRAD | CM | — | — |
| | | IGGPRAOP | CM | — | — |
| IGG0CLAE | CMS DEFINE CATALOG: Open and Build | IGGPDCME | CM | EE2 | 51 |
| | | IGGPMEBM | CM | — | — |
| IGG0CLAF | CMS DELETE CATALOG | IGGPDELC | CM | EB1, EL1 | 50 |
| | | IGGPEMIO | CM | — | — |
| | | IGGPEMSG | CM | — | — |

## Module Directory

| Module Name | Descriptive Name | External Procedure Names | Component | Method of Operation Diagrams | Program Organization Figures |
|---|---|---|---|---|---|
| IGG0CLAG | Catalog I/O Subfunctions | IGGPAOCI | CM | DM2, ED3, ED5 | 50, 51 |
| | | IGGPAXCI | CM | — | 45, 48 |
| | | IGGPCCCR | CM | — | 50 |
| | | IGGPIORA | CM | — | — |
| | | IGGPISCI | CM | — | 43, 48, 51 |
| | | IGGPPAD | CM | DH1 | 50 |
| | | IGGPPADC | CM | — | — |
| | | IGGPPDE | CM | — | 47, 50 |
| | | IGGPPDEC | CM | — | — |
| | | IGGPPUPC | CM | DH1, EM1 | 43, 48, 50 |
| | | IGGPXIO | CM | — | — |
| IGG0CLAH | Search catalog | IGGPRPLF | CM | — | — |
| | | IGGPSCAT | CM | DB1, DC1, DC2, EG1, EM1 | 40, 50 |
| | | IGGPDFM1 | CM | — | — |
| IGG0CLAI | DELETE SPACE: Force Support | IGGPFDSP | CM | EK1 | 50 |
| IGG0CLAJ | CMS DEFINE: Build Data and Index Entries | IGGPDBDI | CM | ED1, ED2, ED3, ED4 | 51 |
| IGG0CLAK | Complete Define of an Entry | IGGPDCMB | CM | ED2, ED4 | 51 |
| IGG0CLAL | CMS DEFINE: 1st Module | IGGPDEF | CM | EB1, EB2, EC1 | 50, 51 |
| | | IGGPDTIM | CM | — | 50 |
| IGG0CLAM | SUPERLOCATE | IGGPDBVC | CM | — | 50 |
| | | IGGPSLEL | CM | — | — |
| | | IGGPSLOC | CM | DB1, DG1, DG2, DG3, DG4 | 40 |
| IGG0CLAN | CMS DEFINE: 2nd Module | IGGPDCCE | CM | ED1, ED3, EE1 | — |
| | | IGGPDRDA | CM | ED1, ED3 | — |
| | | IGGPDSCB | CM | EC1 | 51 |
| | | IGGPPSEM | CM | — | — |
| IGG0CLAP | CMS DEFINE: 3rd Module | IGGPDCDA | CM | EC1, EE1 | 51 |
| IGG0CLAQ | CMS DEFINE SPACE | IGGPDEFS | CM | EB1, ED1, ED3, EE1, EG1, EG2 | 50, 51 |
| IGG0CLAR | Suballocate: Assign Candidate Volumes | IGGPSALL | CM | DI1, DJ1, ED1, ED3, EE1, EH1 | 43, 50, 51 |
| IGG0CLAS | CMS Define Catalog: 1st Module | IGGPDCRC | CM | — | — |
| | | IGGPDEFC | CM | EC1, EE1 | 51 |
| IGG0CLAT | CMS (Catalog Management Services) Common Processing | IGGPCDVR | CM | DB1, DD1, EB1 | 40, 50 |
| IGG0CLAU | Suballocate: Obtain Space | IGGPSALS | CM | DJ1, DJ2, EJ2 | 43, 49, 51 |

## Module Directory

| Module Name | Descriptive Name | External Procedure Names | Component | Method of Operation Diagrams | Program Organization Figures |
|---|---|---|---|---|---|
| IGG0CLAV | Modify Catalog Field | IGGPDEL2 | CM | DM3 | 42, 47 |
| | | IGGPMOD | CM | DI4, DI6, DJ1 DM1, DM3, EM1 | 42 |
| | | IGGPSGOP | CM | — | 47, 48 |
| | | IGGPUPD | CM | DB1, DH1, DI3 | 40, 42 |
| IGG0CLAW | Add Group Occurrence (Set of Fields) | IGGPADGO | CM | DM1 | 42, 45 |
| | | IGGPGNEX | CM | — | 45, 48 |
| | | IGGPGREL | CM | — | 45 |
| | | IGGPIGOP | CM | DM1 | 45, 48 |
| | | IGGPPREC | CM | DH1 | 42, 47, 48 |
| IGG0CLAX | Alter-2 (Modify) | IGGPALT2 | CM | DH1, DM2 | 42, 46 |
| | | IGGPDGO | CM | — | 47, 48 |
| | | IGGPDGOP | CM | — | 47, 48 |
| | | IGGPEXPD | CM | DM2 | 46 |
| | | IGGPMGO | CM | — | 45, 48 |
| | | IGGPSHNK | CM | DM2 | 46 |
| IGG0CLAY | Scan Catalog Parameter List | IGGPSCNC | CM | DB1, DE1, DM2 | 40, 41, 42 |
| IGG0CLAZ | Extract Catalog Field | IGGPEXT | CM | DJ2, DI3, DI4 | 41, 49 |
| | | IGGPLOC | CM | DB1, DE1, DE2 | 40, 41 |
| IGG0CLA0 | Show Catalog Processing | IGG0CLA0 | CM | EI2 | — |
| IGG0CLA1 | Catalog Transient Load: PSA Loader | IGG0CLA1 | CM | DB1 | 40 |
| IGG0CLA6 | CMS DEFINE SPACE: Build the Data Space Group and Data Set Directory Entry Sets of Fields | IGGPDFS2 | CM | EG1 | — |
| | | IGGPBJFB | CM | EG1 | — |
| | | IGGPCBPT | CM | — | — |
| | | IGGPCRTC | CM | EG1 | — |
| IGG0CLA7 | CMS DELETE | IGGPDEMV | CM | — | 50 |
| | | IGGPDUSC | CM | — | 50 |
| | | IGGPDVMV | CM | — | 50 |
| | | IGGPMCRA | CM | — | — |
| | | IGGPVMSC | CM | — | 50 |
| | | IGGPDF4T | CM | EJ2 | 50 |
| IGG0CLA8 | CMS DEFINE | IGGPDFRS | CM | ED2, ED4 | — |
| IGG0CLBA | Tests | IGGPGREC | CM | DL1, DL2 | 41, 42, 45, 47, 48 |
| | | IGGPGVAL | CM | DE2, DL1, DL2, DM2 | 41, 42, 46 |
| | | IGGPTSTS | CM | DE1, DH1 | 41, 42 |
| IGG0CLBB | UPDATE-Extend | IGGPUPDE | CM | DH1, DI1, DI2 | 42, 43 |
| IGG0CLBC | UPDATE-Extend Initialization | IGGPINIT | CM | DI1 | 43 |
| | | IGGPSVOL | CM | DI1 | 43 |

## Module Directory

| Module Name | Descriptive Name | External Procedure Names | Component | Method of Operation Diagrams | Program Organization Figures |
|---|---|---|---|---|---|
| IGG0CLBD | CMS ALTER Processing | IGGPALT | CM | EB1, EH1, EH2, EH3 | 50 |
| IGG0CLBE | CMS ALTER: Volume Processing | IGGPALEC | CM | — | — |
| | | IGGPALVL | CM | EH3 | 50 |
| IGG0CLBF | Subscratch: Return Space | IGGPSSCR | CM | DI5, EJ1, EJ2 | 50 |
| IGG0CLBG | CMS DELETE Processing | IGGPDEL | CM | EB1, EJ1, EJ4 | 50 |
| | | IGGPDEXA | CM | — | — |
| | | IGGPDOPN | CM | EJ1 | — |
| | | IGGPDLXT | CM | — | — |
| IGG0CLBH | CMS DEFINE NonVSAM Processing | IGGPDEFA | CM | EB1, EF1 | 50 |
| IGG0CLBI | Catalog GET and Recovery Subfunctions | IGGPGET | CM | DC2, DI4, DI6, DJ1 | 40, 41, 42, 43, 49, 50 |
| | | IGGPGETC | CM | — | — |
| | | IGGPTNXO | CM | DI5 | — |
| | | IGGPTXO | CM | — | — |
| | | IGGPUCRS | CM | — | — |
| IGG0CLBJ | GENDSP Processing | IGGPGDSP | CM | DB1, DF1 | 40 |
| IGG0CLBK | LSPACE Processing | IGGPLDCS | CM | DK1 | — |
| | | IGGPLSP | CM | DB1, DK1, EG2, EJ3 | 40, 50 |
| IGG0CLBL | CMS DELETE SPACE Processing | IGGPDELS | CM | EB1, EK1 | 50 |
| IGG0CLBM | Check Authorization | IGGPCKAU | CM | DB1, DD1 | 40, 50 |
| IGG0CLBN | CMS ALTER: Remove Volume | IGGPALVR | CM | EH3 | 50 |
| | | IGGPVRD | CM | EH3 | 50 |
| IGG0CLBO | CRA I/O Subfunctions | IGGPRACC | CM | — | — |
| | | IGGPRAEA | CM | — | — |
| | | IGGPRAG | CM | — | — |
| | | IGGPRAPA | CM | — | — |
| | | IGGPRAPC | CM | — | — |
| | | IGGPRAPD | CM | — | — |
| | | IGGPRAPU | CM | — | — |
| | | IGGPRARC | CM | — | — |
| | | IGGPRASC | CM | — | — |
| | | IGGPRAX | CM | — | — |
| | | IGGPXRIO | CM | — | — |
| IGG0CLBP | UPDATE-Extend Interface to DADSM | IGGPSPAC | CM | — | 43 |
| | | IGGPRET1 | CM | — | — |
| IGG0CLBQ | CMS LISTCAT Processing | IGGPLSTC | CM | EB1, EI1 | 50 |
| IGG0CLBR | Bit Manipulation Routine | IGGPBMR | CM | DJ2 | 48, 50 |
| IGG0CLBS | Retrieval of Derived Catalog Record Information | IGGPXEXT | CM | DL1 | — |
| | | IGGPXVAL | CM | DL1 | — |

| Module Name | Descriptive Name | External Procedure Names | Component | Method of Operation Diagrams | Program Organization Figures |
|---|---|---|---|---|---|
| IGG0CLBT | Modification of Derived Volume Catalog Record Information | IGGPXDGO | CM | DM1 | — |
| | | IGGPXEL2 | CM | DM3 | — |
| | | IGGPXLT2 | CM | DM2 | — |
| | | IGGPXMOD | CM | DM1 | — |
| IGG0CLBU | Catalog Read/Write Format 4 DSCB | IGGPF4DQ | CM | — | 50 |
| | | IGGPF4RD | CM | EM1 | 50 |
| | | IGGPF4WR | CM | EM1 | 50 |
| IGG0CLBV | Catalog SMF Record Build | IGGPSMFA | CM | ED2, ED4, ED5, EE2 | — |
| | | IGGPSMFF | CM | DK1 | — |
| | | IGGPSMFL | CM | — | — |
| | | IGGPSMFR | CM | — | — |
| | | IGGPSMFS | CM | — | — |
| IGG0CLBW | Modify: Delete/Insert Processing | IGGPDEIN | CM | — | 46 |
| IGG0CLBX | CMS DEFINE: 4th Module | IGGPDSPC | CM | — | 51 |
| | | IGGPDSIM | CM | — | — |
| IGG0CLBY | CMS DEFINE: 5th Module | IGGPDRSP | CM | ED3 | 51 |
| IGG0CLBZ | Convert Processing | IGGPCONV | CM | EB1, EM1 | 49 |
| | | IGGPGALO | CM | EM1 | 49 |
| | | IGGPVALI | CM | EM1 | 49 |
| IGG0CLB0 | Define Data and Index Characteristics | IGGPCMKY | CM | — | — |
| | | IGGPDCCO | CM | — | — |
| IGG0CLB1 | Upgrade Management | IGGPUADD | CM | — | — |
| | | IGGPUDEL | CM | — | — |
| IGG0CLB2 | ALTER: Upgrade/Update | IGGPAUPG | CM | — | — |
| IGG0CLB3 | SMF GET/PUT for Alter | IGGPSMF | CM | — | — |
| | | IGGPSMFG | CM | — | — |
| IGG0CLB4 | DEFINE CRA | IGGPDCRA | CM | EB1, EE1, EE3 | — |
| IGG0CLB5 | DEFINE AIX/PATH | IGGPDCLS | CM | EJ3, EJ4 | 49 |
| | | IGGPDEAX | CM | EJ1, EJ3 | 49 |
| | | IGGPDEPT | CM | EJ1, EJ3 | 49 |
| | | IGGPDIAX | CM | EJ3, EJ4 | 49 |
| | | IGGPDIPT | CM | EJ3, EJ4 | 49 |
| | | IGGPDUPG | CM | EJ3, EJ4 | 49 |
| IGG0CLB6 | TSO Interface for Security | IGGPGTSO | CM | — | — |
| | | IGGPWTSO | CM | — | — |
| | | IGGPINMD | CM | DD2 | — |
| | | IGGPSPSC | CM | DD1 | — |
| IGG0CLB7 | Reset Reusable Data Set | IGGPRUS | CM | DE1, DH1, DI3, DI4, DI5, DI6 | 44 |

| Module Name | Descriptive Name | External Procedure Names | Component | Method of Operation Diagrams | Program Organization Figures |
|---|---|---|---|---|---|
| IGG0CLB8 | CMS DEFINE: Space Recovery | IGGPCNBO | CM | — | — |
| | | IGGPDFBO | CM | — | 51 |
| | | IGGPDUND | CM | — | 50 |
| IGG0CLB9 | VSAM CMS Define | IGGPPAIX | CM | ED3 | 50 |
| | | IGGPPATH | CM | ED5 | 50 |
| IGG0CLCA | OS/VS Catalog Request Handler | IGG0CLCA | CM | — | — |
| IGG0CLCB | VSAM Catalog Request Handler | IGG0CLCB | CM | — | — |
| IGG0CLC9 | Catalog First Load | IGG0CLC9 | CM | DB1 | 40 |
| IGG0CLDA | CRA Services | IGGPMODI | CM | EE3 | — |
| | | IGGPRAPV | CM | — | — |
| | | IGGPWCAT | CM | EE3 | — |
| | | IGGPWCRA | CM | EE3 | — |
| IGG0102G | OS/VS DADSM: Obtain Space For a VSAM Data Space | IGG0102G | — | — | — |
| IMDUSRF9 | IMDPRDMP Format Appendage | IMDUSRF9 | CM | — | — |

# Module Packaging

VSAM modules reside in pageable virtual storage. The minimum virtual storage that can be specified when VSAM is included in the OS/VS system is 2048K bytes. If VSAM was specified at system generation, the minimum virtual storage must also have been specified. If VSAM was not specified at system generation, you must perform system generation to include VSAM in the system.

The following table lists the VSAM load modules and transients that are resident in the SVCLIB or LPALIB library; they are loaded into the pageable supervisor or link-pack area by nucleus initialization (NIP) at initial program load (IPL).

| Name | Description | VSAM Modules |
|---|---|---|
| **Catalog Management** | | |
| IGG0CLC9 | Catalog Management Modules | IGG0CLAA, IGG0CLAB, IGG0CLAC, IGG0CLAD, IGG0CLAE, IGG0CLAF, IGG0CLAG, IGG0CLAH, IGG0CLAI, IGG0CLAJ, IGG0CLAK, IGG0CLAL, IGG0CLAM, IGG0CLAN, IGG0CLAP, IGG0CLAQ, IGG0CLAR, IGG0CLAS, IGG0CLAT, IGG0CLAU, IGG0CLAV, IGG0CLAW, IGG0CLAX, IGG0CLAY, IGG0CLAZ, IGG0CLA6, IGG0CLA7, IGG0CLA8, IGG0CLBA, IGG0CLBB, IGG0CLBC, IGG0CLBD, IGG0CLBE, IGG0CLBF, IGG0CLBG, IGG0CLBH, IGG0CLBI, IGG0CLBJ, IGG0CLBK, IGG0CLBL, IGG0CLBM, IGG0CLBN, IGG0CLBO, IGG0CLBP, IGG0CLBQ, IGG0CLBR, IGG0CLBS, IGG0CLBT, IGG0CLBU, IGG0CLBV, IGG0CLBW, IGG0CLBX, IGG0CLBY, IGG0CLBZ, IGG0CLB0, IGG0CLB1, IGG0CLB2, IGG0CLB3, IGG0CLB4, IGG0CLB5, IGG0CLB6, IGG0CLB7, IGG0CLB8, IGG0CLB9,IGG0CLC9,IGG0CLC9 |
| **Control Block Manipulation** | | |
| IDA019C1 | Control Block Manipulation | IDA019C1 |

| Name | Description | VSAM Modules |
|---|---|---|
| **Catalog Management** | | |
| **Record Management** | | |
| IDA019L1 | Modules Record Management | IDA019RA, IDA019RB, IDA019RC, IDA019RE, IDA019RF, IDA019RG, IDA019RH, IDA019RI, IDA019RJ, IDA019RK, IDA019RL, IDA019RM, IDA01RN, IDA01RO, IDA019RP, IDA019RQ, IDA01RR, IDA019RU, IDA019RV, IDA01RW, IDA019RX, IDA019RY, IDA019RZ, IDA019R1, IDA019R2, IDA019R3, 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 |
| IDA019R6 | Channel/Abnormal End Appendage | IDA019R6 |
| IDA019R7 | I/O Asynchronous Routine | IDA019R7 |
| IDA019R9 | Start I/O Page-Fix Appendage | IDA019R9 |
| **ISAM Interface** | | |
| IDAIIFBF | ISAM Interface FREEDBUF | IDAIIFBF |
| IDAIIPM1 | ISAM Interface QISAM Load | IDAIIPM1 |
| IDAIIPM2 | ISAM Interface QISAM Scan | IDAIIPM2 |
| IDAIIPM3 | ISAM Interface BISAM | IDAIIPM3 |
| IDAIISM1 | ISAM Interface SYNAD | IDAIISM1 |
| **Checkpoint/Restart** | | |
| The Checkpoint/Restart modules are packaged with Open/Close/EOV module IDA0192A. | | |
| IDA0C06C | Checkpoint | IDA0C06C |
| IDA0A05B | First Restart | IDA0A05B |
| IDA0B05B | Second Restart | IDA0B05B |
| **Open/Close/End of Volume** | | |
| IDA0192A | Open/Close/End of Volume Modules | IDA0192A, IDA0192B, IDA0192C, IDA0192D, IDA0192F, IDA0192G, IDA0192I, IDA0192M, IDA0192P, IDA0192S, IDA0192V, IDA0192W, IDA0192Y, IDA0192Z, IDA0200B, IDA0200S, IDA0200T, IDA0231B, IDA0231T, IDA0557A |
| **VSAM Transient Routines** | | |
| IFG0192A | VSAM Open/Close/End of Volume Loader | IFG0192A |
| IFG0191X | VSAM Catalog Open ACB Processor, Load 1 | IFG0191X |
| IFG0191Y | VSAM Catalog Open ACB Processor, Load 2 | IFG0191Y |
| IFG0200N | VSAM Catalog Close/End of Volume ACB Processor | IFG0200N |
| IGC0A05B | VSAM Release 2 Restart Controller | IGC0A05B |

The following module is brought from LINKLIB into the user's job pack area whenever it is linked to or from the user's program.

| Name | Description | Catalog Management Module |
|---|---|---|
| IGG0CLA0 | Show Catalog Processor | IGG0CLA0 |

# External Procedure Directory

The external procedure directory is organized alphabetically by symbolic procedure name (external entry point name). It lists the module that contains the procedure, the procedure's descriptive name, and the method of operation diagrams and program organization compendium figures in which the procedure is referenced.

| Procedure Name | Module Name | Descriptive Name | Method of Operation Diagrams | Program Organization Figures |
|---|---|---|---|---|
| IDAABF | IDA019RW | Buffer Management: Add Buffer to Placeholder (PLH) | BH4 | 23, 28, 35, 38 |
| IDAADSEG | IDA019RS | Insert a Spanned-Record-Segment Entry into a Sequence-Set Record | BH1, BH2 | 24, 25 |
| IDAADVPH | IDA019RV | Advance Placeholder Backwards | BD1 | 21 |
| IDAAIBF | IDA019RW | Add Insert Buffer to Chain | — | 38 |
| IDAAQR | IDA019RN | Split Index Record: Assign RBA to the Index Record | BG3, BG4, BG5, BH8, BH9 | 29, 30, 33, 34 |
| IDACAT11 | IDACAT11 | Private (User's) Catalog Open | BB1 | — |
| IDACAT12 | IDACAT12 | Private (User's) Catalog Close | — | — |
| IDACAT13 | IDACAT13 | Private Catalog Task Termination | — | — |
| IDACHKKR | IDA019RM | Check Key for Proper Key Range | — | — |
| IDADARTV | IDA019RT | Retrieve a Spanned Record | BC1, BD1 | 20, 21 |
| IDADRQ | IDA019R5 | Data Insert: Defer the Request until the Device is Available | BP1, BP3, BS1 | 35, 38 |
| IDAENDRQ | IDA019RP | ENDREQ Request | BK1, BK2 | 31 |
| IDAEOVIF | IDA019R5 | Data Insert: Interface to the VSAM End of Volume Routine | BE1, BG2, BN2 BO1 | 26, 28, 29, 34 |
| 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, BL1 | — |
| IDAFRBA | IDA019RW | Buffer Management: Determine Next RBA for Sequential Processing | BN3 | 22, 38 |
| IDAFREEB | IDA019RZ | Free a Buffer | BC1, BD1, BG1, BG5, BH4, BH5, BM1, BN1, BN2, BN3, BO1, BO2, BS2 | 20, 21, 22, 24, 26, 27, 28, 30, 33, 34, 35, 38 |
| IDAGETWS | IDA019RX | Get Working Storage | — | — |
| IDAGNFL | IDA019RZ | Buffer Management: Obtain an Empty Buffer | BG3, BG4, BH3, BH8 | 29, 30, 33, 34, 38 |
| IDAGNNFL | IDA019RZ | Buffer Management: Obtain Next Empty Buffer Buffer for the Placeholder | BE1, BF1, BG1, BH4, BH5, BN2, BO1 | 24, 26, 27, 28, 35, 38 |
| IDAGNXT | IDA019RZ | Buffer Management: Obtain Next Buffer in Sequence | BC1, BD1, BH4, BN1, BO1 | 20, 21, 23, 28, 35, 38 |

## External Procedure Directory

| Procedure Name | Module Name | Descriptive Name | Method of Operation Diagrams | Program Organization Figures |
|---|---|---|---|---|
| IDAGRB | IDA019RZ | Buffer Management: Obtain the Buffer that Contains the Specified RBA | BC1, BE1, BH1, BH3, BH4, BH5, BH9,BH10, BJ1, BM1, BN1, BO1, BS1 | 21, 22, 23, 27, 28, 33 34, 35, 38 |
| IDAGWSEG | IDA019RZ | Get a Work Segment (for Shared Resources) | — | 38 |
| IDAGWSGW | IDA019RW | Get a Work Segment | — | 38 |
| IDAHLINS | IDA019RI | Insert Entry into Index-Set Record | BH4 | 28 |
| 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 |
| IDARELWS | IDA019RX | Release Working Storage | — | — |
| IDAREPOS | IDA019RE | Reposition Placeholder | — | — |
| IDARRDRL | IDA019RR | Direct Record Locate for Relative Record | BJ1, BO1 | 23, 35 |
| IDARSTRT | IDA019R5 | Restart | — | — |
| IDARVRS1 | IDA019RV | Order Buffers | — | 38 |
| IDARXBD | IDA019RX | Increase Working Buffer Length | — | — |
| IDAR | IDA019RJ | Split Index Record: Read the Record | BG5, BH9, BH10 | 30 |
| IDASBF | IDA019RZ | Buffer Management: Remove Buffers from Placeholder | BE1, BH5, BK1, BN1, BN2 | 22, 27, 28, 35, 38 |
| IDASCHBF | IDA019RZ | Share a Buffer | — | 38 |
| 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, 23, 24, 25, 26, 27, 35, 38 |
| IDAUPXIT | IDA019RP | Exit to a User Processing Routine | BS3 | 38 |
| IDAWAIT | IDA019RZ | Buffer Management: Wait for Completion Of I/O operations | BS1, BS3 | 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 | BE1, 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 | — | 38 |
| IDAXGPLH | IDA019RU | Get a Placeholder | BB1 | — |
| IDA0A05B | IDA0A05B | Checkpoint/Restart:Restart | AH1, AH2 | 38.2 |
| IDA0B05B | IDA0B05B | Checkpoint/Restart:Restart load 2 | AI1 | 38.2 |
| IDA0C06C | IDA0C06C | Checkpoint/Restart:Checkpoint | AG1 | 38.1 |
| IDA019C1 | IDA019C1 | Control Block Manipulation | CA1, CB1, CB2 | — |
| IDA019RA | IDA019RA | Direct Record Locate | BC1, BE1, BH1, BJ1 | 20, 21, 22, 24 |
| IDA019RB | IDA019RB | Index Search | BC1, BH1, BH8, BJ1, BM1 | 22, 34 |
| IDA019RC | IDA019RC | Search Compressed Index Block | BC1, BH1, BH2, BH6, BI1, BJ1, BM1 | 22, 24, 25, 32, 38 |
| 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, 27, 28, 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, BH8, BH9 | 28, 33, 34 |
| IDA019RJ | IDA019RJ | Split Index Record | BH4, BH7, BH8, BH9 | 34 |
| IDA019RK | IDA019RK | Preformat | BG2, BH4, BH8, BH9, BK2, BN2, BO1 | 26, 28, 29 |
| IDA019RL | IDA019RL | Data Modify | BH2, BI1 | 24, 25 |
| IDA019RM | IDA019RM | Data Insert | BE1, BF1, BH1, BH2, BH3 | 24, 25, 26 27, 28 |
| IDA019RN | IDA019RN | Indexing Subroutines | — | — |
| IDA019RO | IDA019RO | Verify | BM1 | — |
| IDA019RP | IDA019RP | ENDREQ and JRNAD | BK1, BK2 | — |
| IDA019RQ | IDA019RQ | Relative Record Subroutines | BO1, BO2 | 35 |
| IDA019RR | IDA019RR | Relative Record Driver | BC1, BD1, BJ1 | 23, 35 |
| IDA019RS | IDA019RS | Spanned Record Data Modify | BH2, BI1 | 25 |
| IDA019RT | IDA019RT | Spanned Record Data Insert | BE1, BF1, BH1, BH3 | 24 |
| IDA019RU | IDA019RU | Alternate-Index Upgrade Driver | BC1, BD1, BE1, BH1, BI1, BR1 | 37 |
| IDA019RV | IDA019RV | Locate Previous Sequence-Set Record | — | 38 |

## External Procedure Directory

| Procedure Name | Module Name | Descriptive Name | Method of Operation Diagrams | Program Organization Figures |
|---|---|---|---|---|
| IDA019RW | IDA019RW | Buffer Management, Part 2 | — | 22, 38 |
| IDA019RX | IDA019RX | Path Processing Driver | BQ1 | 36 |
| IDA019RY | IDA019RY | Shared Resources Buffer Management | BP1, BP2, BP3, BS1, BS2 | 38 |
| IDA019RZ | IDA019RZ | Buffer Management Interface | — | 27, 38 |
| IDA019R1 | IDA019R1 | Record Management: Request Decode and Validate | AD6, BB1, BK1, BK2, BL1 | 20, 21, 23, 24, 35, 36 |
| IDA019R2 | IDA019R2 | Buffer Management, Part 1 | BS1, BS2 | 38 |
| IDA019R3 | IDA019R3 | I/O Management | BS1, BT1 | 20, 22, 38 |
| IDA019R4 | IDA019R4 | Keyed/Addressed Request Driver | BC1, BD1, BE1, BF1, BH3, BQ1, BR1 | 20, 21, 22, 24, 25, 36, 37 |
| IDA019R5 | IDA019R5 | I/O Error Analysis | BB2, BK1, BL1 | 38 |
| IDA019R6 | IDA019R6 | Channel-End and Abnormal-End Appendages | BS3, BT1 | — |
| IDA019R7 | IDA019R7 | Asynchronous Routine | BT1 | — |
| IDA019R8 | IDA019R8 | Control-Interval Processing | BM1, BN1, BN2, BN3 | — |
| IDA019R9 | IDA019R9 | Page-Fix and Start-I/O Appendages | BT1 | — |
| IDA019SA | IDA019SA | Control-Interval Initialization-Create Entry-Sequenced Data Set | BE1, BF1, 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, DH1 | 8, 9 |
| IDA0192B | IDA0192B | Open a Cluster | AC4, AC5, AC6 | |
| IDA0192C | IDA0192C | VSAM Open/Close: Catalog Interface | AC2,AC4, AC5, AC6, AD5, AD6, AE1, DD1, DE1 | 9, 13, 15, 16, 17 |
| IDA0192D | IDA0192D | Stage/Destage (ACQUIRE/RELINQUISH) | AD5 | 9, 13, 15, 17 |
| IDA0192F | IDA0192F | Open Base Cluster, Path, and Upgrade Alternate Index | AC3, AC4, AC5, AC6 | 9 |
| IDA0192G | IDA0192G | Data Space Security Verification | — | 16 |
| IDA0192I | IDA0192I | ISAM Interface: Open Processing | AC1, AC7 | — |
| IDA0192M | IDA0192M | Virtual-Storage Management | — | 9, 11, 18 |
| IDA0192P | IDA0192P | VSAM Open/Close/EOV: Problem Determination | AD5 | 9, 13, 15 17 |
| IDA0192S | IDA0192S | VSAM Open/Close/EOV: SMF Record Build | AC7 | 9, 13, 15, 17 |
| IDA0192V | IDA0192V | Volume Mount and Verify | AC3 | 9, 17 |

## External Procedure Directory

| Procedure Name | Module Name | Descriptive Name | Method of Operation Diagrams | Program Organization Figures |
|---|---|---|---|---|
| IDA0192W | IDA0192W | Channel Program Area Build | AC1, AC4, AC5, AC6 | 9, 11 |
| IDA0192Y | IDA0192Y | String Build and Shared-Resource Processor | AC1, AC5, AF1 | 9, 11, 18 |
| IDA0192Z | IDA0192Z | Control Block Build | AC4, AC5, AC6 | 9 |
| IDA0200B | IDA0200B | Close a Cluster | AD2, AD3, AD5 | 13 |
| IDA0200S | IDA0200S | ISAM Interface: Close Processing | AD1, AD6 | 12 |
| IDA0200T | IDA0200T | VSAM Close String | AD1, AD2, AD3 AD4, AD6, DH1 | 13 |
| IDA0231B | IDA0231B | Close (TYPE=T) a Cluster | — | 15 |
| IDA0231T | IDA0231T | VSAM Close (TYPE=T) String | — | 15 |
| IDA0557A | IDA0557A | VSAM End of Volume | AE1, AE2, DE1, DH1 | 17 |
| IEFAB410 | IEFAB410 | Private (VSAM User's) Catalog Open | — | — |
| IEFAB411 | IEFAB411 | Private (VSAM User's) Catalog Close | — | — |
| IEFNB902 | IEFNB902 | AMP Parameter Interpreter | — | — |
| IFG0191X | IFG0191X | Catalog Open ACB Processor, Load 1 | — | 10 |
| IFG0191Y | IFG0191Y | Catalog Open ACB Processor, Load 2 | — | 9, 10 |
| IFG0192A | IFG0192A | VSAM Open/Close/EOV String Load | — | 8, 9, 12, 13, 15, 17 |
| IFG0192Z | IFG0192Z | VSAM Catalog Open ACB Processor | — | — |
| IFG0200N | IFG0200N | VSAM Catalog Close ACB Processor | — | 12, 13 |
| IFG0550Y | IFG0200N | VSAM Catalog EOV ACB Processor | — | 16 |
| IGGPACDV | IGG0CLAB | Catalog Management Driver | DB1, DD1, DE1, DH1 | 40 |
| IGGPADGO | IGG0CLAW | Add Group Occurrence (Set of Fields) | DM1 | 42, 45 |
| IGGPALEC | IGG0CLBE | Check for Index or Data and Sequence Set with Data | — | — |
| IGGPALT | IGG0CLBD | CMS ALTER Processing | EB1, EH1, EH2, EH3 | 50 |
| IGGPALT2 | IGG0CLAX | Alter Catalog Record's Field Value | DH1, DM2 | 42, 46 |
| IGGPALVL | IGG0CLBE | CMS ALTER: Volume Processing | EH3 | 50 |
| IGGPALVR | IGG0CLBN | CMS ALTER: Volume Processing | EH3 | 50 |
| IGGPAOCI | IGG0CLAG | Assign Contiguous Control Intervals | DM2, ED3, ED5 | 51.5 |
| IGGPAUPG | IGG0CLB2 | ALTER: Upgrade/Update | — | — |
| IGGPAXCI | IGG0CLAG | Assign One Control Interval | — | 45, 48, 51.6 |
| IGGPBJFB | IGG0CLA6 | Build JFCB | EG1 | — |
| IGGPBMR | IGG0CLBR | Bit-Map Manipulation Routine | DJ2 | 48, 50 |
| IGGPCBPT | IGG0CLA6 | Compute Blocks of Track Value | — | — |
| IGGPCCCR | IGG0CLAG | Checkpoint the Catalog Control Record (CCR) | — | 51.7 |
| IGGPCDVR | IGG0CLAT | CMS (Catalog Management Services): Initial Processing | DB1, DD1, EB1 | 40, 50 |
| IGGPCKAU | IGG0CLBM | Check the Caller's Authorization To Access the Catalog Record | DB1, DD1 | 40, 50 |

## External Procedure Directory

| Procedure Name | Module Name | Descriptive Name | Method of Operation Diagrams | Program Organization Figures |
|---|---|---|---|---|
| IGGPCMKY | IGG0CLB0 | Key-Range Verification | — | — |
| IGGPCNBO | IGG0CLB8 | Remove Candidate Volume Occurrences | — | — |
| IGGPCONV | IGG0CLBZ | CONVERTV Processing | EB1, EM1 | 49 |
| IGGPCPLR | IGG0CLAG | Check Preformat of Low-Key Range | — | — |
| IGGPCRTC | IGG0CLA6 | Convert Records to Tracks | EG1 | — |
| IGGPDAVO | IGG0CLBH | NonVSAM Volume Occurrence Construction | — | — |
| IGGPDBVC | IGG0CLAM | CMS DEFINE: Validity-check control blocks | DG5, EC1 | 50 |
| IGGPDBDI | IGG0CLAJ | CMS DEFINE: Build the Data Set and Index Catalog Records of a Cluster | ED1, ED2, ED3, ED4 | 51 |
| IGGPDCCE | IGG0CLAN | CMS DEFINE: Build the Cluster's Catalog Record | ED1, ED3, EE1 | — |
| IGGPDCCO | IGG0CLB0 | Define Data and Index Characteristics | — | 51 |
| IGGPDCDA | IGG0CLAP | CMS DEFINE CATALOG Processing (2nd of 2) | EC1, EE1 | 51 |
| IGGPDCIM | IGG0CLBX | Integer Conversion | — | — |
| IGGPDCLS | IGG0CLB5 | DELETE Cluster/AIX Records | EJ3, EJ4 | 50 |
| IGGPDCMB | IGG0CLAK | CMS DEFINE: Completion (Build the Volume Information Sets of Fields) | ED2 | 51 |
| IGGPDCME | IGG0CLAE | CMS DEFINE CATALOG: Catalog Open, Build, and Close | EE2 | 51 |
| IGGPDCRA | IGG0CLB4 | DEFINE CRA | EE1, EE3 | 51 |
| IGGPDCRC | IGG0CLAS | RBA Computing Routine | EE2 | 51 |
| IGGPDEAX | IGG0CLB5 | Explicit DELETE AIX | EJ1, EJ3 | 50 |
| IGGPDEF | IGG0CLAL | CMS DEFINE: Initial Processing | EB1, EB2, EC1 | 50, 51 |
| IGGPDEFA | IGG0CLBH | CMS DEFINE NONVSAM Processing | EB1, EF1 | 50 |
| IGGPDEFC | IGG0CLAS | CMS DEFINE CATALOG Processing (1st of 2) | EE1 | 51 |
| IGGPDEFS | IGG0CLAQ | CMS DEFINE SPACE Processing | EB1, ED1, ED3 EE1, EG1, EG2 | 50, 51 |
| IGGPDEIN | IGG0CLBW | Modify: Delete/Insert Processing | — | 46 |
| IGGPDEL | IGG0CLBG | CMS DELETE CLUSTER/AIX/PATH/NONVSAM Processing | EB1, EJ1, EJ4, 50 EJ3 | |
| IGGPDELC | IGG0CLAF | CMS DELETE CATALOG Processing | EB1, EL1 | 50 |
| IGGPDELS | IGG0CLBL | CMS DELETE SPACE Processing | EB1, EK1 | 50 |
| IGGPDEL2 | IGG0CLAV | Delete a Set of Fields | DM3 | 42, 47 |
| IGGPDEMV | IGG0CLA7 | CMS DELETE: Extract the volume information set of fields | — | 50 |
| IGGPDEPT | IGG0CLB5 | Explicit DELETE PATH | EJ1, EJ3 | 50 |
| IGGPDEXA | IGG0CLBG | Extract Associations | — | — |
| IGGPDFBO | IGG0CLB8 | CMS DEFINE: Space Recovery | — | 51 |
| IGGPDFRS | IGG0CLA8 | Free Unused and Unneeded Storage Resources | ED2, ED4 | — |
| IGGPDFS2 | IGG0CLA6 | CMS DEFINE SPACE: Build the Data Space Group and Data Set Directory Entry Sets of fields | — | — |
| IGGPDF4T | IGG0CLA7 | Format-4 Volume Record Timestamp | EJ2 | — |

## External Procedure Directory

| Procedure Name | Module Name | Descriptive Name | Method of Operation Diagrams | Program Organization Figures |
|---|---|---|---|---|
| IGGPDGO | IGG0CLAX | Modify: Delete Group Occurrence (Set of Fields) Processing | — | 47, 48 |
| IGGPDGOP | IGG0CLAX | Modify: Delete Group Occurrence Pointer (Set of Fields Pointer) Processing | — | 47, 48 |
| IGGPDIAX | IGG0CLB5 | Implicit DELETE AIX | EJ1, EJ2, EJ3 | 50 |
| IGGPDIPT | IGG0CLB5 | Implicit DELETE PATH | EJ3, EJ4 | 50 |
| IGGPDLET | IGG0CLBL | Delete Volume Entry Records | — | — |
| IGGPDLXT | IGG0CLBG | DELETE Exit Routine | — | — |
| IGGPDLVM | IGG0CLBL | Build Volume Mount Interface | — | — |
| IGGPDOPN | IGG0CLBG | OPEN Determination | EJ1 | 50 |
| IGGPDRDA | IGG0CLAN | DEFINE AMDSB Processing | ED1, ED3 | 51 |
| IGGPDRSP | IGG0CLBY | CMS DEFINE CLUSTER Processing (5th Module) | ED3 | 51 |
| IGGPDSCB | IGG0CLAN | CMS DEFINE: Initial Processing (Space Calculations and Build the Cluster Catalog Record) | EC1 | 51 |
| IGGPDSPC | IGG0CLBX | CMS DEFINE CLUSTER Processing (4th Module) | ED3 | 51 |
| IGGPDTIM | IGG0CLAL | CMS DEFINE: Call the System Timer | — | 50 |
| IGGPDUND | IGG0CLB8 | CMS DEFINE: Undo the Previous Processing | — | 50 |
| IGGPDUCB | IGG0CLBL | — | — | — |
| IGGPDUPG | IGG0CLB5 | DELETE Upgrade Associations | EJ3, EJ4 | 50 |
| IGGPDUSC | IGG0CLA7 | CMS DELETE: Scratch the Data Space (Format 1—Identifier—DSCB) from the Volume's VTOC | — | 50 |
| IGGPDVMV | IGG0CLA7 | CMS DELETE: Mount and Verify Volumes | — | 50 |
| IGGPEMIO | IGG0CLAF | I/O Error Message Writer | — | — |
| IGGPEMSG | IGG0CLAF | Error Message Writer | — | — |
| IGGPEXPD | IGG0CLAX | Expand a Catalog Record's Variable-length Field | DM2 | 46 |
| IGGPEXT | IGG0CLAZ | Extract Processing | DE1, DE2, DI3, DI4, DJ2 | 41, 49 |
| IGGPFDSP | IGG0CLAI | DELETE SPACE Processing | EK1 | 50 |
| IGGPF4DQ | IGG0CLBU | Dequeue the Format 4 DSCB | — | 50 |
| IGGPF4RD | IGG0CLBU | Read the Format 4 DSCB | EE3, EM1 | 50 |
| IGGPF4WR | IGG0CLBU | Write the Format 4 DSCB | EE3, EM1 | 50 |
| IGGPGALO | IGG0CLBZ | CONVERTV: GET all Low-Range Records | EM1 | 50 |
| IGGPGDSP | IGG0CLBJ | LOCATE: GENDSP Processing | DB1, DF1 | 40 |
| IGGPGET | IGG0CLBI | GET: Call Record Management to Retrieve A Catalog Record | DC2, DI4, DI6, DJ1 | 51.1 |
| IGGPGETC | IGG0CLBI | GET: Call Record Management to Retrieve a Catalog Record | — | 51.1 |
| IGGPGNEX | IGG0CLAW | Format a New Catalog Extension Record | — | 45, 48 |
| IGGPGREC | IGG0CLBA | Retrieve a Catalog Record | DL1, DL2 | 41, 42, 45, 47, 48 |

## External Procedure Directory

| Procedure Name | Module Name | Descriptive Name | Method of Operation Diagrams | Program Organization Figures |
|---|---|---|---|---|
| IGGPGVAL | IGG0CLBA | Get A Catalog Record Field | DE2, DL1, DL2, DM2 | 41, 42, 46 |
| IGGPIGOP | IGG0CLAW | Insert A Group Occurrence Pointer (Set of Fields Pointer) | DM1 | 45, 48 |
| IGGPINIT | IGG0CLBC | UPDATE-Extend: Initialization | DI1 | 43 |
| IGGPINMD | IGG0CLB6 | Build User Authorization Interface | DD2 | — |
| IGGPIORA | IGG0CLAG | Decode record management I/O error codes | — | 51.8 |
| IGGPISCI | IGG0CLAG | Insure Catalog Control Interval Availability | — | 51.9 |
| IGGPLDCS | IGG0CLBK | LSPACE: Gather the Available Space Data | DK1 | — |
| IGGPLOC | IGG0CLAZ | LOCATE Processing | DB1, DE1, DE2 | 40, 41 |
| IGGPLSP | IGG0CLBK | LSPACE Processing | DB1, DK1, EG2, EJ3 | 40, 50 |
| IGGPLSTC | IGG0CLBQ | CMS LISTCAT Processing | EB1, EI1 | 50 |
| IGGPMCO | IGG0CLAC | DEFINE CATALOG: Master Catalog Build and Open (1st of 2) | — | 40 |
| IGGPMCO2 | IGG0CLAD | DEFINE CATALOG: Master Catalog Build and Open (2nd of 2) | EE2 | 40, 50 |
| IGGPMCRA | IGG0CLA7 | DELETE: Verify CRA Volume Mount | — | 50 |
| IGGPMEBM | IGG0CLAE | Handle Multiple Extents for Catalog Open and Build | — | — |
| IGGPMGO | IGG0CLAX | Move Group Occurrences (Sets of Fields) From One Extension Record Into Another | — | 45, 48 |
| IGGPMOD | IGG0CLAV | Modify: Initial Processing | DI4, DI6, DJ1, DM1, DM3, EH3, EM1 | 42 |
| IGGPMODI | IGG0CLDA | Build Interface to MODIFY Function | EE3 | — |
| IGGPORDA | IGG0CLAN | DEFINE Processing | — | 51 |
| IGGPPAD | IGG0CLAG | PUT Add: Call Record Management To Write A New Catalog Record | DH1 | 51.3 |
| IGGPPAIX | IGG0CLB9 | DEFINE AIX | ED3 | 51 |
| IGGPPATH | IGG0CLB9 | DEFINE PATH | ED5 | 51 |
| IGGPPDE | IGG0CLAG | ERASE: Call Record Management To Erase A Catalog Record | — | 47, 51.4 |
| IGGPPREC | IGG0CLAW | Call PUT-Add or PUT-Update to Write a Catalog Record, Then Call SMF | DH1 | 42, 47, 50 |
| IGGPPSEM | IGG0CLAN | DEFINE: Validity-Check Input Parameters | — | 51 |
| IGGPPUPC | IGG0CLAG | PUT-Update: Call Record Management to Rewrite A Catalog Record | DH1 | 51.2 |
| IGGPRACC | IGG0CLBO | Checkpoint CRA CCR | — | — |
| IGGPRAEA | IGG0CLBO | CRA I/O Error Analysis | — | 51.19 |
| IGGPRAG | IGG0CLBO | CRA GET | — | — |
| IGGPRAOP | IGG0CLAD | Open CRA | — | 51.15 |
| IGGPRAOR | IGG0CLBO | Orient to CRA | — | 51.14 |
| IGGPRAPA | IGG0CLBO | CRA PUT Add | — | 51.12 |

## External Procedure Directory

| Procedure Name | Module Name | Descriptive Name | Method of Operation Diagrams | Program Organization Figures |
|---|---|---|---|---|
| IGGPRAPC | IGG0CLBO | CRA Record Chain | — | — |
| IGGPRAPD | IGG0CLBO | CRA PUT Delete | — | 51.13 |
| IGGPRAPU | IGG0CLBO | CRA PUT Update | — | 51.11 |
| IGGPRAPV | IGG0CLDA | Format-4 DSCB Timestamp Processing | — | — |
| IGGPRARC | IGG0CLBO | Read CRA CCR | — | — |
| IGGPRASC | IGG0CLBO | Ensure Sufficient CRA CIs | — | 51.17 |
| IGGPRAX | IGG0CLBO | CRA Exit Routine | — | 51.18 |
| IGGPRCCR | IGG0CLAG | Read Catalog Control Record | EM1 | 51.7,51.10 |
| IGGPRET1 | IGG0CLBP | Return point from DADSM EXTEND | — | — |
| IGGPRPLF | IGG0CLAH | Dequeue the Catalog | EM1 | — |
| IGGPRPLM | IGG0CLAH | RPL Manager | DC2 | — |
| IGGPRUS | IGG0CLB7 | Reset a Reusable Data Set | DE1, DH1, DI3, DI4, DI5, DI6 | 44 |
| IGGPSALL | IGG0CLAR | Suballocate: Candidate Volume Assignment | DI1, DJ1, ED1, ED3, EE1, EH3 | 43, 50, 51 |
| IGGPSALS | IGG0CLAU | Suballocate: Space Assignment | DJ1, DJ2, EJ2 | 43, 49, 51 |
| IGGPSCAT | IGG0CLAH | Search Catalog Processing | DB1, DC1, DC2, EG1, EM1 | 40,50 |
| IGGPSCNC | IGG0CLAY | Initial CTGPL Processing | DB1, DE1, DE2, DM1 | 40, 41, 42 |
| IGGPSCNF | IGG0CLAZ | — | DE1, DE2 | — |
| IGGPSGOP | IGG0CLAV | Search For a Group Occurrence Pointer (Set of Fields Pointer) | DE1, DE2 | 47, 48 |
| IGGPSHNK | IGG0CLAX | Shrink A Catalog Record's Variable-Length Field | DM2 | 46 |
| IGGPSLEL | IGG0CLAM | SUPERLOCATE Processing | DG2, DG3 | 40 |
| IGGPSLEN | IGG0CLAA | SUPERLOCATE Processing | DG1, DG2, DG3 | 40 |
| IGGPSLOC | IGG0CLAM | LOCATE: Super-Locate Processing | DB1, DG1, DG2, DG3, DG4 | 40 |
| IGGPSMF | IGG0CLB3 | SMF-for-ALTER: Write Processing | — | — |
| IGGPSMFA | IGG0CLBV | SMF: Build a Format-63 SMF Record | ED2, ED4, ED5, EE2 | — |
| IGGPSMFF | IGG0CLBV | SMF: Free the Virtual Storage Obtained | DK1 | — |
| IGGPSMFG | IGG0CLB3 | SMF-for-ALTER: Read Processing | — | — |
| IGGPSMFL | IGG0CLBV | SMF: Build a Format-69 SMF Record | DK1 | — |
| IGGPSMFR | IGG0CLBV | SMF: Build a Format-68 SMF Record | — | — |
| IGGPSMFS | IGG0CLBV | SMF: Build a Format-67 SMF Record | — | — |
| IGGPSPAC | IGG0CLBP | UPDATE-Extend: Obtain More Space | — | 43 |
| IGGPSPSC | IGG0CLB6 | Check for Security By-Pass | DD1 | — |
| IGGPSSCR | IGG0CLBF | Subscratch: Release A Cluster's Space Within a VSAM Data Space | DI5, EJ1 | 50 |
| IGGPSVOL | IGG0CLBC | Search For the Volume Information Set-of-Fields | DI1 | 43 |
| IGGPTNXO | IGG0CLBI | Calculate SUMTT Value | — | — |

## External Procedure Directory

| Procedure Name | Module Name | Descriptive Name | Method of Operation Diagrams | Program Organization Figures |
|---|---|---|---|---|
| IGGPTSTS | IGG0CLBA | CTGFL-for-Tests Processing | DE1, DH1 | 41, 42 |
| IGGPTXO | IGG0CLBI | Update SUMTT in Volume Record | — | — |
| IGGPUADD | IGG0CLB1 | Add Upgrade Association | — | — |
| IGGPUCRS | IGG0CLBI | GET a CRA Record by CI Number | — | — |
| IGGPUDEL | IGG0CLB1 | Delete Upgrade Association | — | — |
| IGGPUPD | IGG0CLAV | UPDATE: Initial Processing | DB1, DH1, DI3 | 40, 42 |
| IGGPUPDE | IGG0CLBB | UPDATE-Extend Processing | DH1, DI1, DI2 | 42, 43 |
| IGGPUPGD | IGG0CLAZ | PUT Upgrade Processing | DE1 | — |
| IGGPVALI | IGG0CLBZ | CONVERTV: Validity-Checking | EM1 | 50 |
| IGGPVRD | IGG0CLBN | ALTER: Remove VSAM Space from Volume | EH3 | 50 |
| IGGPVMSC | IGG0CLA7 | CMS DELETE: Delete all space information in the volume catalog record | — | 50 |
| IGGPWCAT | IGG0CLDA | Write Volume Records to Catalog | EE3 | — |
| IGGPWCRA | IGG0CLDA | Write Self-Describing and Volume Records to CRA | EE3 | — |
| IGGPWFLR | IGG0CLAG | Write Free Record | — | — |
| IGGPWTSO | IGG0CLB6 | Write Messages To A TSO Terminal User | — | 40, 50 |
| IGGPXDGO | IGG0CLBT | Add Derived Group Occurrence | DM1 | — |
| IGGPXEL2 | IGG0CLBT | Delete Derived Group Occurrence | DM3 | — |
| IGGPXEXT | IGG0CLBS | Extract Derived Group Occurrence | — | — |
| IGGPXIO | IGG0CLAG | Issue the GET, PUT, or ERASE Macro Instructions | — | 51.8 |
| IGGPXLT2 | IGG0CLBT | Alter Derived Field Value | DM2 | — |
| IGGPXMOD | IGG0CLBT | Modify Derived Group Occurrence | DM1 | — |
| IGGPXRIO | IGG0CLBO | CRA GET/PUT Routine | — | 51.19 |
| IGGPXVAL | IGG0CLBS | Get Derived Field Value | DL1 | — |
| IGG0CLA1 | IGG0CLA1 | Catalog Management Transient Load | DB1 | 40 |
| IGG0CLCA | IGG0CLCA | OS/VS Catalog Request Handler | — | — |
| IGG0CLCB | IGG0CLCB | VSAM Catalog Request Handler | — | — |
| IGG0CLC9 | IGG0CLC9 | Catalog Management First Load | DB1 | 40 |
| IGG0102G | IGG0102G | OS/VS DADSM: Obtain Space For a VSAM Data Space | — | — |
| IMDUSRF9 | IMDUSRF9 | IMDPRDMP-Format Appendage | — | — |

# Procedure Calls Directory

## *Procedure Calls Directory: Open/Close/EOV Modules*

This table lists each Open/Close/EOV module and the modules it calls.

| Calling Module | Called Modules |
|---|---|
| IDA0192A | IDA0192B, IDA0192C, IDA0192F, IDA0192M, IDA0192P, IDA0192S |
| IDA0192B | IDA0192C, IDA0192D, IDA0192M, IDA0192P, IDA0192Y, IDA0192Z |
| IDA0192C | — |
| IDA0192D | IDA0192P |
| IDA0192F | IDA0192M, IDA0192P, IDA0192V |
| IDA0192G | IDA0192C |
| IDA0192I | — |
| IDA0192M | — |
| IDA0192P | — |
| IDA0192S | — |
| IDA0192V | IDA0192C, IDA0192D |
| IDA0192W | IDA0192M |
| IDA0192Y | IDA0192M, IDA0192W |
| IDA0192Z | IDA0192M, IDA0192Y |
| IDA0200S | — |
| IDA0200B | IDA0192C, IDA0192D, IDA0192P, IDA0192S |
| IDA0200T | IDA0192C, IDA0192P, IDA0192Y, IDA0200B |
| IDA0231B | IDA0192C, IDA0192D, IDA0192P, IDA0192S |
| IDA0231T | IDA0192P, IDA0231B |
| IDA0557A | IDA0192C, IDA0192D, IDA0192P, IDA0192S, IDA0192V |

## *Procedure Calls Directory: Checkpoint/Restart*

| Calling Module | Called Modules |
|---|---|
| IGC0A05B | IDA0A05B |
| IDA0A05B | IDA0B05B, IDA0192M |
| IDA0B05B | — |
| IDA0C06C | — |

## *Procedure Calls Directory: Record Management Modules*

This table lists each Record Management module and the modules and procedures it calls.

| Calling Module | Called Procedure (and its Module) |
|---|---|
| IDA019RA | IDAABF (IDA019RW), IDAFRBA (IDA019RW), IDAFREEB (IDA019RZ), IDAGNXT (IDA019RZ), IDAGRB (IDA019RZ), IDASBF (IDA019RZ), IDAWAIT (IDA019RZ), IDA019RB |
| IDA019RB | IDAFREEB (IDA019RZ), IDAGRB (IDA019RZ), IDA019RC |
| IDA019RC | — |

| Calling Module | Called Procedure (and its Module) |
|---|---|
| IDA019RE | IDAFREEB (IDA019RZ), IDAGNFL (IDA019RZ), IDAGRB (IDA019RZ), IDASBF (IDA019RZ), IDATJXIT (IDA019RP), IDAWRBFR (IDA019RZ), IDA019RF, IDA019RH, IDA019RM |
| IDA019RF | IDAABF (IDA019RW), IDAAIBF (IDA019RW), IDAEOVIF (IDA019R5), IDAFREEB (IDA019RZ), IDAGNNFL (IDA019RZ), IDAGNXT (IDA019RZ), IDAGRB (IDA019RZ), IDASBF (IDA019RZ), IDATJXIT (IDA019RP), IDAWRBFR (IDA019RZ), IDA019RI, IDA019RK, IDA019RM, IDA019SF |
| IDA019RG | IDAAQR (IDA019RN), IDAEOVIF (IDA019R5), IDAER (IDA019RN), IDAFREEB (IDA019RZ), IDAGNFL (IDA019RZ), IDAR (IDA019RJ), IDASPNPT (IDA019RT), IDAWR (IDA019RJ) |
| IDA019RH | IDAWRBFR (IDA019RZ), IDA019RC |
| IDA019RI | IDAAQR (IDA019RN), IDAEOVIF (IDA019R5), IDAFREEB (IDA019RZ), IDAGNFL (IDA019RZ), IDAGRB (IDA019RZ), IDA019RB, IDA019RH, IDA019RJ |
| IDA019RJ | IDAAQR (IDA019RN), IDAER (IDA019RN), IDAFREEB (IDA019RZ), IDAGRB (IDA019RZ), IDAIST (IDA019RG), IDAIVIXB (IDA019RH), IDASPACE (IDA019RH), IDASPNPT (IDA019RT), IDAWRBFR (IDA019RZ) |
| IDA019RK | IDAEOVIF (IDA019R5), IDAFREEB (IDA019RZ), IDAGNFL (IDA019RZ), IDAWRBFR (IDA019RZ) |
| IDA019RL | IDATJXIT (IDA019RP), IDA019RM, IDA019RS |
| IDA019RM | IDATJXIT (IDA019RP), IDA019RE, IDA019RT, IDA019RU, IDA019SA |
| IDA019RN | IDAEOVIF (IDA019R5), IDA019RK |
| IDA019RO | IDAFREEB (IDA019RZ), IDAGRB (IDA019RZ), IDA019RB |
| IDA019RP | IDAEXEX (IDA019R5), IDAEXITR (IDA019R5), IDASBF (IDA019RZ), IDAWRBFR (IDA019RZ), IDA019RG, IDA019RK, IDA019R5, IDA019SA |
| IDA019RQ | IDAABF (IDA019RW), IDADRQ (IDA019R5), IDAEOVIF (IDA019R5), IDAFREEB (IDA019RZ), IDAGNNFL (IDA019RZ), IDAGNXT (IDA019RZ), IDARRDRL (IDA019RR), IDATJXIT (IDA019RP), IDAWRBFR (IDA019RZ), IDA019RK |
| IDA019RR | IDAABF (IDA019RW), IDAFREEB (IDA019RZ), IDAGNXT (IDA019RZ), IDAGRB (IDA019RZ), IDATJXIT (IDA019RP), IDAWRBFR (IDA019RZ), IDA019RQ |
| IDA019RS | IDACHKKR (IDA019RM), IDADRQ (IDA019R5), IDAGRB (IDA019RZ), IDAGWSEG (IDA019RZ), IDAIVIXB (IDA019RH), IDAJRNSR (IDA019RT), IDAREPOS (IDA019RE), IDASBF (IDA019RZ), IDAWRBFR (IDA019RZ), IDA019RC, IDA019RF |
| IDA019RT | IDAABF (IDA019RW), IDAADSEG (IDA019RS), IDAAIBF (IDA019RW), IDADRQ (IDA019R5), IDAFREEB (IDA019RZ), IDAGNXT (IDA019RZ), IDAGRB (IDA019RZ), IDAIVIXB (IDA019RH), IDAMVSEG (IDA019RS), IDAREPOS (IDA019RE), IDASBF (IDA019RZ), IDATJXIT (IDA019RP), IDAWRBFR (IDA019RZ), IDA019RC, IDA019RE, IDA019RF, IDA019SA |
| IDA019RU | IDADRQ (IDA019R5), IDAGETWS (IDA019RX), IDARELWS (IDA019RX), IDASBF (IDA019RZ), IDA019R4 |
| IDA019RV | IDAABF (IDA019RW), IDAFREEB (IDA019RZ), IDAGNXT (IDA019RZ), IDAGRB (IDA019RZ), IDA019RA, IDA019RC |
| IDA019RW | IDAGRB (IDA019RZ), IDAWAIT (IDA019RZ), IDAWRBFR (IDA019RZ), IDA019RC, IDA019RV, IDA019R3 |
| IDA019RX | IDADRQ (IDA019R5), IDAENDRQ (IDA019RP), IDAXGPLH (IDA019RU), IDA019R4 |

| Calling Module | Called Procedure (and its Module) |
|---|---|
| IDA019RY | IDADRQ (IDA019R5), IDAEXEX (IDA019R5), IDAFRBA (IDA019RW), IDATJXIT (IDA019RP), IDAWAIT (IDA019RZ), IDA019R3, IDA019R5 |
| IDA019RZ | IDAGWSGW (IDA019RW), IDA019RY, IDA019R2, IDAUPXIT |
| IDA019R1 | IDAENDRQ (IDA019RP), IDAERROR (IDA019R5), IDAFREEB (IDA019RZ), IDAMRKBF (IDA019RZ), IDARSTRT (IDA019R5), IDASBF (IDA019RZ), IDASCHBF (IDA019RZ), IDAWRBFR (IDA019RZ), IDAWRTBF (IDA019RZ), IDA019RK, IDA019RR, IDA019RX, IDA019R4, IDA019R5, IDA019R8 |
| IDA019R2 | IDAFRBA (IDA019RW), IDARVRS1 (IDA019RV), IDAWAIT (IDA019RZ), IDA019R3 |
| IDA019R3 | IDADRQ (IDA019R5), IDAEOVIF (IDA019R5), IDA019SB |
| IDA019R4 | IDAABF (IDA019RW), IDAADVPH (IDA019RV), IDADARTV (IDA019RT), IDADRQ (IDA019R5), IDAFREEB (IDA019RZ), IDAGNNFL (IDA019RZ), IDAGNXT (IDA019RZ), IDAGRB (IDA019RZ), IDARXBD (IDA019RX), IDATJXIT (IDA019RP), IDAWRBFR (IDA019RZ), IDA019RA, IDA019RK, IDA019RL, IDA019RM, IDA019RU |
| IDA019R5 | IDAUPXIT |
| IDA019R6 | — |
| IDA019R7 | — |
| IDA019R8 | IDAABF (IDA019RW), IDADRQ (IDA019R5), IDAEOVIF (IDA019R5), IDAEXCL (IDA019RZ), IDAFRBA (IDA019RW), IDAFREEB (IDA019RZ), IDAGNNFL (IDA019RZ), IDAGNXT (IDA019RZ), IDAGRB (IDA019RZ), IDASBF (IDA019RZ), IDATJXIT (IDA019RP), IDAWRBFR (IDA019RZ), IDA019RK, IDA019RO |
| IDA019R9 | — |
| IDA019SA | IDAEOVIF (IDA019R5), IDAFREEB (IDA019RZ), IDAGNNFL (IDA019RZ), IDAGRB (IDA019RZ), IDATJXIT (IDA019RP), IDAWRBFR (IDA019RZ), IDA019RG, IDA019RK, IDA019RU |
| IDA019SB | IDADRQ (IDA019R5) |
| IDA019SF | IDAABF (IDA019RW), IDAEOVIF (IDA019R5), IDANEWRD (IDA019RI), IDAFREEB (IDA019RZ), IDAGNNFL (IDA019RZ), IDAGRB (IDA019RZ), IDAHLINS (IDA019RI), IDASBF (IDA019RZ), IDAWRBFR (IDA019RZ), IDA019RM, IDA019RV |

## *Procedure Calls Directory: Catalog Management Procedures*

This table contains each Catalog Management module and external and internal procedures within the module. The internal and external calls of each procedure are listed.

| Calling Module and Its Procedures | External Procedures Called (Procedures Outside the Module) | Internal Procedures Called (Procedures Within the Module) |
|---|---|---|
| IGG0CLAA | | |
| **External** | | |
| IGGPSLEN | IGGPSLEL | IGGPSLCG, IGGPSLIV, IGGPSLR, IGGPSLY |
| **Internal** | | |
| IGGPSLCG | IGGPEXT, IGGPGET, IGGPSLEL | IGGPSLY |
| IGGPSLIV | — | — |
| IGGPSLR | IGGPEXT, IGGPGET, IGGPSLEL | IGGPSLY |
| IGGPSLY | IGGPEXT, IGGPGET, IGGPSLEL | — |
| IGG0CLAB | | |
| **External** | | |
| IGGPACDV | IGGPCDVR, IGGPCKAU, IGGPGDSP, IGGPLOC, IGGPLSP, IGGPMCO, IGGPSCAT, IGGPSCNC, IGGPSLOC, IGGPUPD | — |
| **Internal** | | |
| — | | |
| IGG0CLAC | | |
| **External** | | |
| IGGPMCO | IGGPMCO2 | — |
| **Internal** | | |
| IGGPSUCB | — | — |
| IGG0CLAD | | |
| **External** | | |
| IGGPMCO2 | — | — |
| IGGPRAOP | — | IGGPCRAD |
| IGG0CLAE | | |
| **External** | | |
| IGGPDCME | IGGPDCRC, IGGPMOD | IGGPDCOC |
| IGGPMEBM | IGGPBMR | — |
| **Internal** | | |
| IGGPDCBO | IGGPDUND, IGGPF4RD, IGGPF4WR | — |
| IGGPDCCB | — | — |
| IGGPDCOC | IGGPMCO2, IGGPSMFA | IGGPDCCB, IGGPDCBO, IGGPDCPR |
| IGGPDCPR | IGGPXID | — |
| IGG0CLAF | | |
| **External** | | |
| IGGPDELC | IGGPCCCR, IGGPDBVC, IGGPEXT, IGGPFDSP, IGGPF4RD, IGGPF4WR, IGGPGET, IGGPPDE, IGGPSCAT | IGGPDLER, IGGPSDSP |

## Procedure Calls Directory: Catalog Management Modules

| Calling Module and Its Procedures | External Procedures Called (Procedures Outside the Module) | Internal Procedures Called (Procedures within the Module) |
|---|---|---|
| IGGPEMIO | — | — |
| IGGPEMSG | — | IGGPNFND |
| **Internal** | | |
| IGGPDCDS | IGGPDOPN, IGGPEXT | — |
| IGGPDLER | IGGPDF4T | — |
| IGGPNFND | — | — |
| IGGPSDSP | — | — |
| IGG0CLAG | | |
| **External** | | |
| IGGPAOCI | — | IGGPANCI, IGGPCCCR, IGGPRCCR, IGGPXIO |
| IGGPAXCI | — | IGGPANCI, IGGPCCCR, IGGPRCCR, IGGPXIO |
| IGGPCCCR | — | IGGPRBAP, IGGPXIO |
| IGGPIORA | IGGPEMIO, IGGPEMSG | — |
| IGGPISCI | IGGPGET, IGGPRASC | IGGPANCI, IGGPCCCR, IGGPRCCR, IGGPXIO |
| IGGPPAD | IGGPRAPA, IGGPRAP4 | IGGPRCCR, IGGPXIO |
| IGGPPADC | — | — |
| IGGPPDE | IGGPRAPD | IGGPRCCR, IGGPTRPL, IGGPXIO |
| IGGPPUPC | IGGPRAPU | IGGPTRPL, IGGPXIO |
| IGGPRCCR | — | IGGPRBAP, IGGPXIO |
| IGGPXIO | — | IGGPIORA |
| **Internal** | | |
| IGGPANCI | — | IGGPCHAC, IGGPIORA |
| IGGPCHAC | — | — |
| IGGPRBAP | — | — |
| IGGPTRPL | — | IGGPXIO |
| IGG0CLAH | | |
| **External** | | |
| IGGPSCAT | IGGPCKAU, IGGPGET, IGGPUCRS | IGGPIOSI, IGGPRPLF, IGGPRPLM, IGGPSCA |
| IGGPRPLF | — | — |
| IGGPRPLM | — | IGGPPLHC |
| **Internal** | | |
| IGGPIOSI | — | — |
| IGGPPLHC | — | — |
| IGGPSCA | — | — |

## Procedure Calls Directory: Catalog Management Modules

| Calling Module and Its Procedures | External Procedures Called (Procedures Outside the Module) | Internal Procedures Called (Procedures within the Module) |
|---|---|---|
| IGG0CLAI | | |
| **External** | | |
| IGGPDFM1 | IGGPF4DQ, IGGPF4RD | — |
| IGGPFDSP | IGGPDLET, IGGPDLVM, IGGPDOPN, IGGPDUCB, IGGPEXT, IGGPGET, IGGPMOD, IGGPPUPC | IGGPFDEX, IGGPFDXT |
| **Internal** | | |
| IGGPFDEX | — | — |
| IGGPFDXT | — | — |
| IGG0CLAJ | | |
| **External** | | |
| IGGPDBDI | IGGPDCMB, IGGPDFBO, IGGPDTIM, IGGPGET, IGGPUADD | IGGPDCNV, IGGPDFRS, IGGPDRCS, IGGPDSPO |
| **Internal** | | |
| IGGPDCNV | IGGPSALL | — |
| IGGPDEXD | — | — |
| IGGPDFRE | — | — |
| IGGPDFRS | — | IGGPDFRE |
| IGGPDRCS | — | — |
| IGGPDSEX | — | IGGPDEXD |
| IGGPDSPO | IGGPCNBO, IGGPDEFS, IGGPGET, IGGPSALL, IGGPTNXO, IGGPTXO | IGGPDSEX |
| IGG0CLAK | | |
| **External** | | |
| IGGPDCMB | — | IGGPDBVO, IGGPDMOP, IGGPDOMF |
| **Internal** | | |
| IGGPDBCV | — | IGGPDEXD |
| IGGPDBVO | — | IGGPDBCV, IGGPDEXD, IGGPDRCA, IGGPDRNG, IGGPDSSP |
| IGGPDEXD- | | — |
| IGGPDMOP | IGGPMOD | — |
| IGGPDOMF | IGGPSMFA | — |
| IGGPDRCA | — | — |
| IGGPDRNG | — | — |
| IGGPDSSP | — | IGGPDEXD |
| IGG0CLAL | | |
| **External** | | |
| IGGPDEF | IGGPDSCB | IGGPDCAV, IGGPDDEP |
| IGGPDBVC | — | — |
| IGGPDTIM | — | — |

## Procedure Calls Directory: Catalog Management Modules

| Calling Module and Its Procedures | External Procedures Called (Procedures Outside the Module) | Internal Procedures Called (Procedures within the Module) |
|---|---|---|
| **Internal** | | |
| IGGPDCAV | — | IGGPDBVC, IGGPDCSF, IGGPDCWC, IGGPDDNP, IGGPDFSC, IGGPDRPG |
| IGGPDCDE | — | — |
| IGGPDCSF | — | IGGPDBVC |
| IGGPDCWC | — | IGGPDBVC |
| IGGPDDEP | — | IGGPDCDE, IGGPDEDE, IGGPDSTY, IGGPDWAI |
| IGGPDDNP | — | — |
| IGGPDEDE | IGGPGET | IGGPDTIM |
| IGGPDFSC | — | IGGPDBVC |
| IGGPDRPG | IGGPCMKY | — |
| IGGPDSTY | — | IGGPDBVC |
| IGGPDWAI | — | — |
| IGG0CLAM | | |
| **External** | | |
| IGGPDVC | — | — |
| IGGPSLEL | IGGPEXT | — |
| IGGPSLOC | IGGPSLEN | IGGPSLIN |
| **Internal** | | |
| IGGPSLEI | — | — |
| IGGPSLIN | — | IGGPSLEI |
| IGG0CLAN | | |
| **External** | | |
| IGGPDCCE | IGGPAOCI, IGGPPAD | — |
| IGGPDRDA | IGGPDRSP, IGGPDSPC | IGGPDCCE |
| IGGPDSCB | IGGPDBDI, IGGPDCDA, IGGPDEFC | IGGPDALR, IGGPDBSF, IGGPDRDA, IGGPDSPF, IGGPDUND |
| IGGPPSEM | — | IGGPXDCI |
| **Internal** | | |
| IGGPDALR | — | — |
| IGGPDBSF | — | — |
| IGGPDSPF | — | — |
| IGG0CLAP | | |
| **External** | | |
| IGGPDCDA | — | IGGPDCVS |
| **Internal** | | |
| IGGPDCON | — | — |
| IGGPDCPC | IGGPAOCI, IGGPDCCE, IGGPPAD | — |
| IGGPDCSP | — | IGGPDCON, IGGPDCPC |

## Procedure Calls Directory: Catalog Management Modules

| Calling Module and Its Procedures | External Procedures Called (Procedures Outside the Module) | Internal Procedures Called (Procedures within the Module) |
|---|---|---|
| IGGPDCVS | — | IGGPDCSP |
| IGG0CLAQ | | |
| **External** | | |
| IGGPDEFS | IGGPAXI, IGGPBJFB, IGGPCBPT, IGGPCKAU, IGGPCRTC, IGGPDCRA, IGGPDFS2, IGGPDTIM, IGGPGET, IGGPISCI, IGGPLSP, IGGPPAD, IGGPSCAT | IGGPCOBT, IGGPDSXT, IGGPF1BO, IGGPF4PR, IGGPIVER, IGGPTMST, IGGPVMTV |
| **Internal** | | |
| IGGPCOBT | — | — |
| IGGPDSXT | — | — |
| IGGPF1BO | — | — |
| IGGPF4PR | IGGPF4DQ, IGGPF4RD, IGGPF4WR | — |
| IGGPIVER | — | — |
| IGGPTMST | IGGPF4RD, IGGPF4WR | — |
| IGGPVMTV | — | — |
| IGG0CLAR | | |
| **External** | | |
| IGGPSALL | IGGPEXT, IGGPGET, IGGPISCI, IGGPMOD, IGGPSALS | — |
| IGG0CLAS | | |
| **External** | | |
| IGGPDEFC | IGGPCCCR, IGGPDCME | IGGPDCBE, IGGPDCSP, IGGPDCVO |
| **Internal** | | |
| IGGPDCBE | — | IGGPDCEB, IGGPDCPB |
| IGGPDCEB | — | IGGPDCME |
| IGGPDCFL | — | — |
| IGGPDCHD | — | IGGPDCRC |
| IGGPDCIX | — | IGGPDCRC |
| IGGPDCLD | — | IGGPDCRC |
| IGGPDCPB | — | — |
| IGGPDCRC | — | — |
| IGGPDCSP | IGGPDEFS, IGGPSALL, IGGPMEBM | — |
| IGGPDCVO | — | IGGPDCFL, IGGPDCHD, IGGPDCIX, IGGPDCLD |
| IGG0CLAT | | |
| **External** | | |
| IGGPCDVR | IGGPALT, IGGPCKAU,IGGPCONV, IGGPDEF, IGGPDEFA, IGGPDEFS, IGGPDEL, IGGPDELC, IGGPDELS, IGGPLSTC, IGGPSCAT | IGGPCCLN |
| **Internal** | | |
| IGGPCCLN | — | — |

## Procedure Calls Directory: Catalog Management Modules

| Calling Module and Its Procedures | External Procedures Called (Procedures Outside the Module) | Internal Procedures Called (Procedures within the Module) |
|---|---|---|
| IGG0CLAU | | |
| **External** | | |
| IGGPSALS | IGGPBMR, IGGPEXT | IGGPEDS |
| **Internal** | | |
| IGGPEDS | IGGPBMR | — |
| IGG0CLAV | | |
| **External** | | |
| IGGPDEL2 | IGGPDGO, IGGPDGOP, IGGPGREC, IGGPPDE, IGGPPREC, IGGPSMF | IGGPSGOP |
| IGGPMOD | IGGPSCNC | IGGPSFPL |
| IGGPSGOP | IGGPGREC | — |
| IGGPUPD | IGGPRUS, IGGPUPDE | IGGPSFPL |
| **Internal** | | |
| IGGPSFPL | IGGPGREC, IGGPPREC, IGGPSMFG, IGGPTSTS, IGGPXDGO, IGGPXEL2, IGGPXLT2 | — |
| IGG0CLAW | | |
| **External** | | |
| IGGPADGO | IGGPAXCI, IGGPGREC | IGGPAGOP, IGGPASPT, IGGPGNEX, IGGPGREL, IGGPIGOP, IGGPMVGO |
| IGGPGNEX | — | IGGPPREC |
| IGGPGREL | IGGPGREC | — |
| IGGPIGOP | — | — |
| IGGPPREC | IGGPPAD, IGGPPUPC, IGGPSMF | — |
| **Internal** | | |
| IGGPAGOP | IGGPAXCI, IGGPGREC, IGGPMGO | IGGPASPT, IGGPGNEX, IGGPGREL, IGGPIGOP |
| IGGPASPT | — | IGGPIGOP |
| IGGPMVGO | — | — |
| IGG0CLAX | | |
| **External** | | |
| IGGPALT2 | IGGPGVAL | IGGPMVAR |
| IGGPDGO | — | — |
| IGGPDGOP | — | — |
| IGGPEXPD | — | — |
| IGGPMGO | IGGPIGOP, IGGPSGOP | IGGPCGO, IGGPDGO, IGGPDGOP |
| IGGPSHNK | — | — |
| **Internal** | | |
| IGGPCGO | — | — |
| IGGPMBGO | IGGPAXCI, IGGPGNEX, IGGPGREC | IGGPMGO |
| IGGPMVAR | IGGPAXCI, IGGPDEIN, IGGPGNEX, IGGPGREC, IGGPGVAL | IGGPEXPD, IGGPMBGO, IGGPMGO, IGGPSHNK |

## Procedure Calls Directory: Catalog Management Modules

| Calling Module and Its Procedures | External Procedures Called (Procedures Outside the Module) | Internal Procedures Called (Procedures within the Module) |
|---|---|---|
| IGG0CLAY | | |
| **External** | | |
| IGGPSCNC | — | IGGPSNVC |
| **Internal** | | |
| IGGPSNVC | — | — |
| IGG0CLAZ | | |
| **External** | | |
| IGGPEXT | IGGPSCNC, IGGPXEXT | IGGPSCNF |
| IGGPLOC | — | IGGPSCNF |
| **Internal** | | |
| IGGPSCNF | IGGPGREC, IGGPSMFG, IGGPTSTS | IGGPLOC2, IGGPUPGD |
| IGGPLOC2 | IGGPGVAL | IGGPGREP, IGGPSHIN |
| IGGPSHIN | — | — |
| IGGPGREP | IGGPGREC | — |
| IGGPUPGD | IGGPGET, IGGPVAL | — |
| IGG0CLA1 | | |
| **External** | | |
| IGG0CLA1 | IGG0CLC9 | — |
| IGG0CLA6 | | |
| **External** | | |
| IGGPBJFB | — | — |
| IGGPCBPT | — | — |
| IGGPCRTC | — | — |
| IGGPDFS2 | — | IGGPCDSD, IGGPCSDG, IGGPC5HG |
| **Internal** | | |
| IGGPCDSD | IGGPSALL | — |
| IGGPCSDG | — | IGGPDSMD |
| IGGPCSHG | — | IGGPDSMD |
| IGGPDSMD | IGGPMOD | — |
| IGG0CLA7 | | |
| **External** | | |
| IGGPDEMV | IGGPDLXT, IGGPEXT | IGGPSET |
| IGGPDF4T | IGGPDLXT, IGGPF4DQ, IGGPF4RD, IGGPF4WR, IGGPPUPC | — |
| IGGPDUSC | IGGPDLXT | — |
| IGGPDVMV | — | — |
| IGGPMCRA | IGGPDLXT, IGGPGET | IGGPDVMV |
| IGGPVMSC | IGGPGET, IGGPSSCR | IGGPDEDD, IGGPDESH, IGGPDEVG, IGGPDF4T, IGGPDUSC |

## Procedure Calls Directory: Catalog Management Modules

| Calling Module and Its Procedures | External Procedures Called (Procedures Outside the Module) | Internal Procedures Called (Procedures within the Module) |
|---|---|---|
| **Internal** | | |
| IGGPDEDD | IGGPMOD | IGGPSET |
| IGGPDESH | IGGPMOD | IGGPSET |
| IGGPDEVG | IGGPDLXT, IGGPEXT, IGGPMOD | IGGPDVMV, IGGPSET |
| IGGPSET | — | — |
| IGG0CLA8 | | |
| **External** | | |
| IGGPDFRS | — | — |
| **Internal** | | |
| IGGPDFRE | — | — |
| IGG0CLBA | | |
| **External** | | |
| IGGPGREC | IGGPGET, IGGPPREC, IGGPSMFG | — |
| IGGPGVAL | — | IGGPGREC, IGGPLVAL |
| IGGPTSTS | — | IGGPGVAL, IGGPTCMP |
| **Internal** | | |
| IGGPCKLC | — | — |
| IGGPLVAL | — | IGGPCKLC |
| IGGPTCMP | — | — |
| IGG0CLBB | | |
| **External** | | |
| IGGPUPDE | IGGPEXT, IGGPGET, IGGPINIT, IGGPSMFA, IGGPSMFL, IGGPSVOL, IGGPTNXO, IGGPTXO | IGGPCEXT, IGGPCSAL, IGGPMEXT, IGGPMVOL, IGGPSSWD, IGGPUALL |
| **Internal** | | |
| IGGPCEXT | — | — |
| IGGPCSAL | IGGPSALL, IGGPSPAC | — |
| IGGPMEXT | IGGPMOD | — |
| IGGPMVOL | IGGPMOD | — |
| IGGPSSWD | IGGPGET, IGGPINIT, IGGPSVOL | IGGPCEXT, IGGPMEXT, IGGPMVOL |
| IGGPUALL | IGGPGET, IGGPSPAC | — |
| IGG0CLBC | | |
| **External** | | |
| IGGPINIT | IGGPEXT | — |
| IGGPSVOL | IGGPEXT | — |
| IGG0CLBD | | |
| **External** | | |
| IGGPALT | IGGPALVL, IGGPAUPG, IGGPGET, IGGPPUPC, IGGPSMF, IGGPSMFA, IGGPSMFG | IGGPALMD, IGGPALNM |

## Procedure Calls Directory: Catalog Management Modules

| Calling Module and Its Procedures | External Procedures Called (Procedures Outside the Module) | Internal Procedures Called (Procedures within the Module) |
|---|---|---|
| **Internal** | | |
| IGGPALBT | — | — |
| IGGPALF1 | — | IGGPALGV, IGGPALSV |
| IGGPALGV | IGGPEXT | — |
| IGGPALMD | IGGPMOD | IGGPALBT |
| IGGPALNM | IGGPGET, IGGPPAD, IGGPPDE, IGGPSMFR | IGGPALF1 |
| IGGPALSV | — | — |
| IGG0CLBE | | |
| **External** | | |
| IGGPALEC | IGGPGET | IGGPALIX |
| **Internal** | | |
| IGGPALBC | — | — |
| IGGPALAE | — | — |
| IGGPALIX | IGGPEXT, IGGPGET | IGGPALBC |
| IGGPALSA | IGGPMOD, IGGPSALL | IGGPALBC, IGGPALEC |
| IGGPALVA | IGGPEXT | IGGPALBC, IGGPALAE, IGGPALSA |
| IGGPALVL | IGGPALVR, IGGPEXT, IGGPVRD | IGGPALBC, IGGPALVA |
| IGGPVRD | — | IGGPVRCV |
| IGG0CLBF | | |
| **External** | | |
| IGGPSSCR | IGGPBMR | — |
| IGG0CLBG | | |
| **External** | | |
| IGGPDEL | IGGPDCLS, IGGPDEAX, IGGPDEMV, IGGPDEPT, IGGPDIAX, IGGPDIPT, IGGPDUND, IGGPDUPG, IGGPDUSC, IGGPDVMV, IGGPGET, IGGPMCRA, IGGPSMFS, IGGPVMSC | IGGPDEXA, IGGPDLXT, IGGPDOPN, IGGPERAS |
| IGGPDEXA | IGGPEXT | IGGPDLXT |
| IGGPDLXT | — | — |
| IGGPDOPN | — | — |
| **Internal** | | |
| IGGPERAS | IGGPRPLF, IGGPRPLM | IGGPDLXT |
| IGG0CLBH | | |
| **External** | | |
| IGGPDAVO | IGGPMOD | — |
| IGGPDEFA | IGGPDUND, IGGPSMFA | IGGPDAIN |
| **Internal** | | |
| IGGPDAIN | IGGPAXCI | — |
| IGGPDANL | — | — |

## Procedure Calls Directory: Catalog Management Modules

| Calling Module and Its Procedures | External Procedures Called (Procedures Outside the Module) | Internal Procedures Called (Procedures within the Module) |
|---|---|---|
| IGG0CLBI | | |
| **External** | | |
| IGGPGET | IGGPRAG | IGGPUCCT, IGGPXIO |
| IGGPTNXO | — | — |
| IGGPTXO | IGGPEXT, IGGPMOD | — |
| IGGPUCRS | — | IGGPGET, IGGPUCVT |
| **Internal** | | |
| IGGPUCCT | — | — |
| IGGPUCVT | — | — |
| IGGPXIO | IGGPIORA | — |
| IGG0CLBJ | | |
| **External** | | |
| IGGPGDSP | IGGPEXT | IGGPGUDS |
| **Internal** | | |
| IGGPGUDS | IGGPGET | — |
| IGG0CLBK | | |
| **External** | | |
| IGGPLDCS | — | IGGPLBVC |
| IGGPLSP | IGGPGET, IGGPSMFL | IGGPLDCE, IGGPLDCS, IGGPLEMP, IGGPLSMP |
| **Internal** | | |
| IGGPLBVC | — | — |
| IGGPLDAS | — | — |
| IGGPLDCE | IGGPEXT | IGGPLDAS, IGGPLBVC |
| IGGPLEMP | — | IGGPLBVC |
| IGGPLSMP | — | — |
| IGGPLSMS | — | IGGPLBVC |
| IGG0CLBL | | |
| **External** | | |
| IGGPDELS | IGGPDBVC, IGGPDF4T, IGGPDLVM, IGGPEXT, IGGPFDSP | IGGPDLCD, IGGPDLEX, IGGPDLSC, IGGPDLSH, IGGPDLVC, IGGPDUCB |
| IGGPDLET | IGGPDFM1, IGGPF4RD, IGGPF4WR, IGGPGET, IGGPMOD, IGGPDEL, IGGPRAOP | — |
| IGGPDLVM | IGGPDVMV | — |
| IGGPDUCB | — | — |
| **Internal** | | |
| IGGPDLCD | IGGPEXT | — |
| IGGPDLEX | — | — |
| IGGPDLMF | IGGPLSP | — |
| IGGPDLSC | — | — |

## Procedure Calls Directory: Catalog Management Modules

| Calling Module and Its Procedures | External Procedures Called (Procedures Outside the Module) | Internal Procedures Called (Procedures within the Module) |
|---|---|---|
| IGGPDLSD | IGGPMOD | — |
| IGGDLSH | IGGPMOD | — |
| IGGPDLVC | IGGPEXT | IGGPDLET, IGGPDLMF |
| IGGPDLVM | — | — |
| IGG0CLBM | | |
| **External** | | |
| IGGPCKAU | IGGPINMD, IGGPSPSC | IGGPBSGT, IGGPCKCC, IGGPCKEX, IGGPLVST, IGGPPWGT, IGGPPWVR |
| **Internal** | | |
| IGGPBSGT | IGGPGET | — |
| IGGPCKCC | IGGPGET | IGGPBSGT, IGGPCKEX |
| IGGPCKEX | IGGPEXT | — |
| IGGPCLGT | IGGPEXT, IGGPGET | IGGPBSCGT, IGGPCKEX |
| IGGPLVST | — | — |
| IGGPPWGT | IGGPGET, IGGPRPLF, IGGPRPLM IGGPWTSO | IGGPCKEX |
| IGGPPWVR | — | IGGPBSGT, IGGPCKCC, IGGPCKEX, IGGPCLGT |
| IGG0CLBN | | |
| **External** | | |
| IGGPALVR | IGGPALEC, IGGPEXT, IGGPMOD | IGGPALPL, IGGPALVE, IGGPALVO |
| IGGPVRD | IGGPGET | IGGPVRCV |
| **Internal** | | |
| IGGPACHR | — | — |
| IGGPALPL | — | — |
| IGGPALVE | IGGPGET, IGGPMOD, IGGPPUPC | IGGPALPL |
| IGGPALVO | — | — |
| IGGPVRCV | IGGPF4DQ, IGGPF4RD, IGGPF4WR | IGGPACHR |
| IGG0CLBO | | |
| **External** | | |
| IGGPRAG | — | IGGPRAOR, IGGPRAX, IGGPXRIO |
| IGGPRAPA | IGGPPAD, IGGPRAPV | IGGPRAOR, IGGPRAPC, IGGPRARA, IGGPRAX |
| IGGPRAPD | IGGPRAPV | IGGPRAOR, IGGPRARC, IGGPRAX, IGGPXRIO |
| IGGPRAPU | IGGPRAPV | IGGPRAOR, IGGPRAX, IGGPXRIO |
| IGGPRARC | — | IGGPXRIO |
| IGGPRASC | — | IGGPRAOR, IGGPRARA, IGGPRARC, IGGPRAX |
| IGGPXRIO | — | IGGPRAEA |

## Procedure Calls Directory: Catalog Management Modules

| Calling Module and Its Procedures | External Procedures Called (Procedures Outside the Module) | Internal Procedures Called (Procedures within the Module) |
|---|---|---|
| **Internal** | | |
| IGGPRACC | — | IGGPXRIO |
| IGGPRAEA | IGGPEMIO, IGGPEMSG | — |
| IGGPRAOR | IGGPRAOP, IGGPCRAD | — |
| IGGPRAPC | — | — |
| IGGPRARA | — | IGGPRACC, IGGPRARC, IGGPXRIO |
| IGGPRAX | — | IGGPRACC |
| IGG0CLBP | | |
| **External** | | |
| IGGPSPAC | IGGPF4DQ, IGGPF4RD, IGGPF4WR, IGGPISCI, IGGPPUPC | IGGPDALL, IGGPDEXT, IGGPINEX, IGGPOBTN, IGGPSRH1, IGGPSRH2 IGGPWRIT |
| IGGPRET1 | — | — |
| **Internal** | | |
| IGGPDALL | — | IGGPGNAM |
| IGGPDEXT | — | IGGPGNAM |
| IGGPGNAM | — | — |
| IGGPINEX | — | — |
| IGGPOBTN | — | — |
| IGGPSRH1 | IGGPEXT | — |
| IGGPSRH2 | IGGPEXT | — |
| IGGPWRIT | IGGPMOD | — |
| IGG0CLBQ | | |
| **External** | | |
| IGGPLSTC | IGGPEXT, IGGPGET | — |
| **Internal** | | |
| — | | |
| IGG0CLBR | | |
| **External** | | |
| IGGPBMR | — | IGGPGETM, IGGPPART, IGGPPUTM |
| **Internal** | | |
| IGGPGETM | IGGPGET | IGGPPUTM |
| IGGPPART | — | — |
| IGGPPUTM | IGGPPUPC | — |
| IGG0CLBS | | |
| **External** | | |
| IGGPXEXT | — | — |
| IGGPXVAL | IGGPBMR, IGGPGREC, IGGPGVAL, IGGPSGOP | IGGPXVOL |

## Procedure Calls Directory: Catalog Management Modules

| Calling Module and Its Procedures | External Procedures Called (Procedures Outside the Module) | Internal Procedures Called (Procedures within the Module) |
|---|---|---|
| **Internal** | | |
| IGGPXEXS | IGGPGREC, IGGPGVAL | — |
| IGGPXVOL | — | IGGPXEXS |
| IGG0CLBT | | |
| **External** | | |
| IGGPXDGO | IGGPADGO, IGGPBMR, IGGPGVAL | IGGPCSMP |
| IGGPXEL2 | IGGPBMR, IGGPDEL2, IGGPGVAL | — |
| IGGPXLT2 | IGGPALT2 | — |
| IGGPXMOD | — | — |
| **Internal** | | |
| IGGPCSMP | IGGPAOCI, IGGPPUPC | — |
| IGG0CLBU | | |
| **External** | | |
| IGGPF4DQ | — | — |
| IGGPF4RD | — | IGGPF4DQ |
| IGGPF4WR | — | IGGPF4DQ |
| IGG0CLBV | | |
| **External** | | |
| IGGPSMFA | IGGPGET | IGGPSMFC, IGGPSMFD, IGGPSMFE, IGGPSMFF |
| IGGPSMFF | — | — |
| IGGPSMFL | IGGPGET, IGGPLDCS | IGGPSMFC |
| IGGPSMFR | — | IGGPSMFC |
| IGGPSMFS | — | IGGPSMFE, IGGPSMFL |
| **Internal** | | |
| IGGPSMFC | — | — |
| IGGPSMFD | — | — |
| IGGPSMFE | — | IGGPSMFM |
| IGGPSMFM | IGGPGET | IGGPSMFP |
| IGGPSMFP | — | — |
| IGG0CLBW | | |
| **External** | | |
| IGGPDEIN | IGGPGVAL | IGGPRISE, IGGPSINK, IGGPSNK2 |
| **Internal** | | |
| IGGPDOWN | IGGPAXCI, IGGPEXPD, IGGPGNEX, IGGPGVAL, IGGPPREC, IGGPSHNK | — |
| IGGPRISE | IGGPAXCI, IGGPGVAL, IGGPPREC, IGGPSHNK | IGGPSINK, IGGPUPUP |
| IGGPSINK | IGGPEXPD, IGGPSHNK | — |
| IGGPSNK2 | IGGPAXCI, IGGPGREC, IGGPPDE, IGGPPREC | IGGPDOWN |

## Procedure Calls Directory: Catalog Management Modules

| Calling Module and Its Procedures | External Procedures Called (Procedures Outside the Module) | Internal Procedures Called (Procedures within the Module) |
|---|---|---|
| IGGPUPUP | — | — |
| IGG0CLBX | | |
| **External** | | |
| IGGPDCIM | — | — |
| IGGPDSPC | IGGPDCCO | IGGPDCCC, IGGPDCCO, IGGPDDCE |
| **Internal** | | |
| IGGPDCCC | — | IGGPDCID, IGGPDCIM, IGGPDCPT |
| IGGPDCCI | — | IGGPDPBI |
| IGGPDCID | — | IGGPDCIM, IGGPMAXA |
| IGGPDCIM | — | — |
| IGGPDCPT | — | — |
| IGGPDDCE | IGGPGET | — |
| IGGPDPBI | — | — |
| IGG0CLBY | | |
| **External** | | |
| IGGPDRSP | — | IGGPDATA, IGGPDCIS, IGGPDDRT, IGGPDDSA, IGGPDDTC, IGGPDISA |
| **Internal** | | |
| IGGPDATA | — | — |
| IGGPDCIS | — | — |
| IGGPDDRT | — | — |
| IGGPDDSA | — | IGGPDATA, IGGPDDRT, IGGPDDTC |
| IGGPDDTC | — | — |
| IGGPDISA | — | — |
| IGG0CLBZ | | |
| **External** | | |
| IGGPCONV | IGGPF4RD, IGGPF4WR, IGGPMOD, IGGPPUPC, IGGPRCCR, IGGPRPLF, IGGPSCAT | IGGPGALO, IGGPVALI |
| IGGPGALO | — | — |
| IGG0CLB0 | | |
| **External** | | |
| IGGPCMKY | — | — |
| IGGPDCCO | IGGPDSPC | IGGPDCCI |
| **Internal** | | |
| IGGPDCCI | — | — |
| IGGPOPBI | — | — |
| IGG0CLB1 | | |
| **External** | | |

## Procedure Calls Directory: Catalog Management Modules

| Calling Module and Its Procedures | External Procedures Called (Procedures Outside the Module) | Internal Procedures Called (Procedures within the Module) |
|---|---|---|
| IGGPUADD | IGGPAXCI, IGGPEXT, IGGPGET, IGGPMOD, IGGPPAD, IGGPPDE | IGGPUCOM, IGGPUEND |
| IGGPUDEL | IGGPEXT, IGGPGET, IGGPMOD IGGPPDE | IGGPUCOM, IGGPUEND |
| **Internal** | | |
| IGGPUCOM | IGGPEXT | — |
| IGGPUEND | IGGPGET | — |
| IGG0CLB2 | | |
| **External** | | |
| IGGPAUPG | IGGPEXT, IGGPUADD, IGGPUDEL | IGGPAEXA |
| **Internal** | | |
| IGGPAEXA | — | — |
| IGG0CLB3 | | |
| **External** | | |
| IGGPSMF | — | — |
| IGGPSMFG | — | — |
| IGG0CLB4 | | |
| **External** | | |
| IGGPDCRA | IGGPMDDI, IGGPWCAT, IGGPWCRA | IGGPCACB, IGGPCCIO, IGGPCHIU, IGGPCI15, IGGPCRPL, IGGPCXWA, IGGPDCXI, IGGPFMT4, IGGPOCRA, IGGPRDEF, IGGPSBAL, IGGPSTRG |
| **Internal** | | |
| IGGPCACB | — | — |
| IGGPCCIO | — | — |
| IGGPCHIU | — | — |
| IGGPCI15 | — | — |
| IGGPCRPL | — | — |
| IGGPCXWA | — | — |
| IGGPDCXT | — | — |
| IGGPFMT4 | IGGPF4RD, IGGPF4WR | — |
| IGGPOCRA | — | — |
| IGGPRDEF | IGGPDEFS | — |
| IGGPSBAL | IGGPSALL | — |
| IGGPSTRG | — | IGGPRDEF, IGGPSBAL |
| IGG0CLB5 | | |
| **External** | | |
| IGGPDCLS | IGGPDLXT, IGGPDUND, IGGPGET, IGGPSMFS, IGGAVMSC | — |
| IGGPDEAX | IGGPDLXT, IGGPGET | IGGPDBMD, IGGPDEXB, IGGPDIPT, IGGPDUPG |
| IGGPDEPT | IGGPDLXT, IGGPGET, IGGPPDE, IGGPSMFS | IGGPDBMD |

## Procedure Calls Directory: Catalog Management Modules

| Calling Module and Its Procedures | External Procedures Called (Procedures Outside the Module) | Internal Procedures Called (Procedures within the Module) |
|---|---|---|
| IGGPDIAX | IGGPDEXA, IGGPDLXT, IGGPDOPN, IGGPGET | IGGPDBMD, IGGPDEXB, IGGPDUPG |
| IGGPDIPT | IGGPDLXT, IGGPGET, IGGPPDE, IGGPSMFS | IGGPDBMD, IGGADEXB |
| **Internal** | | |
| IGGPDBMD | IGGPDLXT, IGGPMOD | — |
| IGGPDEXB | IGGPDLXT, IGGPEXT | — |
| IGGPDUPG | IGGPDLXT, IGGPGET, IGGPUDEL | IGGPDEMV, IGGPDVMV |
| IGG0CLB6 | | |
| **External** | | |
| IGGPINMD | IGGPGET | — |
| IGGPSPSC | IGGPGET | — |
| IGG0CLB7 | | |
| IGGPWTSO | — | IGGPGTSO |
| **Internal** | | |
| IGGPGTSO | IGGPRPLF, IGGPRPLM | |
| **External** | | |
| IGGPRUS | IGGPEXT, IGGPGET, IGGPLDC, IGGPMOD, IGGPSSCR, IGGPTNXO, IGGPTXO | IGGPFRWK |
| **Internal** | | |
| IGGPFRWK | IGGPGET, IGGPLOC | — |
| IGG0CLB8 | | |
| **External** | | |
| IGGPDFBO | — | — |
| IGGPCNBO | — | — |
| IGGPDUND | IGGPGET, IGGPPDE | IGGPDEUN |
| IGG0CLB9 | | |
| **External** | | |
| IGGPPAIX | IGGPDRDA, IGGPDUND, IGGPPSEM | IGGPDCBG, IGGPIGDC, IGGPMODC, IGGPPRPW, IGGPRGBC |
| IGGPPATH | — | IGGPBAMC, IGGPBAWP, IGGPCGGC, IGGPIGDC, IGGPPRBW, IGGPSCRG, IGGPWSMF |
| **Internal** | | |
| IGGPBAMC | IGGPMOD | — |
| IGGPBAWP | IGGPAOCI, IGGPMOD, IGGPPDE | IGGPINAE, IGGPINBE |
| IGGPCGGC | IGGPEXT | IGGPEXGC |
| IGGPDCBG | IGGPDCCE | — |
| IGGPEXGC | IGGPEXT, IGGPGET | — |
| IGGPIGDC | — | — |

## Procedure Calls Directory: Catalog Management Modules

| Calling Module and Its Procedures | External Procedures Called (Procedures Outside the Module) | Internal Procedures Called (Procedures within the Module) |
|---|---|---|
| IGGPINAE | IGGPEXT | — |
| IGGPINBE | — | — |
| IGGPMODC | IGGPAOCI, IGGPGET, IGGPMOD | — |
| IGGPPRPW | IGGPGET, IGGPCKAU | — |
| IGGPRGBC | IGGPMOD | — |
| IGGPSCRG | — | — |
| IGGPWSMF | IGGPSMFA | — |
| IGG0CLC9 | | |
| **External** | | |
| IGG0CLC9 | IGGPACDV, IGGPRET1 | BLDCCA, IGGPRCU |
| **Internal** | | |
| BLDCCA | — | — |
| IGGPRCU | IGGPCCCR, IGGPEMSG, IGGPRARC, IGGPRPLF | — |
| IGG0CLDA | | |
| **External** | | |
| IGGPMODI | IGGPMOD | — |
| IGGPRAPV | IGGPF4DQ, IGGPF4RD, IGGPF4WR, IGGPXIO, IGGPXRIO | — |
| IGGPWCAT | IGGPPAD | — |
| IGGPWCRA | IGGPXRIO | — |

# Procedure Called-By Directory

## *Open/Close/EOV Procedure Called-By (Backward-Reference) Table*

| Called Module | Calling Modules |
|---|---|
| IDA0192A | — |
| IDA0192B | IDA0192F |
| IDA0192C | IDA0192A, IDA0192B, IDA0192G, IDA0192V, IDA0200B, IDA0200T, IDA0231B, IDA0557A |
| IDA0192D | IDA0192B, IDA0192V, IDA0200B, IDA0231B, IDA0557A |
| IDA0192F | IDA0192A |
| IDA0192G | IFG0195T, SECLOADA |
| IDA0192I | — |
| IDA0192M | IDA0192A, IDA0192B, IDA0192F, IDA0192W, IDA0192Y, IDA0192Z, IDA0A05B |
| IDA0192P | IDA0192A, IDA0192B, IDA0192D, IDA0192F, IDA0192G, IDA0200B, IDA0200T, IDA0231B, IDA0231T, IDA0557A, IFG0193A, IFG0200V, IGC0002C |
| IDA0192S | IDA0192A, IDA0200B, IDA0231B, IDA0557A |
| IDA0192V | IDA0192F, IDA0557A |
| IDA0192W | IDA0192Y |
| IDA0192Y | IDA0192B, IDA0192Z, IDA0200T |
| IDA0192Z | IDA0192B |
| IDA0200B | IDA0200T |
| IDA0200S | — |
| IDA0200T | — |
| IDA0231B | IDA0231T |
| IDA0231T | — |
| IDA0557A | — |

## *Record Management Procedure Called-By (Backward-Reference) Table*

| Called Module and Its Procedures | Calling modules |
|---|---|
| IDA019RA | IDA019RV, IDA019R4 |
| IDA019RB | IDA019RA, IDA019RI, IDA019RO |
| IDA019RC | IDA019RB, IDA019RH, IDA019RS, IDA019RT, IDA019RV, IDA019RW |
| IDA019RE | IDA019RM, IDA019RT |
| **Procedure** | |
| IDAREPOS | IDA019RS, IDA019RT |
| IDA019RF | IDA019RE, IDA019RS, IDA019RT |

## Record Management Procedure Called-By (Backward Reference) Table

| Called Module and Its Procedures | Calling Modules |
|---|---|
| IDA019RG | IDA019RP, IDA019SA |
| **Procedure** | |
| IDAIST | IDA019RJ |
| IDA019RH | IDA019RE, IDA019RI |
| **Procedures** | |
| IDAIVIXB | IDA019RJ, IDA019RS, IDA019RT |
| IDASPACE | IDA019RJ |
| IDA019RI | IDA019RF |
| **Procedures** | |
| IDAHLINS | IDA019SF |
| IDANEWRD | IDA019SF |
| IDA019RJ | IDA019RI |
| **Procedures** | |
| IDAR | IDA019RG |
| IDAWR | IDA019RG |
| IDA019RK | IDA019RF, IDA019RN, IDA019RP, IDA019RQ, IDA019R1, IDA019R4, IDA019R8, IDA019SA |
| IDA019RL | IDA019R4 |
| IDA019RM | IDA019RE, IDA019RF, IDA019RL, IDA019R4, IDA019SF |
| **Procedure** | |
| IDACHKKR | IDA019RS |
| IDA019RN | — |
| **Procedures** | |
| IDAAQR | IDA019RG, IDA019RI, IDA019RJ |
| IDAER | IDA019RG, IDA019RJ |
| IDA019RO | IDA019R8 |
| IDA019RP | — |
| **Procedures** | |
| IDAENDRQ | IDA019RX, IDA019R1 |
| IDATJXIT | IDA019RE, IDA019RF, IDA019RL, IDA019RM, IDA019RQ, IDA019RR, IDA019RT |
| IDA019RQ | IDA019RR |
| IDA019RR | IDA019R1 |
| **Procedure** | |
| IDARRDRL | IDA019RQ |
| IDA019RS | IDA019RL |
| **Procedures** | |
| IDAADSEG | IDA019RT |
| IDAMVSEG | IDA019RT |

## Record Management Procedure Called-By (Backward Reference) Table

| Called Module and Its Procedures | Calling Modules |
|---|---|
| IDA019RT | IDA019RM |
| **Procedures** | |
| IDADARTV | IDA019R4 |
| IDAJRNSR | IDA019RS |
| IDASPNPT | IDA019RG, IDA019RJ |
| IDA019RU | IDA019RM, IDA019R4, IDA019SA |
| **Procedure** | |
| IDAXGPLH | IDA019RX |
| IDA019RV | IDA019RW, IDA019SF |
| **Procedures** | |
| IDAADVPH | IDA019R4 |
| IDARVRS1 | IDA019R2 |
| IDA019RW | — |
| **Procedures** | |
| IDAABF | IDA019RA, IDA019RF, IDA019RQ, IDA019RR, IDA019RT, IDA019RV, IDA019R4, IDA019R8, IDA019SF |
| IDAAIBF | IDA019RF, IDA019RT |
| IDAFRBA | IDA019RA, IDA019RY, IDA019R2, IDA019R8 |
| IDAGWSGW | IDA019RZ |
| IDA019RX | IDA019R1 |
| **Procedures** | |
| IDAGETWS | IDA019RU |
| IDARELWS | IDA019RU |
| IDARXBD | IDA019R4 |
| IDA019RY | IDA019RZ |
| IDA019RZ | — |
| **Procedures** | |
| IDAEXCL | IDA019R8 |
| IDAFREEB | IDA019RA, IDA019RB, IDA019RE, IDA019RF, IDA019RG, IDA019RI, IDA019RJ, IDA019RK, IDA019RO, IDA019RQ, IDA019RR, IDA019RT, IDA019RV, IDA019R1, IDA019R4, IDA019R8, IDA019SA, IDA019SF |
| IDAGNFL | IDA019RE, IDA019RG, IDA019RI, IDA019RK |
| IDAGNNFL | IDA019RF, IDA019RQ, IDA019R4, IDA019R8, IDA019SA, IDA019SF |
| IDAGNXT | IDA019RA, IDA019RF, IDA019RQ, IDA019RR, IDA019RT, IDA019RV, IDA019R4, IDA019R8 |
| IDAGRB | IDA019RA, IDA019RB, IDA019RE, IDA019RF, IDA019RI, IDA019RJ, IDA019RO, IDA019RR, IDA019RS, IDA019RT, IDA019RV, IDA019RW, IDA019R4, IDA019R8, IDA019SA, IDA019SF |
| IDAGWSEG | IDA019RS |
| IDAMRKBF | IDA019R1 |

## Record Management Procedure Called-By (Backward Reference) Table

| Called Module and Its Procedures | Calling Modules |
|---|---|
| IDASBF | IDA019RA, IDA019RE, IDA019RF, IDA019RP, IDA019RS, IDA019RT, IDA019RU, IDA019R1, IDA019R8, IDA019SF |
| IDASCHBF | IDA019R1 |
| IDAWAIT | IDA019RA, IDA019RW, IDA019RY, IDA019R2 |
| IDAWRBFR | IDA019RE, IDA019RF, IDA019RH, IDA019RJ, IDA019RK, IDA019RP, IDA019RQ, IDA019RR, IDA019RS, IDA019RT, IDA019RW, IDA019R1, IDA019R4, IDA019R8, IDA019SA, IDA019SF |
| IDAWRTBF | IDA019R1 |
| IDA019R2 | IDA019RZ |
| IDA019R3 | IDA019RW, IDA019RY, IDA019R2 |
| IDA019R4 | IDA019RU, IDA019RX, IDA019R1 |
| IDA019R5 | IDA019RP, IDA019RY, IDA019R1 |
| **Procedures** | |
| IDADRQ | IDA019RQ, IDA019RS, IDA019RT, IDA019RU, IDA019RX, IDA019RY, IDA019R3, IDA019R4, IDA019R8, IDA019SB |
| IDAEOVIF | IDA019RF, IDA019RG, IDA019RI, IDA019RK, IDA019RN, IDA019RQ, IDA019R3, IDA019R8, IDA019SA, IDA019SF |
| IDAERROR | IDA019R1 |
| IDAEXEX | IDA019RP, IDA019RY |
| IDAEXITR | IDA019RP |
| IDARSTRT | IDA019R1 |
| IDA019R8 | IDA019R1 |
| IDA019SA | IDA019RM, IDA019RP, IDA019RT |
| IDA019SB | IDA019R3 |
| IDA019SF | IDA019RF |

## *Catalog Management Procedure Called-By (Backward Reference) Table*

This table lists procedures (not within the called procedure's module) that call a Catalog Management procedure. In addition to the calling procedures listed, a called procedure might be called by procedures within its module. See "Procedure Calls Directory: Catalog Management Procedures" to determine the calling procedures within the module.

| Called Module and its External Procedures | Calling Procedure (in Module) |
|---|---|
| IGGOCLAA | |
| **Procedure:** | |
| IGGPSLEN | IGGPSLOC(AM) |
| IGG0CLAB | |
| **Procedure:** | |
| IGGPACDV | IGG0CLC9(C9) |
| IGG0CLAC | |
| **Procedures:** | |
| IGGPDCRC | IGGPDCME(AE) |
| IGGPMCO | IGGPACDV(AB) |
| IGG0CLAD | |
| **Procedures:** | |
| IGGPMCO2 | IGGPDCOC(AE), IGGPMCO(AC) |
| IGGPRAOP | IGGPDLET(BL), IGGPRAOR(BO) |
| IGGPCRAD | IGGPRAOR(BO) |
| IGG0CLAE | |
| **Procedures:** | |
| IGGPDCME | IGGPDEFC(AS) |
| IGGPMEMB | IGGPDCSP(AS) |
| IGG0CLAF | |
| **Procedures:** | |
| IGGPDELC | IGGPCOVR(AT) |
| IGGPEMIO | IGGPIORA(AG), IGGPRAEA(BO) |
| IGGPEMSG | IGGPIORA(AG), IGGPRAEA(BO), IGGPRCU(C9) |
| IGG0CLAG | |
| **Procedures:** | |
| IGGPAOCI | IGGPBAWP(B9), IGGPCSMP(BT), IGGPDCCE(AN), IGGPDCPC(AP), IGGPMODC(B9) |
| IGGPAXCI | IGGPADGO(AW), IGGPAGOP(AW), IGGPDAIN(BH), IGGPDEFS(AQ), IGGPDOWN(BW), IGGPMBGO(AX), IGGPMVAR(AX), IGGPRISE(BW), IGGPSNK2(BW), IGGPUADD(B1) |
| IGGPCCCR | IGGPDEFC(AS), IGGPDELC(AF), IGGPRCU(C9) |
| IGGPIORA | IGGPXIO(BI) |
| IGGPISCI | IGGPDEFS(AQ), IGGPSALL(AR), IGGPSPAC(BP) |

## Catalog Management Procedure Called-By (Backward Reference) Table

| Called Module and Its External Procedures | Calling Procedure (in Module) |
|---|---|
| IGGPPAD | IGGPALNM(BD), IGGPDCCE(AN), IGGPDCPC(AN), IGGPDEFS(AQ), IGGPPREC(AW), IGGPRAPA(BO), IGGPUADD(B1), IGGPWCAT(DA) |
| IGGPPDE | IGGPALNM(BD), IGGPBAWP(B9), IGGPDELC(AF), IGGPDEL2(AV), IGGPDEPT(B5), IGGPDIPT(B5), IGGPDUND(B8), IGGPSNK2(BW), IGGPUADD(B1), IGGPUDEL(B1) |
| IGGPPLHC | IGGPRPLM(AH) |
| IGGPPUPC | IGGPALT(BD), IGGPALVE(BN), IGGPCONV(BZ), IGGPCSMP(BT), IGGPDF4T(A7), IGGPFDSP(A1), IGGPPREC(AW), IGGPPUTM(BR), IGGPSPAC(BP) |
| IGGPRCCR | IGGPCONV(BZ) |
| IGGPXIO | IGGPDCPR(AE), IGGPDELC(AF), IGGPRAPV(DA) |
| IGG0CLAH | |
| **Procedures:** | |
| IGGPRPLF | IGGPCONV(BZ), IGGPERAS(BG), IGGPPWGT(BM), IGGPRCU(C9), IGGPRCU(C9), IGGPGTSO(B6) |
| IGGPRPLM | IGGPERAS(BG), IGGPPWGT(BM), IGGPGTSO(B6) |
| IGGPSCAT | IGGPACDV(AB), IGGPCDVR(AT), IGGPCONV(BZ), IGGPDEFS(AQ), IGGPDELC(AF) |
| IGG0CLAI | |
| **Procedures:** | |
| IGGPDFM1 | IGGPDLET(BL) |
| IGGPFDSP | IGGPDELC(AF), IGGPDELS(BL) |
| IGG0CLAJ | |
| **Procedure:** | |
| IGGPDBDI | IGGPDSCB(AN) |
| IGG0CLAK | |
| **Procedure:** | |
| IGGPDCMB | IGGPDBDI(AJ) |
| IGG0CLAL | |
| **Procedures:** | |
| IGGPDEF | IGGPCDVR(AT) |
| IGGPDTIM | IGGPDBDI(AJ), IGGPDEFS(AQ) |
| IGG0CLAM | |
| **Procedures:** | |
| IGGPDBVC | IGGPDELC(AF), IGGPDELS(BL) |
| IGGPSLEL | IGGPSLCG(AA), IGGPSLEN(AA), IGGPSLR(AA), IGGPSLY(AA) |
| IGGPSLOC | IGGPACDV(AB) |
| IGG0CLAN | |
| **Procedures:** | |
| IGGPDCCE | IGGPDCBG(B9), IGGPDCPC(AP) |
| IGGPDRDA | IGGPAIX(B9) |

## Catalog Management Procedure Called-By (Backward Reference) Table

| Called Module and Its External Procedures | Calling Procedure (in Module) |
|---|---|
| IGGPDSCB | IGGPDEF(AL) |
| IGGPPSEM | IGGPPAIX(B9) |
| IGG0CLAP | |
| **Procedure:** | |
| IGGPDCDA | IGGPDSCB(AN) |
| IGG0CLAQ | |
| **Procedure:** | |
| IGGPDEFS | IGGPCDVR(AT), IGGPDSCP(AS), IGGPDSPO(AJ), IGGPRDEL(B4) |
| IGG0CLAR | |
| **Procedure:** | |
| IGGPSALL | IGGPALSA(BE), IGGPCDSD(A6), IGGPCSAL(BB), IGGPDCNV(AJ), IGGPDCSP(AS), IGGPDSPO(AJ), IGGPSBAL(B4) |
| IGG0CLAS | |
| **Procedure:** | |
| IGGPDEFC | IGGPDCME(AE), IGGPDSCB(AN) |
| IGG0CLAT | |
| **Procedure:** | |
| IGGPCOVR | IGGPACOV(AB) |
| IGG0CLAU | |
| **Procedure:** | |
| IGGPSALS | IGGPSALL(AR) |
| IGG0CLAV | |
| **Procedures:** | |
| IGGPDEL2 | IGGPXEL2(BT) |
| IGGPMOD | IGGPALMD(BD), IGGPALSA(BE), IGGPALVE(BN), IGGPALVR(BN), IGGPBAMC(B9), IGGPBAWP(B9), IGGPCONV(BZ), IGGPDAVO(BH), IGGPDBMD(B5), IGGPDCME(AE), IGGPDEDD(A7), IGGPDESH(A7), IGGPDEVG(A7), IGGPDLET(BL), IGGPDLSD(BL), IGGPDLSH(BL), IGGPDMOP(AK), IGGPDSMD(A6), IGGPFDSP(AI), IGGPMEXT(BB), IGGPMODC(B9), IGGPMODI(DA), IGGPMVOL(BB), IGGPRGBC(B9), IGGPRUS(B7), IGGPSALL(AR), IGGPTXO(BI), IGGPUADD(B1), IGGPUDEL(B1), IGGPWRIT(BP) |
| IGGPSGOP | IGGPMGO(AX), IGGPXVAL(B5) |
| IGGPUPD | IGGPACOV(AB) |
| IGG0CLAW | |
| **Procedures:** | |
| IGGPADGO | IGGPXDGO(BT) |
| IGGPGNEX | IGGPDOWN(BW), IGGPMBGO(AX), IGGPMVAR(AX) |
| IGGPIGOP | IGGPMGO(AX) |
| IGGPPREC | IGGPDEL2(AV), IGGPDOWN(BW), IGGPGREC(BA), IGGPRISE(BW), IGGPSFPL(AV), IGGPSNK2(BW) |

## Catalog Management Procedure Called-By (Backward Reference) Table

| Called Module and Its External Procedures | Calling Procedure (in Module) |
|---|---|
| IGG0CLAX | |
| **Procedures:** | |
| IGGPALT2 | IGGPXLT2(BT) |
| IGGPDGO | IGGPDEL2(AV) |
| IGGPDGOP | IGGPDEL2(AV) |
| IGGPEXPD | IGGPDOWN(BW), IGGPSINK(BW) |
| IGGPMGO | IGGPAGOP(AW) |
| IGGPSHNK | IGGPDOWN(BW), IGGPRISE(BW), IGGPSINK(BW) |
| IGG0CLAY | |
| **Procedure:** | |
| IGGPSLNC | IGGPACOV(AB), IGGPEXT(AZ), IGGPMOD(AV) |
| IGG0CLAZ | |
| **Procedures:** | |
| IGGPEXT | IGGPALGV(BD), IGGPALIX(BE), IGGPALVA(BE), IGGPALVL(BE), IGGPALVR(BN), IGGPAUPG(B2), IGGPCGGC(B9), IGGPCKEX(BM), IGGPCLGT(BM), IGGPDCDS(AF), IGGPDELC(AF), IGGPDELS(BL), IGGPDEMV(A7), IGGPDEVG(A7), IGGPDEXA(BG), IGGPDEXB(B5), IGGPDLCD(BL), IGGPDLVC(BL), IGGPEXGC(B9), IGGPFDSP(AI), IGGPGDSP(BJ), IGGPINAE(B9), IGGPINIT(BC), IGGPLDCE(BK), IGGPLSTC(BQ), IGGPRUS(B7), IGGPSALL(AR), IGGPSALS(AU), IGGPSHR1(B9), IGGPSHR2(BP), IGGPSLCG(AA), IGGPSLEL(AM), IGGPSLR(AA), IGGPSVOL(BC), IGGPTXO(BI), IGGPUADD(B1), IGGPUCOM(B1), IGGPUDEL(B1), IGGPUPDE(BB) |
| IGGPLOC | IGGPACDV(AB), IGGPFRWK(B7), IGGPRUS(B7) |
| IGG0CLA6 | |
| **Procedures:** | |
| IGGPBJFB | IGGPDEFS(AQ) |
| IGGPCBPT | IGGPDEFS(AQ) |
| IGGPCRTC | IGGPDEFS(AQ) |
| IGGPDFS2 | IGGPDEFS(AQ) |
| IGG0CLA7 | |
| **Procedures:** | |
| IGGPDEMV | IGGPDEL(BG), IGGPDUPG(B5) |
| IGGPDF4T | IGGPDELS(BL), IGGPDLER(AF) |
| IGGPDUSC | IGGPDEL(BG) |
| IGGPDVMV | IGGPDEL(BG), IGGPDLVM(BL) |
| IGGPMCRA | IGGPDEL(BG) |
| IGGPVMSC | IGGPDCLS(B5), IGGPDEL(BG) |

## Catalog Management Procedure Called-By (Backward Reference) Table

| Called Module and Its External Procedures | Calling Procedure (in Module) |
|---|---|
| IGG0CLBA | |
| **Procedures:** | |
| IGGPGREC | IGGPADGO(AW), IGGPAGOP(AW), IGGPGREL(AW), IGGPGREP(AZ), IGGPMBGO(AX), IGGPMVAR(AX), IGGPSCNF(AZ), IGGPSFPL(AV), IFFPSGOP(AV), IGGPSNK2(BW), IGGPXEXS(BS), IGGPXVAL(BS) |
| IGGPGVAL | IGGPALT2(AX), IGGPDEIN(BW), IGGPDOWN(BW), IGGPLOC2(AZ), IGGPMVAR(AX), IGGPRISE(BW), IGGPSCNF(AZ), IGGPUPGD(AZ), IGGPXDGO(BT), IGGPXEL2(BT), IGGPXEXS(BS), IGGPXVAL(BS) |
| IGGPTSTS | IGGPSFPL(AV) |
| IGG0CLBB | |
| **Procedure:** | |
| IGGPUPDE | IGGPUPD(AV) |
| IGG0CLBC | |
| **Procedures:** | |
| IGGPINIT | IGGPSSWD(BB), IGGPUPDE(BB) |
| IGGPSVOL | IGGPSSWD(BB), IGGPUPDE(BB) |
| IGG0CLBD | |
| **Procedure:** | |
| IGGPALT | IGGPCDVR(AT) |
| IGG0CLBE | |
| **Procedures:** | |
| IGGPALEX | IGGPALVR(BN) |
| IGGPALVL | IGGPALT(BD) |
| IGG0CLBF | |
| **Procedure:** | |
| IGGPSSCR | IGGPRUS(B7), IGGPVMSC(A7) |
| IGG0CLBG | |
| **Procedures:** | |
| IGGPDEL | IGGPCDVR(AT) |
| IGGPDEXA | IGGPDIAX(B5) |
| IGGPDLXT | IGGPDBMB(B5), IGGPDCLS(B5), IGGPDEAX(B5), IGGPDEMV(A7), IGGPDEPT(B5), IGGPDEVG(A7), IGGPDEXB(B5), IGGPDF4T(A7), IGGPDIAX(B5), IGGPDIPT(B5), IGGPDUPG(B5), IGGPDUSC(A7), IGGPMCRA(A7) |
| IGGPDOPN | IGGPDCDS(AF), IGGPDIAX(B5), IGGPFDSP(AI) |
| IGG0CLBH | |
| **Procedure:** | |
| IGGPDEFA | IGGPCDVR(AT) |

## Catalog Management Procedure Called-By (Backward Reference) Table

| Called Module and Its External Procedures | Calling Procedure (in Module) |
|---|---|
| IGG0CLBI | |
| **Procedures:** | |
| IGGPGET | IGGPALEX(BE), IGGPALIX(BE), IGGPALNM(BD), IGGPALT(BD), IGGPALVE(BN), IGGPBSGT(BM), IGGPCKCC(BM), IGGPCLGT(BM), IGGPDBBI(AJ), IGGPDCLS(B5), IGGPDDCE(BX), IGGPDEAX(B5), IGGPDEDE(AL), IGGPDEFS(AQ), IGGPDEL(BG), IGGPDELC(AF), IGGPDEPT(B5), IGGPDIAX(B5), IGGPDIPT(B5), IGGPDLET(BL), IGGPDSPO(AJ), IGGPDUND(BB), IGGPDUPG(B5), IGGPEXGC(B9), IGGPFDSP(AI), IGGPFRWK(B7), IGGPGETM(BR), IGGPGREC(BA), IGGPGUDS(BJ), IGGPINMD(B6), IGGPISCI(AG), IGGPLSP(BK), IGGPLSTC(BQ), IGGPMCRA(A7), IGGPMODC(B9), IGGPPRPW(B9), IGGPWGT(BM), IGGPRUS(B7), IGGPSALL(AR), IGGPSCAT(AH), IGGPSLCG(AA), IGGPSLR(AA), IGGPSLY(AA), IGGPSMFA(BV), IGGPSMFL(BV), IGGPSMFM(BV), IGGPSPSC(B6), IGGPSSWD(BB), IGGPUADD(B1), IGGPUALL(BB), IGGPUDEL(B1), IGGPUEND(B1), IGGPUPDE(BB), IGGPUPGD(AZ), IGGPVMSC(A7), IGGPVRD(BM) |
| IGGPTNXO | IGGPDSPO(AJ), IGGPRUS(B7), IGGPUPDE(BB) |
| IGGPTXO | IGGPDSPO(AJ), IGGPRUS(B7), IGGPUPDE(BB) |
| IGGPUCRS | IGGPRUS(B7), IGGPSCAT(AH) |
| IGG0CLBJ | |
| **Procedure:** | |
| IGGPGDSP | IGGPACDV(AB) |
| IGG0CLBK | |
| **Procedures:** | |
| IGGPLDCS | IGGPSMFL(BV) |
| IGGPLSP | IGGPACOV(AB), IGGPDEFS(AQ), IGGPDLMF(BL) |
| IGG0CLBL | |
| **Procedures:** | |
| IGGPDELS | IGGPCDVR(AT) |
| IGGPDLET | IGGPFDSP(AI) |
| IGGPDLVM | IGGPDELS(BL), IGGPFDSP(AI) |
| IGGPDUCB | IGGPFDSP(AI) |
| IGG0CLBM | |
| **Procedure:** | |
| IGGPCKAU | IGGPACDV(AB), IGGPCDVR(AT), IGGPDEFS(AQ), IGGPPRPW(B9), IGGPSCAT(AH) |
| IGG0CLBN | |
| **Procedures:** | |
| IGGPALVR | IGGPALVL(BE) |
| IGGPVRD | IGGPALVL(BE) |

## Catalog Management Procedure Called-By (Backward Reference) Table

| Called Module and Its External Procedures | Calling Procedure (in Module) |
|---|---|
| IGG0CLBO | |
| **Procedures:** | |
| IGGPRAG | IGGPGET(BI) |
| IGGPRARA | IGGPPAD(AG) |
| IGGPRAPD | IGGPPDE(AG) |
| IGGPRAPU | IGGPPAD(AG), IGGPUPC(AG) |
| IGGPRARC | IGGPRCU(C9) |
| IGGPRASC | IGGPISCI(AG) |
| IGGPXRIO | IGGPRAPV(DA), IGGPWCRA(DA) |
| IGG0CLBP | |
| **Procedures:** | |
| IGGPRET1 | IGG0CLC9(C9) |
| IGGPSPAC | IGGPCSAL(BB), IGGPUALL(BB) |
| IGG0CLBQ | |
| **Procedure:** | |
| IGGPLSTC | IGGPCDVR(AT) |
| IGG0CLBR | |
| **Procedure:** | |
| IGGPBMR | IGGPEDS(AU), IGGPSALS(AU), IGGPSSCR(BF), IGGPXDGO(BT), IGGPXEL2(BT), IGGPXVAL(BS) |
| IGG0CLBS | |
| **Procedure:** | |
| IGGPXEXT | IGGPEXT(AZ) |
| IGG0CLBT | |
| **Procedures:** | |
| IGGPXDGO | IGGPSFPL(AV) |
| IGGPXEL2 | IGGPSFPL(AV) |
| IGGPXLT2 | IGGPSFPL(AV) |
| IGG0CLBU | |
| **Procedures:** | |
| IGGPF4DQ | IGGPDFM1(AI), IGGPDF4T(A7), IGGPF4PR(AQ), IGGPRAPV(DA), IGGPSPAC(BP), IGGPVRCV(BN) |
| IGGPF4RD | IGGPCONV(BZ), IGGPDCBO(AE), IGGPDELC(AF), IGGPDFM1(AI), IGGPDF4T(A7), IGGPDLET(BL), IGGPFMT4(B4), IGGPF4PR(AQ), IGGPRAPV(DA), IGGPSPAC(BP), IGGPTMST(AQ), IGGPVRCV(BN) |
| IGGPF4WR | IGGPCONV(BZ), IGGPDCBO(AE), IGGPDELC(AF), IGGPDF4T(A7), IGGPDLET(BL), IGGPFMT4(B4), IGGPF4PR(AQ), IGGPRAPV(DA), IGGPSPAC(BP), IGGPTMST(AQ), IGGPVRCV(BN) |

## Catalog Management Procedure Called-By (Backward Reference) Table

| Called Module and Its External Procedures | Calling Procedure (in Module) |
|---|---|
| IGG0CLBV | |
| **Procedures:** | |
| IGGPSMFA | IGGPALT(BD), IGGPDCOC(AE), IGGPDDMF(AK), IGGPDEFA(BH), IGGPUPDE(BB), IGGPWSMF(B9) |
| IGGPSMFL | IGGPLSP(BK), IGGPUPDE(BB) |
| IGGPSMFR | IGGPALNM(BD) |
| IGGPSMFS | IGGPDCLS(B5), IGGPDEL(BG), IGGPDEPT(B5), IGGPDIPT(B5) |
| IGG0CLBW | |
| **Procedure:** | |
| IGGPDEIN | IGGPMVAR(AX) |
| IGG0CLBX | |
| **Procedure:** | |
| IGGPDSPC | IGGPDCCO(BO), IGGPDRDA(AN) |
| IGG0CLBY | |
| **Procedure:** | |
| IGGPDRSP | IGGPDRDA(AN) |
| IGG0CLBZ | |
| **Procedures:** | |
| IGGPAUPG | IGGPALT(BD) |
| IGGPCONV | IGGPCOVR(AT) |
| IGG0CLB0 | |
| **Procedures:** | |
| IGGPCMKY | IGGPDRPG(AL) |
| IGGPDCCO | IGGPDSPC(BX) |
| IGG0CLB1 | |
| **Procedures:** | |
| IGGPUADD | IGGPAUPG(B2), IGGPDBDI(AJ) |
| IGGPUDEL | IGGPAUPG(B2), IGGPDUPG(B5) |
| IGG0CLB3 | |
| **Procedures:** | |
| IGGPSMF | IGGPALT(BD), IGGPDEL2(AV), IGGPPREC(AW) |
| IGGPSMFG | IGGPALT(BD), IGGPDEL2(AV), IGGPGREC(BA), IGGPSCNF(AZ) |
| IGG0CLB4 | |
| **Procedure:** | |
| IGGPDCRA | IGGPDEFS(AQ) |

## Catalog Management Procedure Called-By (Backward Reference) Table

| Called Module and Its External Procedures | Calling Procedure (in Module) |
|---|---|
| IGG0CLB5 | |
| **Procedures:** | |
| IGGPDCLS | IGGPDEL(BG) |
| IGGPDEAX | IGGPDEL(BG) |
| IGGPDEPT | IGGPDEL(BG) |
| IGGPDIAX | IGGPDEL(BG) |
| IGGPDIPT | IGGPDEL(BG) |
| IGGPDUPG | IGGPDEL(BG) |
| IGG0CLB6 | |
| **Procedures:** | |
| IGGPINMD | IGGPCKAU(BM) |
| IGGPSPSC | IGGPCKAU(BM) |
| IGGPWTSO | IGGPPWET(BM) |
| IGG0CLB7 | |
| **Procedure:** | |
| IGGPRUS | IGGPUPD(AV) |
| IGG0CLB8 | |
| **Procedures:** | |
| IGGPCNBO | IGGPDSPO(AJ) |
| IGGPDFBO | IGGPDBOI(AJ) |
| IGGPDUND | IGGPDCBO(AE), IGGPDCLS(B5), IGGPDEFA(BH), IGGPDEL(BG), IGGPPAIX(B9) |
| IGG0CLC9 | |
| **Procedure:** | |
| IGG0CLC9 | IGG0CLA1(A1) |
| IGG0CLDA | |
| **Procedures:** | |
| IGGPMODI | IGGPDCRA(B4) |
| IGGPRAPV | IGGPRAPA(BO), IGGPRAPD(BO), IGGPRAPU(BO) |
| IGGPWCAT | IGGPDCRA(B4) |
| IGGPWCRA | IGGPDCRA(B4) |

# DATA AREAS

"Data Areas" describes the VSAM data set, index, and catalog and their record formats.

"Data Areas" also describes each VSAM control block and shows the relationships between VSAM control blocks. *OS/VS1 VSAM Cross Reference* 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 topics 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, through 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 blocksize. 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 52 shows the format of a control interval.

When a VSAM 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

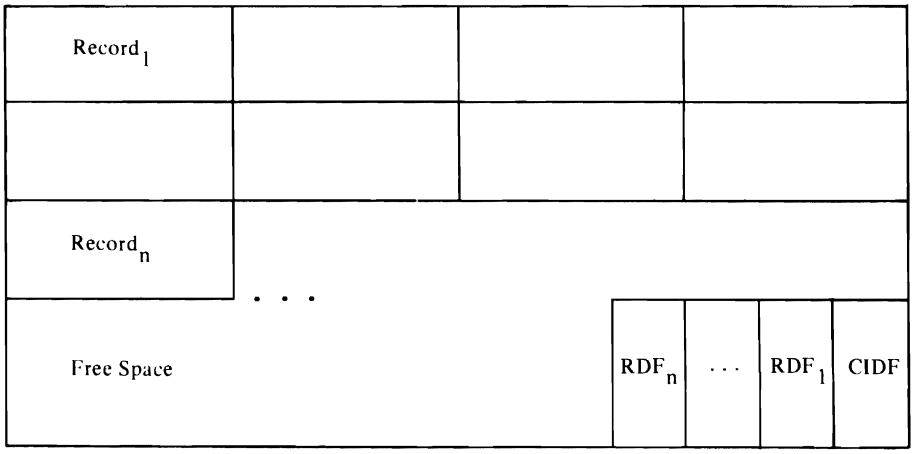



Figure 52. Control Interval Format

area for use by records to be added to the data set. As records are added to 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 records are 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 accordin g 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:<br>00 No.<br>01 Yes—this is the first segment.<br>10 Yes—this is the last segment.<br>11 Yes—this is an intermediate segment. |
| | ....x... | Indicates what the 2-byte binary number that follows this control field gives:<br>0 The length of the record, segment, or slot described by this RDF.<br>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:<br>• When bit 4 in the control field is 0, gives the length of the record, segment, or slot described by this RDF.<br>• 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.<br>• 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

The Control Interval Definition Field (CIDF) describes the control interval. The format of the CIDF is:

| Offset | Bytes and Length | Description |
|---|---|---|
| 0(0) | 2 | The displacement from the beginning of the control interval to the beginning of the unused space, or, if there is no unused space, to the beginning of the control information. This number is equal to the length of the data (records, record slots, or record segment). In a control interval without data, the number is 0. |
| 2(2) | 2 | The length of the unused space. This number is equal to the length of the control interval, minus the length of the control information, minus the 2-byte 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. |

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.

### *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 key-sequenced cluster is fully compatible with the format of 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 53 shows the basic format of an index control interval. The RDF and CIDF fields are described under "Control Interval" earlier in this chapter.

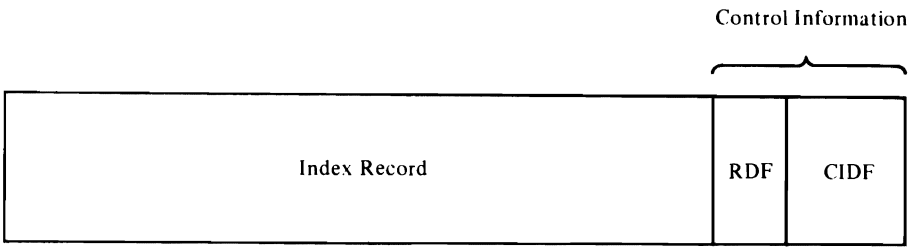



Figure 53. Index Control Interval Format

Figure 54 shows an expansion of the record portion of the index control interval.

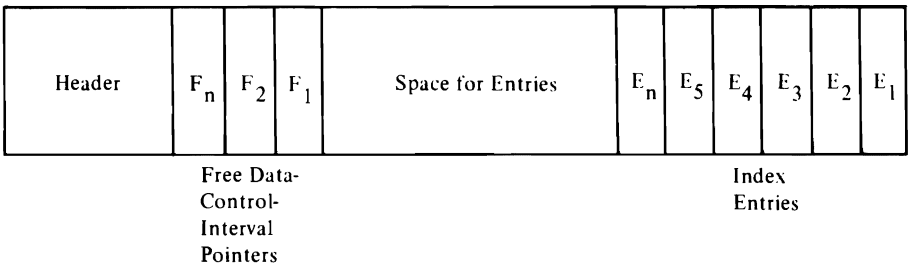



Figure 54. Index Record Format

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 55 shows the index entry portion of the index record divided into sections.

The parts of an index record—header, free data-control interval pointers, and entry sections—are described in the paragraphs that follow.

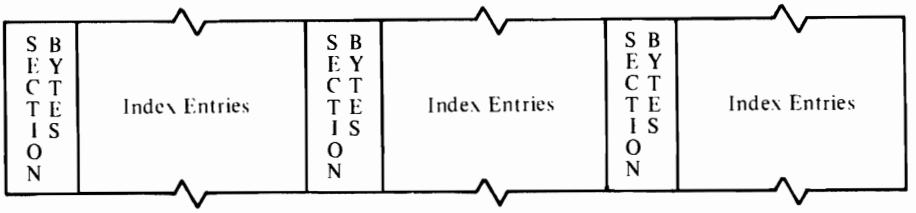



Figure 55. Index Entries Grouped into Sections

## Index Record Header

The format of the index record header is:

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 2 | IXHLL | Length, in bytes, of the index record, including this field. |
| 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 control intervals in the data component; in the index set, they point to 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:<br>X'01' 1-byte pointer<br>X'03' 2-byte pointer<br>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) section entry in the index record (the leftmost entry).[2] |
| 22 (16) | 2 | IXHSEO | Displacement from the beginning of this record to the first (low-key) section entry in the index record (the leftmost entry in the rightmost section).[2] |

[1] Pointers are allowed to vary in length to conserve index space. If, for example, the number of items to be referenced by an index record is less than 256, a one-byte pointer can be used; if the number is greater than 256 and less than 65,536, a two-byte pointer can be used; and if the number is greater than 65,536, a three-byte pointer can be 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 interval in the logically next lower index level.

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 56 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.

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

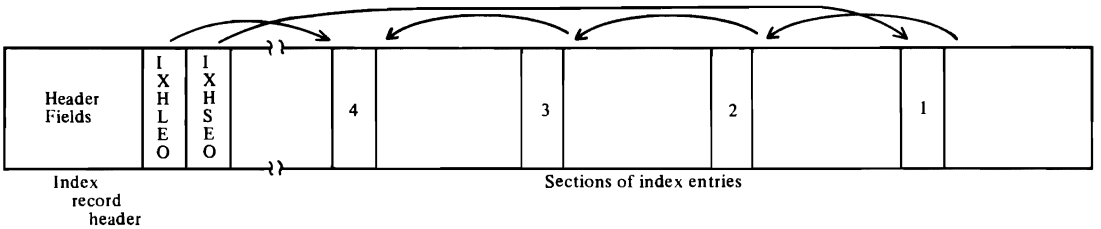



Figure 56. Index-Entry Section Pointers

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:

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| **Header** | | | |
| 0(0) | 1 | AIXFG | Flags that indicate what kind of pointer(s) the record contains: |
| | .... ...0<br>.... ...1<br>xxxx xxx. | | RBA pointer(s).<br>Prime-key pointer(s).<br>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. |

## Catalog

VSAM catalogs—the master catalog and any user catalogs—are built and processed by catalog management modules. Catalog management modules, via the catalog, enable a user to locate a data set, volume, index, or cluster by specifying a dsname or volume serial number. In addition, VSAM catalogs provide VSAM with the information required to allocate space for data sets, verify authorization to gain access to them, compile usage statistics on them, and relate RBAs to physical locations within data sets. The catalog indicates, therefore, much more than the simple location of data sets. The catalog maintains the relationship between a key-sequenced data set and its index, describes the location of VSAM data spaces and the data sets that reside in them, and describes the space that is available for new data sets.

The VSAM catalog is conceptually a key-sequenced VSAM data set divided into two key ranges called the low-address range and the high-address range. VSAM data set processing options, such as index record replication and sequence set with data, are utilized in both parts of the catalog. The catalog record size is variable; the catalog control interval size is 512 bytes. Figure 57 shows a VSAM catalog. The figure shows:

- The low-address range of the catalog, shown on the left, contains records that describe objects—data sets, indexes, alternate indexes, paths, upgrade sets, volumes, and clusters.
- The high-address range of the catalog, shown on the right, contains the true name (a data-set name, cluster name, or volume serial number) of an object specified by the user.

- The index, shown in the middle, points to both the low- and high-address parts of the catalog.

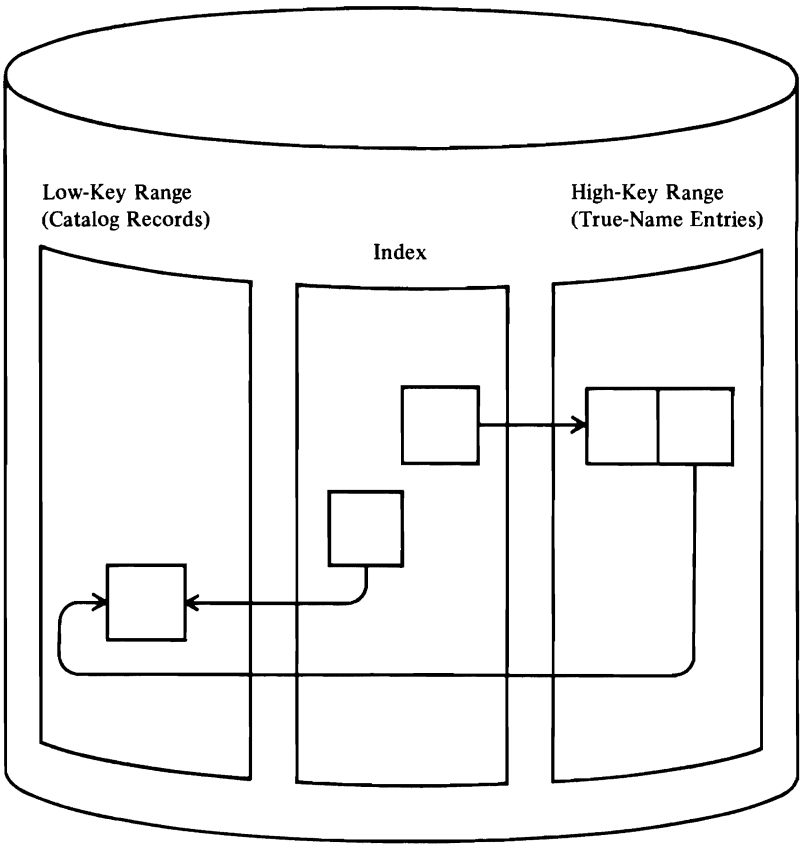



Figure 57. Parts of a VSAM Catalog

With the exception of catalog records that are built when the catalog is created and describe the catalog itself, catalog records are built as objects are cataloged. The order the records are in depends upon which portion of the catalog the records belong to. If the catalog records reside in the low-address part of the catalog, the records are ordered according to control interval number. As objects are cataloged, available control intervals are used. If the catalog records reside in the high-address part of the catalog, they are ordered according to their true name (data-set name or volume serial number).

Catalog management relies on VSAM record management for all record retrieval and storage. When a user specifies, a data-set name, the index points to a catalog record in the high-address part of the catalog that contains the data-set name; that record, in turn, contains the control interval number of the catalog record that describes the data set. Catalog management converts the control interval number to an RBA in the low-address part of the catalog.

### *High-Address Range of the Catalog*

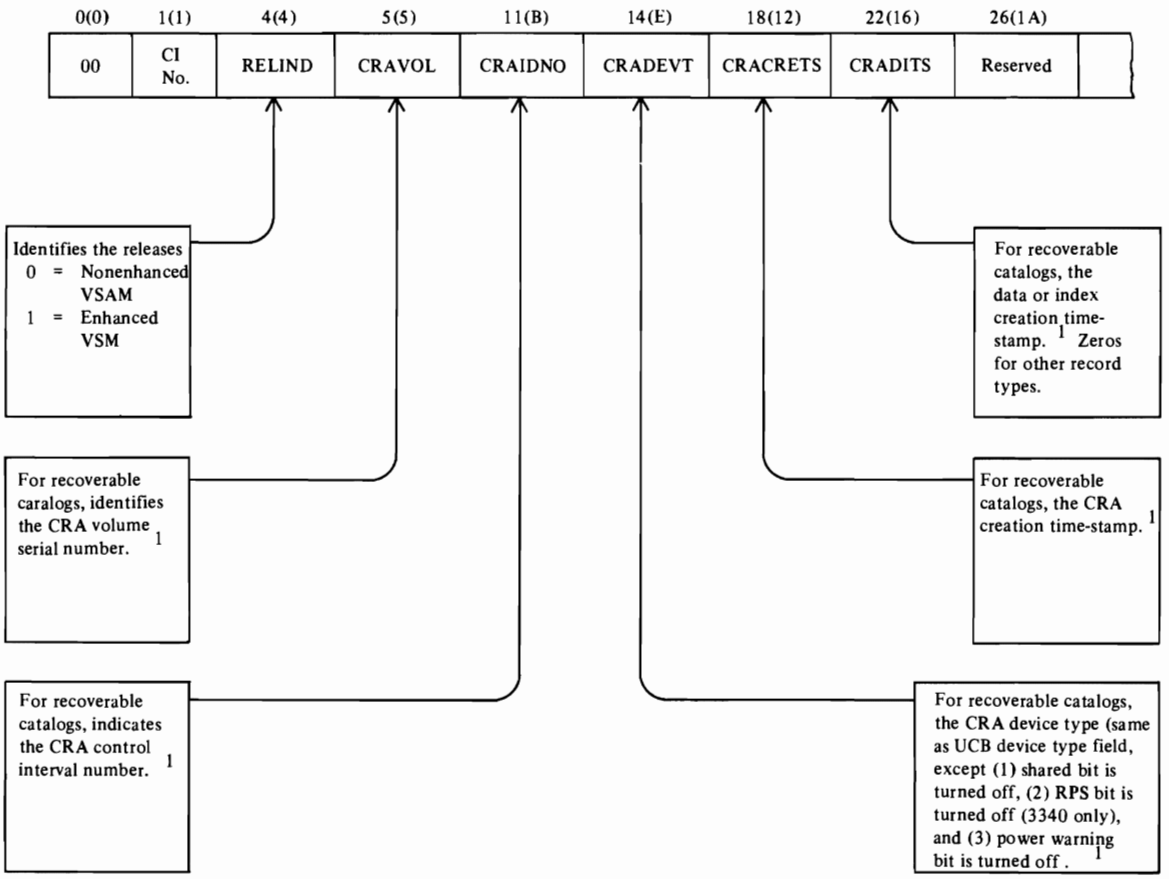
The high-address range of the catalog contains 47-byte True Name records in 512-byte control intervals. The True Name records associate user-specified names or volume serial numbers with the control interval number of the catalog record that describes the specified object.

### *Low-Address Range of the Catalog*

Records in the low-key range are 505 bytes long. Each record resides in its own control interval. Each record also identifies its record type. The low-key range of the catalog is made up of the following types of records:

- Control record, which describes the free control intervals in the low-address part of the catalog. The Control record is always the fourth record in the catalog. This record is record type "L."
- Free record, which marks the control interval in which it resides as available for use as another kind of catalog record. There is one Free record for each previously assigned control interval that is available for use. This record is record type "F."
- Cluster record, which describes a VSAM data-set cluster. This record contains the control interval number of a Data record and, if the VSAM data set is a key-sequenced data set, the control interval number of an Index record. There is one Cluster record for each VSAM cluster cataloged. This record is record type "C."
- Alternate index record, which relates the alternate index to its associated base cluster and also to any paths over it. This record is record type "G."
- Data and Index records, which describe data sets and indexes. There is one Data or Index record for each data set or index cataloged. These records are record types "D" and "I."
- Path record, which relates a base cluster and possibly an alternate index. This record is record type "R."
- Upgrade set record, which relates the data components and index components of the alternate indexes that make up the upgrade set. This record is record type "Y."
- NonVSAM record, which describes a data set organized differently than VSAM. There is one NonVSAM record for each nonVSAM data set cataloged. This record is record type "A."
- User-Catalog record, which describes a VSAM user catalog. There is one User-Catalog record for each user catalog connected to this master catalog. This record is record type "U."
- Volume record, which describes each VSAM data space on a volume, the data sets that reside in the data space, and the space available within the data space. There is one Volume record for each volume controlled by this catalog. This record is record type "V."
- Extension record, which contains overflow information from another catalog record. There are as many Extension records as are required to contain overflow information. This record is record type "W" when it is an extension of a Volume record; it is record type "E" when it is an extension of any other catalog record.

The Cluster, Data, Index, Alternate Index, Path, Upgrade, NonVSAM, Extension, and User-Catalog records have a common general format. Figure 58 shows the general format for these records.




[1] Contains zeros if the catalog is not recoverable or if there is no associated CRA entry.

Figure 58 (Part 1 of 2). Catalog Record—General Format

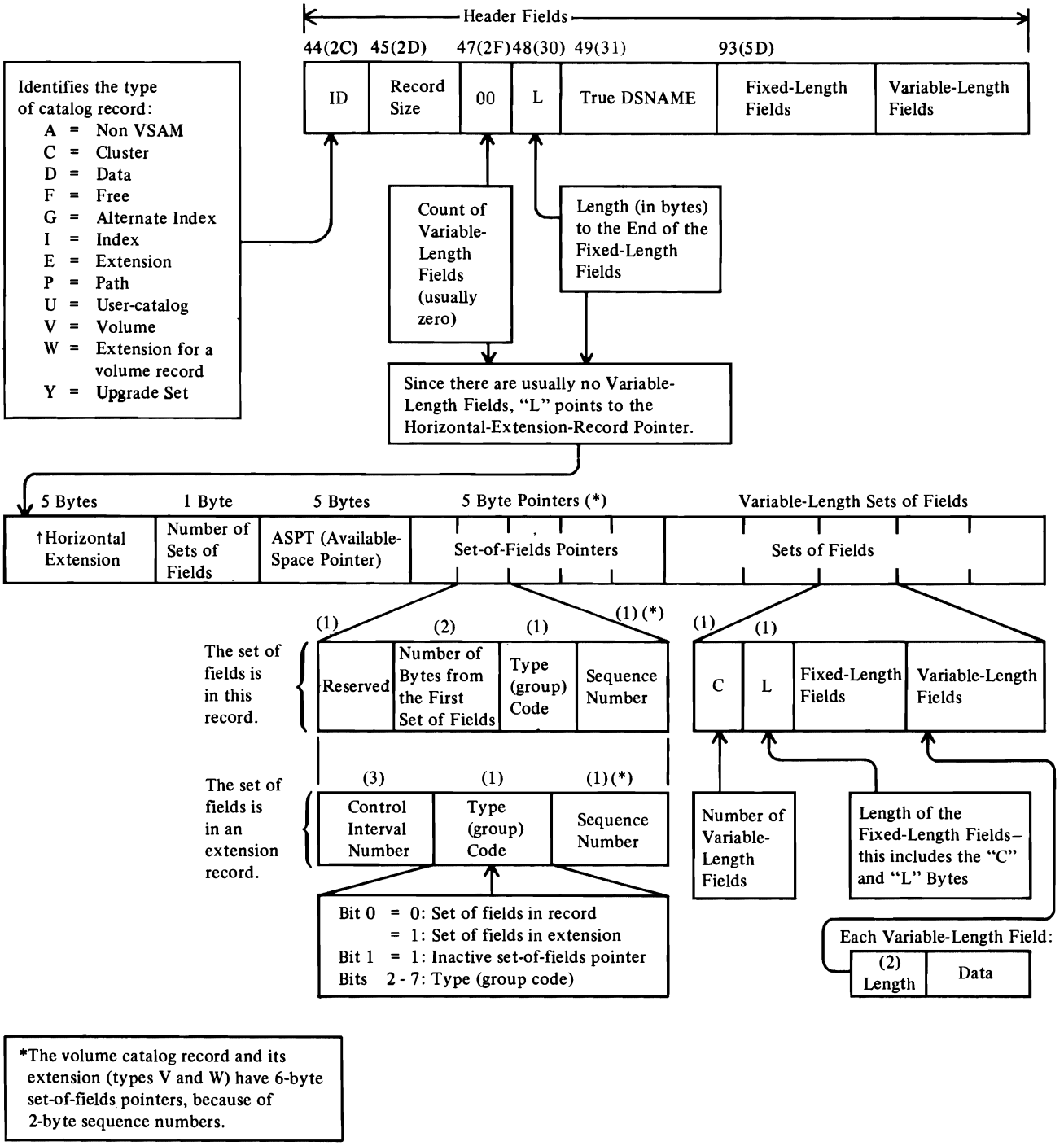


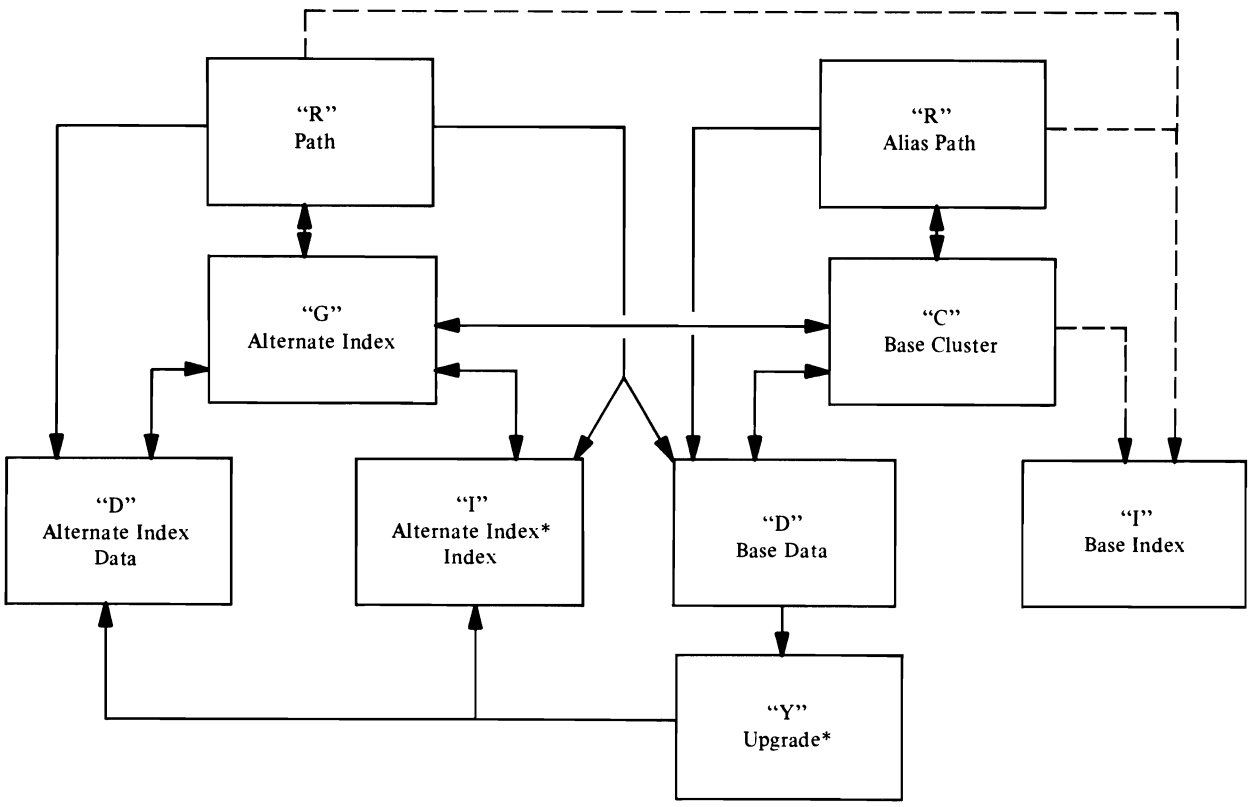

Figure 58 (Part 2 of 2). Catalog Record—General Format

## *Sets of Fields in the Catalog Records*

Related fields of information are grouped into sets of fields so they can be treated as a unit. For example, all fields relating to one volume on which a data set resides are grouped together. If a data set resides on three volumes, there are three sets of volume information; these sets of volume information fields are not necessarily contiguous. Each pointer to a set of fields, however, contains a code that identifies the kind of information the set of fields contains. It is possible for one record to contain many sets of fields.

Following are the sets of fields that can occur in Cluster, Alternate Index, Path, Upgrade, Data, Index, NonVSAM, and User-Catalog records:

- AMDSB (Access Method Data Set Statistics Block), which appears in Data Set and Index records. Only one copy of an AMDSB appears in a record. A pointer to AMDSB information contains a code of 1.
- Association information, which appears in Data, Index, Cluster, Alternate Index, Path, and Upgrade catalog records. Figure 59 illustrates the associations that can occur in these records. Each arrow represents an association. Associations shown by a broken line exist only when the base cluster is a key-sequenced data set. Multiple alternate indexes and paths may exist.
- Volume information, which appears in Data, Index, User-Catalog and NonVSAM records. This set of fields describes all of the direct-access




*The alternate index is in the upgrade set; therefore, the "Y" entry has associations pointing to the data and index of the alternate index.

Figure 59. Catalog Record Associations

device space allocated to the data set (or index, etc.) on a particular volume. A separate set of volume information fields is used to describe the space on each volume. If the data set's space on a volume is divided into key ranges, each key range is described in a separate set of volume information fields. As many sets of volume information fields as are required to describe allocated space can appear. A pointer to volume information contains a code of 3.

- Password information, which can appear in Data, Index, Alternate Index, Path, and Cluster records. This set of fields contains the security information for a data set (or index, etc.). Only one set of password information fields can appear. A pointer to password information contains a code of 4.

The Volume record can also contain sets of fields, as follows:

- Track allocation information (Space Map set of fields). This set of fields describes each track on the volume as allocated to a VSAM object or unallocated. Each volume record contains as many of these sets of fields as are required to describe the entire volume. A pointer to track allocation information contains a code of 5.
- VSAM data space information (Data Space Group set of fields). This set of fields describes a VSAM data space on the volume. One set of fields is required to describe each data space and its extents on the volume. A pointer to data-space information contains a code of 6.
- Data Set Directory Entry set of fields. This set of fields describes a data set that resides in a VSAM data space. One set of fields is required for each data set. A pointer to data set information contains a code of 8.

**Note:** If a Cluster, Alternate Index, Upgrade, Data, Index, NonVSAM, or Volume record is extended, these sets of fields (except for the AMDSB set of fields) are moved, as required, into an Extension record.

### *Catalog Records that Describe the Catalog*

Catalog records that describe the catalog as a data set are in fixed positions at the beginning of the catalog. The following table shows the control interval numbers of records that describe the catalog, the kind of catalog record each is, and the contents of each.

| Control Interval Number | Record Type | Contents |
|---|---|---|
| 0 | Data | Description of the data portion of the catalog. |
| 1 | Index | Description of the index portion of the catalog. |
| 2 | Cluster | Description of the catalog cluster. |
| 3 | Control | Catalog control record (CCR), which describes the catalog's free control intervals. |
| 4 | Extension | Extension of the index catalog record (control interval number 1). This Extension record contains a description of the high-level index extents of the catalog. |

**Catalog Records that Describe the Catalog**

| Control Interval Number | Record Type | Contents |
|---|---|---|
| 5 | Extension | Extension of the data catalog record (control interval number 0). This Extension record contains a description of the low-address data extents of the catalog. |
| 6 | Extension | Extension of the index catalog record (control interval number 1). This Extension record contains a description of the low-address index sequence set extents of the catalog. |
| 7 | Extension | Extension of the data catalog record (control interval number 0). This Extension record contains a description of the extents of the True Name records in the high-address part of the catalog. |
| 8 | Extension | Extension of the index catalog record (control interval number 1). This Extension record contains a description of the high-address index sequence set extents of the catalog. |
| 9 | Volume | Description of the track allocation and VSAM data spaces on this volume. |
| 10-13 | Volume | As many volume extension records as are necessary to describe the total space on the volume. |

When the catalog is built, there are two True Name records. One contains the catalog volume's serial number and points to control interval number 9. The other contains the catalog's name and points to control interval number 2.

### *Locating Fields in Catalog Records*

A field-name dictionary, which is part of the catalog management code, allows catalog management to locate fields within catalog records by name. The dictionary also allows for combination field names—each combination field name allows catalog management to locate a group of related fields.

Catalog records and the field-name dictionary are described in the topics that follow.

## Recoverable Catalog Support

Catalogs can be defined with an optional recovery attribute that allows data sets to be recovered or restored. Recovery is based on information that is recorded in the catalog and also in a catalog recovery area on the volumes owned by the catalog. The recovery area is established when the recoverable catalog acquires ownership of the volume. Thus, all volumes owned by a recoverable catalog contain catalog recovery area space.

Whenever records in a recoverable catalog are defined, deleted, or modified, the corresponding information in the catalog recovery area is updated to reflect the change. Although no specific commands are required to maintain the recovery area, certain volumes must be mounted during defines, alters, deletes, and any catalog entry modifications resulting from open, close, or end of volume activity. The volumes are:

ALTER The prime catalog recovery volume for the objects being altered.

DELETE All volumes that are reference by the entry being deleted and

the prime CRA volume.

DEFINE — All volumes that are referenced in the DEFINE command. Also, the first volume of the base cluster must be mounted when alternate indexes and paths are being defined.

Once a recoverable catalog is defined, it cannot be made nonrecoverable. Also, a recoverable catalog cannot be copied. A nonrecoverable catalog can be converted and made recoverable through Access Method Services commands. This conversion is not necessary unless there is a requirement for the recovery capability.

The Access Method Services commands used to achieve catalog recovery are described in *OS/VS2 SVS Independent Component: Access Method Services*. That book also contains specific instructions on how to make an existing master catalog recoverable. User catalogs can be converted by using the EXPORT, DELETE, DEFINE, and IMPORT commands.

This publication contains additional information about catalog recovery. In the "Method of Operation" chapter, Diagram EE3 describes the processing required to define a catalog recovery area. In the "Program Organization" chapter, detailed drawings in the "Catalog Management I/O Functions" section illustrate CRA I/O operations. The "Data Areas" chapter contains the description and the format of all the recovery area records.

## Catalog Recovery Area Record Descriptions

Catalogs that are defined with the recoverable attribute are associated with one or more CRAs (catalog recovery areas). A CRA is a VSAM entry-sequenced data set, and every volume owned by a recoverable catalog contains one CRA.

Each CRA contains three types of 512-byte records:

- Self-describing records
- Duplicate copies of VSAM catalog entry records
- CRA free records

Self-describing records and free records occupy control intervals 0 - 8 in the CRA. The control intervals and the specific record they contain are:

| CI Number | Record Type | Description |
|---|---|---|
| 0 | D | Data set record, which describes the CRA data component of the CRA cluster |
| 1 | F | Free record |
| 2 | C | Cluster record, which describes the CRA cluster |
| 3 | L | Control record, which manages control interval allocation in the CRA |
| 4 | F | Free record |
| 5 | E | Extension record, which is an extension of the data set record in control interval 0 |
| 6,7,8 | F | Free records |

The formats and contents of the self-describing records are shown later in this section.

Duplicate copies of catalog records are recorded in the CRA in control intervals 9 - n. The volume record for the CRA volume is in control interval 9. The format and content of these duplicated records are identical to their counterparts in the recoverable catalog associated with the CRA.

Each catalog record that is not self-describing is duplicated in a specific CRA. The table that follows shows which CRA contains a given catalog record. In the table, *initial volume* is the first volume on which space was allocated for the entity.

| Catalog Entry | Record Type | CRA |
|---|---|---|
| Volume records | V,W | Subject volume |
| KSDS cluster records | C,E | Initial prime index volume |
| KSDS data records | D,E | Initial prime index volume |
| KSDS index records | I,E | Initial prime index volume |
| AIX cluster record (KSDS) | G,E | Initial prime index volume of the base cluster |
| AIX data record (KSDS) | D,E | Initial prime index volume of the base cluster |
| AIX index record (KSDS) | I,E | Initial prime index volume of the base cluster |
| AIX cluster records (ESDS) | G,E | Initial base data volume |
| AIX data records (ESDS) | D,E | Initial base data volume |
| AIX index records (ESDS) | I,E | Initial base data volume |
| Path records (KSDS, no AIX) | R,E | Initial prime index volume |
| Path records (ESDS, no AIX) | R,E | Initial base data volume |
| Path records (KSDS, AIX) | R,E | Initial prime index volume of the base cluster |
| Path records (ESDS, AIX) | R,E | Initial base data volume |
| Upgrade records (KSDS) | Y,E | Initial prime index volume of the base cluster |
| Upgrade records (ESDS) | Y,E | Initial base data volume |
| NonVSAM records | A,E,U | Catalog volume |

## True Name Catalog Record Format

The True Name record associates the volume serial number, data-set name, or cluster name specified by the user with the control interval number of the catalog record that describes the object (volume, cluster, alternate index, path, or data set). True Name records are contained in the high-address part of the catalog and are pointed to by the catalog's index records. The True Name record is retrieved using key-sequenced processing. The catalog management modules convert the control interval number in the True Name record to an RBA for entry-sequenced processing.

True Name records are 47 bytes long; several might be contained in a catalog's (512-byte) control interval.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 44 | | Name of a data set, alternate index, path, or cluster, filled on the right with blanks, or a volume serial number, filled on the right with zeros, specified by the user. |
| 44 (2C) | 3 | | Control interval number of the catalog record that describes the object. |

# Catalog Control Record (CCR) Format

The catalog control record (CCR) is used by catalog management to control the allocation of control intervals in the low-address part of the catalog, where catalog records, excluding the True Name records and the index, reside. The CCR also shows the catalog's high-used and high-allocated RBA values. The catalog control record is the fourth record (control interval) in the catalog.

For a request of one catalog record, catalog management tries to use a record that was freed because of deletion. This process is done before using unassigned control interval. If more than one catalog record is needed, catalog management tries to use contiguous unassigned space in the current extent; if sufficient unassigned control interval is not available, records that have been deleted are used.

**Catalog Control Record (CCR) Format**

| Offset | Bytes and Bit Pattern | Description |
|---|---|---|
| 0 (0) | 44 | Key. |
| | | **Byte** — **Meaning**<br>0 — Zeros.<br>1-3 — Control interval number of this record.<br>4 — Release indicator<br>5-43 — Zeros. |
| 44 (2C) | 1 | Record type—"L." |
| 45 (2D) | 3 | Number of the highest control interval within the current extent.[1] |
| 48 (30) | 3 | Number of the next free control interval of those that have not been previously assigned.[1] |
| 51 (33) | 3 | Count of deleted control intervals, that is, the number of control intervals that are free because of deletion.[2] |
| 54 (36) | 3 | First deleted control interval in a chain of control intervals that are free because of deletion.[2] |

### Catalog Control Record Format

| Offset | Bytes and Bit Pattern | Description |
|---|---|---|
| *The following fields are used to keep track of the RBA values that denote the current logical end (high RBA) of parts of the catalog.* | | |
| **Note:** The low key range is the low-address part of the catalog; the high key range is the high-address part. | | |
| 57 (39) | 4 | Data, low key range: high-key RBA |
| 61 (3D) | 4 | Data, low key range: high-used RBA |
| 65 (41) | 4 | Data, low key range: high-allocated RBA |
| 69 (45) | 4 | Data, high key range: high-key RBA |
| 73 (49) | 4 | Data, high key range: high-used RBA |
| 77 (4D) | 4 | Data, high key range: high-allocated RBA |
| 81 (51) | 4 | Index, high level: high-used RBA |
| 85 (55) | 4 | Index, high level: high-allocated RBA |
| 89 (59) | 4 | Index, low key range—sequence set: high-used RBA |
| 93 (5D) | 4 | Index, low key range—sequence set: high-allocated RBA |
| 97 (61) | 4 | Index, high key range—sequence set: high-used RBA |
| 101 (65) | 4 | Index, high key range—sequence set: high-allocated RBA |

[1] This field is used to keep track of unassigned space within the current extent.

[2] This field is used to keep track of previously-used records that are now available for use as another catalog record.

## Free Catalog Record Format

The Free record indicates that the control interval in which it resides is free and points to the next control interval that is free because of deletion. Note that the free space in the catalog that has never been assigned is not represented by Free records; the Free record is used only to mark a record that was used and deleted.

| Offset | Bytes and Bit Pattern | Description | |
|---|---|---|---|
| 0 (0) | 44 | Key. | |
| | | **Byte** | **Meaning** |
| | | 0 | Zeros. |
| | | 1-3 | Control interval number of this record. |
| | | 4-43 | Zeros. |
| 44 (2C) | 1 | Record type—"F." | |
| 45 (2D) | 3 | Control interval number of the next free control interval. | |

# Data and Index Catalog Record Format

Data and Index records describe data sets and their indexes.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 1 | | Zeros. |
| 1 (1) | 3 | ENTIDNO | Control interval number of this record. |
| 4 (4) | 1 | RELIND | Release indicator: |
| | | | 0 = Nonenhanced VSAM;<br>1 = Enhanced VSAM. |
| 5 (5) | 6 | CRAVOL | CRA volume serial.[1] |
| 11 (B) | 3 | CRAIDNO | CRA control interval number.[1] |
| 14 (E) | 4 | CRADEVT | CRA device type.[1] |
| 18 (12) | 4 | CRACRETS | CRA creation time stamp. |
| 22 (16) | 4 | CRADITS | Data/index identifier timestamp. |
| 26 (1A) | 18 | | Zeros. |
| 44 (2C) | 1 | ENTYPE | Record type—"D" for a Data record or "I" for an Index record. |
| 45 (2D) | 2 | | Record length. |
| 47 (2F) | 1 | | Number of variable-length fields that precede the pointer to an extension record. Always zero. |
| 48 (30) | 1 | | Length of the fixed-length fields in this record, excluding any fixed-length fields following displacement 143 (X'8F'). This value is always equal to the displacement from the beginning of the record to the pointer to an extension record. |
| 49 (31) | 44 | ENTNAME | For a Data or Index record, the data set name. |
| 93 (5D) | 8 | OWNERID | Owner of the data set, specified when the data set was defined. |
| 101 (65) | 3 | DSETCRDT | Data set creation date, in packed-decimal form YDD, specified when the data set was defined. |
| 104 (68) | 3 | DSETEXDT | Data set expiration date, in packed-decimal form YDD, specified when the data set was defined. |
| 107 (6B) | 1 | ATTR1 | Data set attributes, which are defined using Access Method Services commands, as follows: |
| | 1... .... | | Speed, which indicates that storage for the data set or index is not to be preformatted before records are output. |
| | .1.. .... | | Unique, which indicates that this data set or index must reside in a data space all its own. |
| | ..1. .... | | The cluster associated with this component is reusable. |
| | ...1 .... | | Erase, which indicates that the data set or index is to be overwritten with binary zeros when deleted. |
| | .... 1... | | This catalog is recoverable. |
| | .... .1.. | | Inhibit update, which indicates that the data set or index is not to be updated. |

## Data and Index Catalog Record Format

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| | .... ..1. | | Temporary export, which indicates that the original copy of this data set or index is not to be deleted, even though another copy of it exists somewhere else. |
| | .... ...x | | Reserved. |
| 108 (6C) | 1 | ATTR2 | Data-set sharing attributes as follows: |
| | 00.. .... | | The data set can be shared by READ users *or* it can be used by one UPDATE/OUTPUT user. |
| | 01.. .... | | The data set can be shared by READ users *and* one UPDATE/OUTPUT user. |
| | 10.. .... | | The data set can be fully shared. |
| | 11.. .... | | The data set can be fully shared; with assistance supplied by VSAM. |
| | | | Data-set sharing attributes across systems, as follows: |
| | ..00 .... | | Reserved. |
| | ..01 .... | | Reserved. |
| | ..10 1... | | The data set can be fully shared. |
| | ..11 .... | | The data set can be fully shared; with assistance supplied by VSAM. |
| | .... ...1 | | Component is not usable. |
| | .... xxx. | | Reserved. |
| 109 (6D) | 1 | OPENIND | Open indicator flag; if this byte contains X'80', the data set is open for output. |
| 110 (6E) | 4 | BUFSIZE | Minimum buffer size. |
| 114 (72) | 3 | PRIMSPAC | Primary space allocation for the data set or index, specified when the data set or index was defined. |
| 117 (75) | 3 | SCONSPAC | Secondary space allocation for the data set or index, specified when the data set or index was defined. |
| 120 (78) | 1 | SPACOPTN | Space options flags. |
| | 10.. .... | | Track request, which indicates that space allocation was specified in tracks. |
| | 11.. .... | | Cylinder request, which indicates that space allocation was specified in cylinders. |
| | ..xx xxxx | | Reserved. |
| 121 (79) | 4 | HURBADS | High used RBA of the data set or index. |
| 125 (7D) | 4 | HARBADS | High allocated RBA of the data set or index. |
| 129 (81) | 4 | LRECL | For a Data record, the logical record size of the data set described by this Data record. For an Index record, always X 'FF's. |
| 133 (85) | 2 | USERINFO | User information for the DOS/VS indexed-sequential access method compatibility interface. |
| 135 (87) | 8 | EXCPEXIT | Exception exit. |
| *The following 6-byte entry contains control information for the sets of fields that follow it.* | | | |
| 143 (8F) | 5 | | Pointer to horizontal extension record. If this record is not continued in an extension record, this field contains zeros. |

## Data and Index Catalog Record Format

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 148 (94) | 1 | | The number of set of fields pointers that follow.[2]<br><br>**Note:** The first set-of-fields pointer contains sequence number = 0 and type code = 0. When bytes 0 to 2 are nonzero, this set-of-fields pointer points to the first (vertical) extension record that contains free space. The set-of-fields pointer is called the ASPT—available-space pointer. When bytes 0 to 2 are zero, the pointer is a dummy set-of-fields pointer—not available to be used to point to a set of fields. |
| 149 (95) | VL | | 5-byte pointers to sets of fields within the record.<br><br>**Byte** — **Meaning**<br>0 — Reserved.<br>1-2 — Displacement of the set of fields from the beginning of all sets of fields in this record.<br>3 — Bits 0 and 1 are set to zero. If bit 1 is on, the set of fields associated with this pointer has been deleted; the code in bits 2 through 7 of this byte and the sequence number in byte 4, however, are kept. Bits 2 through 7 contain a code describing the set of fields pointed to.[2]<br>4 — Sequence number of the set of fields pointed to by code.[3] For example, all sets of fields associated with a code of 2 are in one sequence. |
| | VL | | 5-byte pointers to sets of fields contained in vertical Extension records.<br><br>**Byte** — **Meaning**<br>0-2 — Control interval number of the extension record that contains this set of fields.<br>3 — Bits 0 and 1 are set to B'10'. Bits 2 through 7 contain a code that describes the set of fields pointed to.[3]<br>4 — Sequence number of the set of fields pointed to by code.[3] For example, all sets of fields associated with a code of 2 are in one sequence. |

[1] Zeros if the catalog is not recoverable or if there is no assocated CRA volume.

[2] Fields describing (a) the AMDSB, (b) the control interval number of a Cluster record associated with this record, (c) the volumes on which a data set resides, and (d) the password information associated with a data set are grouped into sets of fields. Pointers to each set of fields identify the type of information contained in each set of fields.

[3] If the pointer is associated with AMDSB information, the code is 1; with cluster information, the code is 2; with volume information, the code is 3; or with password information, the code is 4.

### *AMDSB (Access Method Data Set Statistics Block) Set of Fields Format*

The AMDSB set of fields contains a copy of the AMDSB control block, and is updated each time the data set is closed. This set of fields is associated with a pointer that contains a type (group) code of 1.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 2 | | Control information.<br>**Byte** **Meaning**<br>0 Count of the number of variable-length fields in this set of fields.<br>1 Hexadecimal displacement to the first variable-length field from the beginning of this set of fields. |
| 2 (2) | ;96 | AMDSBCAT | AMDSB—Access Method Data Statistics Block. See "Data Areas" for detailed information about the AMDSB. |

### *Association (Cluster) Set of Fields Format*

The control interval number of the cluster catalog record associated with the data or index catalog record is contained in the association set of fields. This set of fields is associated with a pointer that contains a type (group) code of 2.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 2 | | Control information.<br>**Byte** **Meaning**<br>0 Count of the number of variable-length fields in this set of fields.<br>1 Hexadecimal displacement to the first variable-length field from the beginning of this set of fields. |

### Association (Cluster) Set of Fields Format

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 2 (2) | 1 | TYPE | "C," which indicates that this record is associated with a Cluster record, or "G", which indicates this is the data component or the index component of an alternate index. The data component of a cluster may have an additional "Y" association if the cluster has an alternate index that is part of the upgrade set. |
| 3 (3) | 3 | NAME | Control interval number of the Cluster or Alternate Index record associated with this record. |

## *Volume Information Set of Fields Format*

All extents allocated to the data set, index, or data set's key range on a volume are described by a volume information set of fields. This set of fields is associated with a pointer that contains a type (group) code of 3.

| Offset | Byte and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 2 | | Control information.<br>**Byte** — **Meaning**<br>0 — Count of the number of variable-length fields in this set of fields.<br>1 — Hexadecimal displacement to the first variable-length field from the beginning of this set of fields. |
| 2 (2) | 4 | DEVTYP | Device type. |
| 6 (6) | 6 | VOLSER | Volume serial number. |
| 12 (C) | 2 | FILESEQ | File sequence number. (This field is provided for compatibility with the OS/VS catalog.) |

## Data and Index Catalog Record: Volume Information Set of Fields Format

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 14 (E) | 1 | VOLFLG | Volume flags, as follows: |
| | 1... .... | | Prime, which indicates that this volume was allocated when the data set was defined or that a data set that is not divided into parts according to key has been extended to this volume. |
| | .1.. .... | | Candidate, which indicates that this volume is available for use by the data set described by this record. |
| | ..1. .... | | Overflow, which indicates that this volume is being used by a data set that is divided into parts according to key, but this volume was not allocated when the data set was defined. |
| | ...x xxxx | | Reserved |
| 15 (F) | 1 | NOEXTNT | Number of extents allocated in this set of extents on this volume for this data set. |
| 16 (10) | 4 | HKRBA | RBA of the data control interval with the high key. |
| 20 (14) | 4 | HURBA | High-used RBA. |
| 24 (18) | 4 | HARBA | High-allocated RBA. |
| 28 (1C) | 4 | PHYBLKSZ | Blocksize. |
| 32 (20) | 2 | NOBLKTRK | Number of blocks per track. |
| 34 (22) | 2 | NOTRKAU | Number of tracks per allocation unit. |
| 36 (24) | 1 | ITYPEXT | Flags: |
| | 1... .... | | In an index record: the sequence set is with the data. |
| | .1.. .... | | The extents are not preformatted |
| | ..xx xxxx | | Reserved |
| 37 (25) | 2 | DSDIRSN | Data set directory sequence number in the volume record. |
| 39 (27) | VL | LOKEYV | Low key on the volume. This field can be a maximum of 64 bytes long; the first two bytes indicate the length of the field. |
| | VL | HIKEYV | High key on the volume. This field can be a maximum of 64 bytes long; the first two bytes indicate the length of the field. |
| | VL | EXTENT | This field contains a 2-byte length field, followed by a 20-byte field for each extent. The 20-byte field describes the start and end of the extent, in the form SSCCHHCCHHTTDDDDDDDD, where SS is the data space extent's sequence number, CCHHCCHH is the low and high cylinder and head, TT is the number of tracks, and DDDDDDDD is the low and high RBA of the extent. |

### *Password Set of Fields Format*

Password information, if any, is contained in the password set of fields. This set of fields is associated with a pointer that contains a type (group) code of 4.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 2 | | Control information<br>**Byte** **Meaning**<br>0 Count of the number of variable-length fields in this set of fields.<br>1 Hexadecimal displacement to the first variable-length field from the beginning of this set of fields. |
| 2 (2) | 32 | PASSWORD | Four eight-character passwords, in the following order: Master, Control Interval, Update, and Read. |
| 34 (22) | 8 | PASSPRMT | Password prompting code name that allows the operator to provide the correct password without displaying the data set name. |
| 42 (24) | 2 | PASSATMP | Maximum number of attempts allowed for the operator or TSO operator to provide correct password. |
| 44 (2C) | 8 | USVRMDUL | Name of user's security-verification module, if any. |
| 52 (34) | VL | USERAREC | User-authorization record. This field can be a maximum of 256 bytes long. |

## Cluster Catalog Record Format

The Cluster record describes a data set and its index, if any, and may point to one or more alternate indexes.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 1 | | Zeros. |
| 1 (1) | 3 | ENTIDNO | Control interval number of this record. |
| 4 (4) | 1 | RELIND | Release indicator:<br>0 = Nonenhanced VSAM;<br>1 = Enhanced VSAM. |
| 5 (5) | 6 | CRAVOL | CRA volume serial.[1] |
| 11 (B) | 3 | CRAIDNO | CRA control interval number.[1] |
| 14 (E) | 4 | CRADEVT | CRA device type.[1] |
| 18 (12) | 4 | CRACRETS | CRA creation time stamp. |
| 22 (16) | 4 | CRADITS | Data/index identifier timestamp. |
| 26 (1A) | 18 | | Zeros. |
| 44 (2C) | 1 | ENTYPE | Record type—"C." |
| 45 (2D) | 2 | | Record length. |
| 47 (2F) | 1 | | Number of variable-length fields that precede the pointer to an extension record. Always zero. |

## Cluster Catalog Record Format

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 48 (30) | 1 | | Length of the fixed-length fields in this record, excluding any fixed-length fields that follow displacement 106 (6A). This value is always equal to the displacement from the beginning of the record to the pointer to an extension record. |
| 49 (31) | 44 | ENTNAME | Name of the cluster described by this record. |
| 93 (5D) | 8 | OWNERID | Owner of the data set, specified when the data set was defined. |
| 101 (65) | 3 | DSETCRDT | Data set creation date, in packed-decimal form YDD, specified when the data set was defined. |
| 104 (68) | 3 | DSETEXDT | Data set expiration date, in packed-decimal form YDD, specified when the data set was defined. |
| *The following six-byte entry contains control information for the sets of fields that follow it.* | | | |
| 107 (6B) | 5 | | Pointer to horizontal extension record. If this record is not continued on an extension record, this field contains zeros. |
| 112 (70) | 1 | | The number of set-of-field pointers that follow.[2]<br><br>**Note:** The first set-of-fields pointer may be special (meaning that the field-name dictionary permits catalog management to locate information that is not contained in catalog records); if this is the case, its sequence number = 0 and type code = 0. When bytes 0 to 2 are nonzero, this set-of-fields pointer points to the first (vertical) extension record that contains free space. The set-of-fields pointer is called the ASPT—available-space pointer. When bytes 0 to 2 are zero, the pointer is a dummy set-of-fields pointer—not available to be used to point to a set of fields. |
| 113 (71) | VL | | 5-byte pointers to sets of fields within the record.<br><br>**Byte** **Meaning**<br><br>0 Reserved.<br><br>1-2 Displacement of the set of fields from the beginning of all sets of fields in this record.<br><br>3 Bits 0 and 1 are set to zero. If bit 1 is on, the set of fields associated with this pointer has been deleted; the code in bits 2 through 7 of this byte and the sequence number in byte 4, however, are kept. Bits 2 through 7 contain a code describing the set of fields pointed to.[3]<br><br>4 Sequence number of the set of fields pointed to by code.[3] For example, all |

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| | | | sets of fields associated with a code of 2 are in one sequence. |
| | VL | | 5-byte pointers to sets of fields contained in vertical Extension records. |
| | | | **Byte** **Meaning** |
| | | | 0-2 Control interval number of the extension record that contains this set of fields. |
| | | | 3 Bits 0 and 1 are set to B'10'. Bits 2 through 7 contain a code that describes the set of fields pointed to.[3] |
| | | | 4 Sequence number of the set of fields pointed to by code.[3] For example, all sets of fields associated with a code of 2 are in one sequence. |

[1] Zeros if the catalog is not recoverable or if there is no associated CRA entry.

[2] Fields describing (a) the control interval number of a Data or Index record associated with this cluster or (b) the password information associated with a data set are grouped into sets of fields. Pointers to each set of fields identify the type of information contained in each set of fields.

[3] If the pointer is associated with cluster information (the control interval number of a Data or Index record), the code is 2; with password information, the code is 4.

## *Association (Data and Index) Set of Fields Format*

The control interval number of the Data and Index Catalog Record associated with the cluster is contained in an association set of fields. This set of fields is associated with a pointer that contains a type (group) code of 2.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 2 | | Control information. |
| | | | **Byte** **Meaning** |
| | | | 0 Count of the number of variable-length fields in this set of fields. |
| | | | 1 Hexadecimal displacement to the first variable-length field from the beginning of this set of fields. |
| 2 (2) | 1 | TYPE | If this entry describes an Index Record, "I"; if this entry describes a Data record, "D;" if this entry describes an alternate index entry, "G." |
| 3 (3) | 3 | NAME | Control interval number of a Data, Index, or Alternate Index record that is part of the cluster described by this record. |

### *Password Set of Fields Format*

Password information, if any, is contained in the password set of fields. This set of fields is associated with a pointer that contains a type (group) code of 4.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 2 | | Control information.<br>**Byte** **Meaning**<br>0 Count of the number of variable-length fields in this set of fields.<br>1 Hexadecimal displacement to the first variable-length field from the beginning of this set of fields. |
| 2 (2) | 32 | PASSWORD | Four eight-character passwords, in the following order: Master, Control Interval, Update, and Read. |
| 34 (22) | 8 | PASSPRMT | Password prompting code name that allows the operator to provide the correct password without displaying the data set name. |
| 42 (2A) | 2 | PASSATMP | Maximum number of attempts allowed for the operator or TSO operator to provide correct password. |
| 44 (2C) | 8 | USVRMDUL | Name of the user's security-verification module, if any. |
| 52 (34) | VL | USERAREC | User-authorization record. This field can be a maximum of 256 bytes long. |

## Alternate Index Catalog Record Format

The alternate index record describes the data and index components associated with the alternate index. In addition, it points to the related cluster entry and it can point to one or more path entries. The alternate index grouping is similar to the grouping of a key-sequenced data set except for different record types ("G" rather than "C").

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 1 | | Zeros. |
| 1 (1) | 3 | ENTIDNO | Control interval number of this record. |
| 4 (4) | 1 | RELIND | Release indicator:<br>0=Nonenhanced VSAM;<br>1=Enhanced VSAM. |
| 5 (5) | 6 | CRAVOL | CRA volume serial. |
| 11 (B) | 3 | CRAIDNO | CRA control interval number. |
| 14 (E) | 4 | CRADEVT | CRA device type. |
| 18 (12) | 4 | CRACRETS | CRA creation time stamp. |
| 22 (16) | 22 | | Zeros. |
| 44 (2C) | 1 | ENTYPE | Record type—"G." |
| 45 (2D) | 2 | | Record length. |
| 47 (2F) | 1 | | Number of variable-length fields that precede the pointer to an extension record (always 0). |

**Alternate Index Catalog Record Format**

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 48 (30) | 1 | | Length of nonrepeating fixed-length fields (always X'6C'). |
| 49 (31) | 44 | ENTNAME | Name of the alternate index described by this record. |
| 93 (5D) | 8 | OWNERID | Owner of the alternate index described by this record; specified when the alternate index was defined. |
| 101 (65) | 3 | DSETCRDT | Alternate index creation date, in the packed decimal form YDD. |
| 104 (68) | 3 | DSETEXDT | Alternate index expiration date, in the packed decimal form YDD. |
| 107 (6B) | 1 | RGATTR | Alternate index attributes: |
| | 1... .... | | If there is an association in the upgrade entry, indicates that this alternate index is a member of the upgrade set. |
| | .xxx xxxx | | Reserved. |
| *The following 6-byte entry contains control information for the sets of fields that follow it.* | | | |
| 108 (6C) | 5 | | Pointer to the horizontal extension record. If this record is not continued on an extension record, this field contains zeros. |
| 113 (71) | 1 | | The number of set of fields pointers that follow.[1] |
| | | | **Note:** The first set-of-fields pointer may be special (meaning that the field-name dictionary permits catalog management to locate information that is not contained in catalog records); if this is the case, its sequence number = 0 and type code = 0. When bytes 0 to 2 are nonzero, this set-of-fields pointer points to the first (vertical) extension record that |

contains free space. The set-of-fields pointer is called the ASPT—available-space pointer. When bytes 0 to 2 are zero, the pointer is a dummy set-of-fields pointer—not available to be used to point to a set of fields.

| | | | |
|---|---|---|---|
| 114 (72) | VL | | 5-byte pointers to sets of fields within the record. |

| Byte | Meaning |
|---|---|
| 0 | Reserved. |
| 1-2 | Displacement of the set of fields from the beginning of all sets of fields in this record. |
| 3 | Bits 0 and 1 are set to zero. If bit 1 is on, the set of fields associated with this pointer has been deleted; the type (group) code in bits 2 through 7 of this byte and the sequence number in byte 4, however, are kept. Bits 2 through 7 contain a type (group) code describing the set of fields pointed to. |

**Alternate Index Catalog Record Format**

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| | | | 4 Sequence number of set of fields pointed to by code. For example, all sets of fields associated with a code of 2 are in one sequence. |
| | VL | | 5-byte pointers to sets of fields contained in vertical extension records. |
| | | | **Byte** **Meaning** |
| | | | 0-2 Control interval number of the extension record that contains this set of fields. |
| | | | 3 Bits 0 and 1 are set to B'10'. Bits 2 through 7 contain a type (group) code that describes set of fields pointed to. |
| | | | 4 Sequence number of the set of fields pointed to, by type (group) code. For example, all sets of fields associated with a code of 2 are in one sequence. |

[1] Fields describing (a) the control interval number of a Data or Index catalog record associated with this cluster, or (b) the password information associated with a data set are grouped into sets of fields. Pointers to each set of fields identify the type of information contained in each set of fields.

## *Association Set of Fields Format*

The associations in this entry are partially ordered; however, no assumptions should be made as to the relative placement or physical position of these associations in the alternate index record.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 2 | | Control information. |
| 2 (2) | 1 | TYPE | Record type pointed to by the following control interval number. |
| 3 (3) | 3 | NAME | Control interval number. |

The association ordering by record type is:

| Record Type | Description |
|---|---|
| D | Alternate index data (D) component association with occurrence sequence number = 1. |
| I | Alternate index index (I) component association with occurrence sequence number = 2. |
| C | Base cluster (C) component association with occurrence sequence number = 3. |
| R | Maximum of 252 path (R) associations. |

### *Password Set of Fields Format*

Password information, if any, is contained in the password set of fields. This set of fields is associated with a pointer that contains a type (group) code of 4.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 2 | | Control information.<br>**Byte** — **Meaning**<br>0 — Count of the number of variable-length fields in this set of fields.<br>1 — Hexadecimal displacement to the first variable-length field from the beginning of this set of fields. |
| 2 (2) | 32 | PASSWORD | Four eight-character passwords, in the following order: MASTER, CONTROL INTERVAL, UPDATE, and READ-ONLY. |
| 34 (22) | 8 | PASSPRMT | Password prompting code name that allows the operator to provide the correct password without displaying the data set's DSNAME. |
| 42 (2A) | 2 | PASSATMP | Maximum number of attempts allowed for the operator to provide the correct password. |
| 44 (2C) | 8 | USVRMDUL | Name of user's security-verification module, if any. |
| 52 (34) | VL | USERAREC | User-authorization record. This field can be a maximum of 256 bytes long. |

## Path Catalog Record Format

The path record describes an alternate index and its associated base data set to give an alternate, logical view of the base data set. It also may be used as an alias for a base data set to inhibit upgrade set unit allocation.

| Offset | Bytes and Bit Patterns | Field Name | Description |
|---|---|---|---|
| 0 (0) | 1 | | Zeros. |
| 1 (1) | 3 | ENTIDNO | Control interval number of this record. |
| 4 (4) | 1 | RELIND | Release indicator:<br>0=Nonenhanced VSAM;<br>1=Enhanced VSAM. |
| 5 (5) | 6 | CRAVOL | CRA (catalog recovery area) volume serial. |
| 11 (B) | 3 | CRAIDNO | CRA control interval number. |
| 14 (E) | 4 | CRADEVT | CRA device type. |
| 18 (12) | 4 | CRACRETS | CRA creation time stamp. |
| 22 (16) | 22 | | Zeros. |
| 44 (2C) | 1 | ENTYPE | Record type—"R." |
| 45 (2D) | 2 | | Record length. |
| 47 (2F) | 1 | | Number of variable-length fields that precede the address of an extension record (always zero). |
| 48 (30) | 1 | | Length of nonrepeating fixed-length fields (always X'6C'). |

## Path Catalog Record Format

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 49 (31) | 44 | ENTNAME | Name of the path described by this record. |
| 93 (5D) | 8 | OWNERID | Owner of the path; specified when the path was defined. |
| 101 (65) | 3 | DSETCRDT | Path creation date; in the packed-decimal form YDD. |
| 104 (68) | 3 | DSETEXDT | Path expiration date; in the packed-decimal form YDD. |
| 107 (6B) | 1 | RGATTR | Path attributes: |
| | 1... .... | | Include upgrade set during unit allocation and when opening this path. |
| | .xxx xxxx | | Reserved. |
| *The following 6-byte entry contains control information for the sets of fields that follow it.* | | | |
| 108 (6C) | 5 | | Pointer to the horizontal extension record. If this record is not continued on an extension record, this field contains zeros. |
| 113 (71) | 1 | | The number of set of fields pointers that follow.[1]<br><br>**Note:** The first set-of-fields pointer contains sequence number = 0 and type code = 0. When bytes 0 to 2 are nonzero, this set-of-fields pointer points to the first (vertical) extension record that contains free space. The set-of-fields pointer is called the ASPT—available-space pointer. When bytes 0 to 2 are zero, the pointer is a dummy set-of-fields pointer—not available to be used to point to a set of fields. |
| 114 (72) | VL | | 5-byte pointers to sets of fields within the record. |

| Byte | Meaning |
|---|---|
| 0 | Reserved. |
| 1-2 | Displacement of the set of fields from the beginning of all sets of fields in this record. |
| 3 | Bits 0 and 1 are set to zero. If bit 1 is on, the set of fields associated with this pointer has been deleted; the type (group) code in bits 2 through 7 of this byte and the sequence number in byte 4, however, are kept. Bits 2 through 7 contain a type (group) code describing the set of fields pointed to. |
| 4 | Sequence number of the set of fields pointed to by code. For example, all sets of fields associated with a code of 2 are in one sequence. |

| | | | |
|---|---|---|---|
| | VL | | 5-byte pointers to sets of fields contained in vertical extension records. |

### Path Catalog Record Format

| Offset | Bytes and Bit Pattern | Field Name | Description | |
|---|---|---|---|---|
| | | | **Byte** | **Meaning** |
| | | | 0-2 | Control interval number of the extension record that contains this set of fields. |
| | | | 3 | Bits 0 and 1 are set to B'10.' Bits 2 through 7 contain a type (group) code that describes set of fields pointed to. |
| | | | 4 | Sequence number of the set of fields pointed to, by type (group) code. For example, all sets of fields associated with a code of 2 are in sequence. |

[1] Fields describing (a) the control interval number of a Data or Index catalog record associated with this cluster, or (b) the password information associated with a data set are grouped into sets of fields. Pointers to each set of fields identify the type of information contained in each set of fields.

## *Association Set of Fields Format*

The associations in this entry are ordered in the sense that each association occurrence has a defined group occurrence sequence number; however, no assumptions should be made as to the relative placement or physical position of these associations in the path record.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 2 | | Control information. |
| 2 (2) | 1 | TYPE | Record type which is pointed to by the following control interval number. |
| 3 (3) | 3 | NAME | Control interval number. |

If this record describes a path over an alternate index, the association ordering by record type is:

| Record Type | Description |
|---|---|
| G | Alternate index (G) entry association with occurrence sequence number = 1. |
| D | Alternate index data (D) component association with occurrence sequence number =2. |
| I | Alternate index index (I) component association with occurrence sequence number = 3. |
| D | Base data (D) component association with occurrence sequence number = 4. |
| I | Base index (I) component association with occurrence sequence number = 5. This association exists only if the base cluster is a key-sequenced data set. |

If this record describes a path over a base cluster, the association ordering by record type is:

| Record Type | Description |
|---|---|
| C | Base cluster (C) component association with occurrence sequence number = 1. |
| D | Base data (D) component association with occurrence sequence number = 2. |
| I | Base index (I) component association with occurrence sequence number = 3. This association exists only if the base cluster is a key-sequenced data set. |

## *Password Set of Fields Format*

Password information, if any, is contained in the password set of fields. This set of fields is associated with a pointer that contains a type (group) code of 4.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 2 | | Control information.<br>**Byte** — **Meaning**<br>0 — Count of the number of variable-length fields in this set of fields.<br>1 — Hexadecimal displacement to the first variable-length field from the beginning of this set of fields. |
| 2 (2) | 32 | PASSWORD | Four eight-character passwords, in the following order: MASTER, CONTROL INTERVAL, UPDATE, and READ-ONLY. |
| 34 (22) | 8 | PASSPRMT | Password prompting code name that allows the operator to provide the correct password without displaying the data set's DSNAME. |
| 42 (2A) | 2 | PASSATMP | Maximum number of attempts allowed for the operator to provide the correct password. |
| 44 (2C) | 8 | USVRMDUL | Name of user's security-verification module, if any. |
| 52 (34) | VL | USERAREC | User-authorization record. This field can be a maximum of 256 bytes long. |

# Upgrade Catalog Record Format

The upgrade record describes all the alternate indexes that make up the upgrade set. It is pointed to by an association in the base data component.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 1 | | Zeros. |
| 1 (1) | 3 | ENTIDNO | Control interval number of this record. |
| 4 (4) | 1 | RELIND | Release indicator:<br>0=Nonenhanced VSAM;<br>1=Enhanced VSAM |
| 5 (5) | 6 | CRAVOL | CRA volume serial number. |
| 11 (B) | 3 | CRAIDNO | CRA control interval number. |
| 14 (E) | 4 | CRADEVT | CRA device type. |
| 18 (12) | 4 | CRACRETS | CRA creation timestamp. |
| 22 (16) | 22 | | Zeros. |
| 44 (2C) | 1 | ENTYPE | Record type—"Y." |
| 45 (2D) | 2 | | Record length. |
| 47 (2F) | 1 | | Number of variable-length records that precede the pointer to an extension record (always 0). |
| *The following 6-byte entry contains control information for the sets of fields that follow it.* | | | |
| 49 (31) | 5 | | Pointer to the horizontal extension record. If this record is not continued on an extension record, this field contains zeros.<br><br>The number of set of fields pointers that follow.[1]<br><br>**Note:** The first set-of-fields pointer contains sequence number = 0 and type code = 0. When bytes 0 to 2 are nonzero, this set-of-fields pointer points to the first (vertical) extension record that contains free space. The set-of-fields pointer is called the ASPT—available-space pointer. When bytes 0 to 2 are zero, the pointer is a dummy set-of-fields pointer—not available to be used to point to a set of fields. |
| 55 (37) | VL | | 5-byte pointers to sets of fields within the record. |

| Byte | Meaning |
|---|---|
| 0 | Reserved. |
| 1-2 | Displacement of the set of fields from the beginning of all sets of fields in this record. |
| 3 | Bits 0 and 1 are set to zero. If bit 1 is on, the set of fields associated with this pointer has been deleted; the type (group) code in bits 2 through 7 of this byte 4, however, are kept. Bits 2 through 7 contain a type (group) code describing the set of fields pointed to. |

### Upgrade Catalog Record Format

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| | | | 4 — Sequence number of the set of fields pointed to by code. For example, all sets of fields associated with a code of 2 are in one sequence. |
| | VL | | 5-byte pointers to sets of fields contained in vertical extension records. |
| | | | **Byte** — **Meaning** |
| | | | 0-2 — Control interval number of the extension record that contains this set of fields. |
| | | | 3 — Bits 0 and 1 are set to B'10.' Bits 2 through 7 contain a type (group) code that describes set of fields pointed to. |
| | | | 4 — Sequence number of the set of fields pointed to, by type (group) code. For example, all sets of fields associated with a code of 2 are in one sequence. |

[1] Fields describing (a) the control interval number of a Data or Index catalog record associated with this cluster, or (b) the password information associated with a data set are grouped into sets of fields. Pointers to each set of fields identify the type of information contained in each set of fields.

## *Association Set of Fields Format*

The associations in this entry are actually twin associations consisting of a data association and index association.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 2 | | Control information. |
| 2 (2) | 1 | TYPE | "D", which indicates the following control interval number is for the data component of an alternate index in the upgrade set. |
| 3 (3) | 3 | NAME | Control interval number of the alternate index data component. |
| 6 (6) | 1 | TYPE2 | "I", which indicates the following control interval number is for the index component of an alternate index in the upgrade set. |
| 7 (7) | 3 | NAME2 | Control interval number of the alternate index index component. |

These twin associations exist only in upgrade records (type "Y"), and the set of twin associations in any given upgrade record entry is always unique.

# NonVSAM Catalog Record Format

The NonVSAM record describes a data set organized differently from VSAM data set organization.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 1 | | Zeros. |
| 1 (1) | 3 | ENTIDNO | Control interval number of this record. |
| 4 (4) | 1 | RELIND | Release indicator:<br>0=Nonenhanced VSAM;<br>1=Enhanced VSAM. |
| 5 (5) | 6 | CRAVOL | CRA volume serial.[1] |
| 11 (B) | 3 | CRAIDNO | CRA control interval number.[1] |
| 14 (E) | 4 | CRADEVT | CRA device type.[1] |
| 18 (12) | 4 | CRACRETS | CRA creation time stamp. |
| 22 (16) | 4 | CRADITS | Data/index identifier time stamp. |
| 26 (1A) | 18 | | Zeros. |
| 44 (2C) | 1 | ENTYPE | Record type—"A." |
| 45 (2D) | 2 | | Record length. |
| 47 (2F) | 1 | | Number of variable-length fields that precede the pointer to an extension record. Always zero. |
| 48 (30) | 1 | | Length of the fixed-length fields in this record, excluding any fixed-length fields following displacement 92 (5C). This value is always equal to the displacement from the beginning of the record to the pointer to an extension record. |
| 49 (31) | 44 | ENTNAME | Name of the data set described by this record. |
| 93 (5D) | 8 | OWNERID | Owner of the data set; specified when the data set was defined. |
| 101 (65) | 3 | DSETCRDT | Date data set was created; in packed-decimal form YDD. |
| 104 (68) | 3 | DSETEXDT | Date data set expires; in packed-decimal form YDD. |
| *The following 6-byte entry contains control information for the set of fields that follow it.* | | | |
| 107 (6B) | 5 | | Pointer to horizontal extension record. If this record is not continued on an extension record, this field contains zeros. The NonVSAM catalog record is usually not extended. |
| 112 (70) | 1 | | The number of set-of-fields pointers that follow.[2]<br><br>**Note:** The first set-of-fields pointer contains sequence number = 0 and type code = 0. When bytes 0 to 2 are nonzero, this set-of-fields pointer points to the first (vertical) extension record that contains free space. The set-of-fields pointer is called the ASPT—available-space pointer. When bytes 0 to 2 are zero, the pointer is a dummy |

## NonVSAM Catalog Record Format

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| | | | set-of-fields pointer—not available to be used to point to a set of fields. |
| 113 (71) | VL | | 5-byte pointers to sets of fields within the record. |
| | | | **Byte** **Meaning** |
| | | | 0 Reserved. |
| | | | 1-2 Displacement of the set of fields from the beginning of all sets of fields in this record. |
| | | | 3 Bits 0 and 1 are set to zero. If bit 1 is set on, the set of fields associated with this pointer has been deleted; the code in bits 2 through 7 and the sequence number in byte 4, however, are kept. Bits 2 through 7 contain a value of 3, which indicates that the set of fields pointed to contains information about a volume on which this data set resides. |
| | | | 4 Sequence number of the set of fields pointed to. |
| | VL | | 5-byte pointers to sets of fields contained in vertical Extension records. |
| | | | **Byte** **Meaning** |
| | | | 0-2 Control interval number of the extension record that contains this set of fields. |
| | | | 3 Bits 0 and 1 are set to B'10'. Bits 2 through 7 contain a value of 3, which indicates that the set of fields pointed to contains information about a volume on which this data set resides. |
| | | | 4 Sequence number of the set of fields pointed to. |

[1] Zeros if the catalog is not recoverable or if there is no associated CRA volume.

[2] Fields describing the volumes on which a data set resides are grouped into sets of fields. Pointers to each set of fields identify the type of information contained in the set of fields.

### *Volume Information Set of Fields Format*

Each volume that contains space allocated to the nonVSAM data set is described by a volume information set of fields. This set of fields is associated with a pointer that contains a type (group) code of 3.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 2 | | Control information.<br>**Byte** **Meaning**<br>0 Count of the number of variable-length fields in this set of fields.<br>1 Hexadecimal displacement to the first variable-length field from the beginning of this set of fields. |
| 2 (2) | 4 | DEVTYP | Device type. |
| 6 (6) | 6 | VOLSER | Volume serial number. |
| 12 (C) | 2 | FILESEQ | File sequence number. (This field is provided for compatibility with the OS/VS catalog, and is used for nonVSAM data sets that reside on tape volumes.) |
| 14 (D) | 1 | VOLFLG | Volume flags, as follows: |
| | 1... .... | | Prime, which indicates that this volume was allocated when the by the data set described by this record. |
| | ..1. .... | | Overflow, which indicates that this volume is being used by the data set, but this volume was not allocated when the data set was defined. |
| | ...x xxxx | | Reserved. |

## User-Catalog Catalog Record Format

The User-Catalog record describes a user catalog.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 1 | | Zeros. |
| 1 (1) | 3 | ENTIDNO | Control interval number of this record. |
| 4 (4) | 1 | RELIND | Release indicator:<br>0=Nonenhanced VSAM;<br>1=Enhanced VSAM. |
| 5 (5) | 6 | CRAVOL | CRA volume serial.[1] |
| 11 (B) | 3 | CRAIDNO | CRA control interval number.[1] |
| 14 (E) | 4 | CRADEVT | CRA device type.[1] |
| 18 (12) | 4 | CRACRETS | CRA creation time stamp. |
| 22 (16) | 4 | CRADITS | Data/index identifier timestamp. |
| 26 (1A) | 18 | | Zeros. |
| 44 (2C) | 1 | ENTYPE | Record type—"U." |
| 45 (2D) | 2 | | Record length. |
| 47 (2F) | 1 | | Number of variable-length fields that precede the pointer to an extension record. Always zero. |

## User-Catalog Catalog Record Format

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 48 (30) | 1 | | Length of the fixed-length fields in this record, excluding any fixed-length fields following displacement 92 (5C). This value is always equal to the displacement from the beginning of the record to the pointer to an extension record. |
| 49 (31) | 44 | ENTNAME | Name of the user catalog described by this record. |
| *The following 6-byte entry contains control information for the sets of fields that follow it.* | | | |
| 93 (5D) | 5 | | Pointer to horizontal extension record. If this record is not continued on an extension record, this field contains zeros. The user-catalog catalog record is usually not extended. |
| 98 (62) | 1 | | The number of set-of-fields pointers that follow.[2] |
| | | | **Note:** The first set-of-fields pointer contains sequence number = 0 and type code = 0. When bytes 0 to 2 are nonzero, this set-of-fields pointer points to the first (vertical) extension record that contains free space. The set-of-fields pointer is called the ASPT—available-space pointer. When bytes 0 to 2 are zero, the pointer is a dummy set-of-fields pointer—not available to be used to point to a set of fields. |
| | VL | | 5-byte pointers to sets of fields within the record. |

| Byte | Meaning |
|---|---|
| 0 | Reserved |
| 1-2 | Displacement of the set of fields from the beginning of all sets of fields in this record. |
| 3 | Bits 0 and 1 are set to zero. If bit 1 is on, the set of fields associated with this pointer has been deleted; the code in bits 2 through 7 of this byte and the sequence number in byte 4, however, are kept. Bits 2 through 7 contain a value of 3, which indicates that the set of fields pointed to contains volume information. |
| 4 | Sequence number of the set of fields pointed to. |

### User-Catalog Catalog Record Format

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| | VL | | 5-byte pointers to sets of fields contained in vertical Extension records.<br>**Byte** **Meaning**<br>0-2 Control interval number of the extension record that contains this set of fields.<br>3 Bits 0 and 1 are set to B'10'. Bits 2 through 7 contain a value of 3, which indicates that the set of fields pointed to contains volume information.<br>4 Sequence number of the set of fields pointed to. |

[1] Zeros if the catalog is not recoverable or if there is no associated CRA volume.

[2] Fields describing the volumes on which the user catalog resides are grouped into sets of fields. Pointers to each set of fields identify the type of information contained in the set of fields.

## *Volume Information Set of Fields Format*

Each volume that contains space allocated to the user catalog (and, therefore, is owned by the user catalog is described by a volume information set of fields. This set of fields is associated with a pointer that contains a type (group) code of 3.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 2 | | Control information.<br>**Byte** **Meaning**<br>0 Count of the number of variable-length fields in this set of fields.<br>1 Hexadecimal displacement to the first variable-length field from the beginning of this set of fields. |
| 2 (2) | 4 | DEVTYP | Device type. |
| 6 (6) | 6 | VOLSER | Volume serial number. |
| 12 (C) | 2 | FILESEQ | File sequence number. (This field is provided for compatibility with the OS/VS catalog.) |
| 14 (E) | 1 | VOLFLG | Volume flags, as follows: |
| | 1... .... | | Prime, which indicates that this volume was allocated when the catalog was defined. |
| | .1.. .... | | Candidate, which indicates that this volume is available for use by the catalog. |
| | ..1. .... | | Overflow, which indicates that this volume is being used by the catalog. but this volume was not allocated when the catalog was defined. |
| | ...x xxxx | | Reserved. |

## Volume Catalog Record Format

The Volume record describes VSAM data spaces, their extents, and the data sets that reside in VSAM data spaces.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 1 | | Zeros. |
| 1 (1) | 3 | ENTIDNO | Control interval number of this record. |
| 4 (4) | 1 | RELIND | Release indicator:<br>0=Nonenhanced VSAM;<br>1=Enhanced VSAM. |
| 5 (5) | 6 | CRAVOL | CRA volume serial.[1] |
| 11 (B) | 3 | CRAIDNO | CRA control interval number. |
| 14 (E) | 4 | CRADEVT | CRA device type.[1] |
| 18 (12) | 4 | CRACRETS | CRA creation time stamp. |
| 22 (16) | 22 | | Zeros. |
| 44 (2C) | 1 | ENTYPE | Record type—"V." |
| 45 (2D) | 2 | | Record length. |
| 47 (2F) | 1 | | Number of variable-length fields that precede the pointer to an extension record. Always zero. |
| 48 (30) | 1 | | Length of the fixed-length fields in this record, excluding any fixed-length fields following displacement 126 (7E). This value is always equal to the displacement from the beginning of the record to the pointer to an extension record. |
| 49 (31) | 44 | ENTNAME | Volume serial number, filled with binary zeros on the right, of the volume described by this record. |
| 93 (5D) | 8 | VOLSTMP | Volume time stamp, which indicates when the first VSAM data space was defined on this volume. |
| 101 (65) | 20 | VOLDVCHR | Device characteristics. |

| Byte | Meaning |
|---|---|
| 0-3 | Volume device type. |
| 4-7 | Maximum device blocksize. |
| 8-9 | Number of cylinders on this volume. |
| 10-11 | Number of tracks per cylinder on this volume. |
| 12-13 | Number of bytes per track on this volume. |
| 14 | Number of bytes required for gaps and check bits for each keyed block other than the last block on a track for this volume.[2] |
| 15 | Number of bytes required for gaps and check bits for the last keyed block on a track for this volume.[2] |
| 16 | Number of bytes to be subtracted for a block that is not keyed.[2] |

## Volume Catalog Record Format

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| | | | 17 Flags. Bits 0 through 6 are reserved. If bit 7 is set to 1, use tolerance factor on all blocks but the last to calculate the effective length of a block.[2] |
| | | | 18-19 Tolerance factor to be used in calculating the effective length of a block. |
| 121 (79) | 1 | VOLRFLG | Volume record flags. |
| 122 (7A) | 1 | SYSEXTDS | Number of extents per suballocation-request allowed by the OS/VS system. |
| 123 (7B) | 4 | | Reserved |
| *The following field names identify information that is not contained in the volume catalog record; the information is derived from fields in the volume catalog record.* | | | |
| | 1 | NODSPACE | Number of data spaces on the volume—a count of the Data Space Group sets of fields. |
| | 1 | NODSET | Number of data sets on the volume—a count of the Data Set Directory Entry sets of fields. *The following six-byte entry contains control information for the sets of fields that follow it.* |
| 127 (7F) | 5 | | Pointer to horizontal extension record. If this record is not continued on an extension record, this field contains zeros. |
| 131 (83) | 1 | | Number of set-of-fields pointers that follow.[3] |
| | | | **Note:** The first set-of-fields pointer contains sequence number = 0 and type code = 0. When bytes 0 to 2 are nonzero, this set-of-fields pointer points to the first (vertical) extension record that contains free space. The set-of-fields pointer is called the ASPT—available-space pointer. When bytes 0 to 2 are zero, the pointer is a dummy set-of-fields pointer—not available to be used to point to a set of fields. |
| 132 (84) | VL | | 6-byte pointers to sets of fields within the record. |
| | | | **Byte** **Meaning** |
| | | | 0 Reserved. |
| | | | 1-2 Displacement of the set of fields from the beginning of all sets of fields in this record. |
| | | | 3 Bits 0 and 1 are set to zero. If bit 1 is on, the set of fields associated with this pointer has been deleted; the code in bits 2 through 7 of this byte and the sequence number in bytes 4 and 5, however, are kept. Bits 2 through 7 contain a code,[4] describing the set of fields pointed to. |
| | | | 4-5 Sequence number of the set of fields pointed to. |

### Volume Catalog Record Format

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| | VL | | 6-byte pointers to sets of fields contained in vertical Extension records. |

| Byte | Meaning |
|---|---|
| 0-2 | Control interval number of the Extension record that contains this set of fields. |
| 3 | Bits 0 and 1 are set to B'10'. Bits 2 through 7 contain a code describing the set of fields pointed to. |
| 4-5 | Sequence number of the set of fields pointed to. |

[1] Zeros if the catalog is not recoverable or if there is no associated CRA entry.

[2] This value is used to calculate overhead bytes for keyed blocks to provide for compatibility with the DEVTYPE macro instruction. Blocks used by VSAM are, however, not keyed blocks.

[3] Fields describing (a) the volume's track-allocated/unallocated status, (b) each VSAM data space on the volume, and (c) each data set that resides in a VSAM data space are grouped into sets of fields. Pointers to each set of fields identify the type of information contained in each set of fields.

[4] If the pointer is associated with track status (space map) information, the code is 5; with data-space information, the code is 6; with a data set directory entry, the code is 8.

## *Space Map Set of Fields Format*

The tracks on a VSAM volume are allocated to a VSAM object, or are unallocated, as described by the Space Map set of fields. Each bit position describes one track as allocated (bit = 0) or unallocated (bit = 1). This set of fields is associated with a pointer that contains a type code of 5.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 2 | | Control Information: |

| Byte | Meaning |
|---|---|
| 0 | Count of the number of variable-length fields in this set of fields (X'01'). |
| 1 | Hexadecimal displacement to the variable-length field, from the beginning of the set of fields (X'02'). |

| | | | |
|---|---|---|---|
| 2 (2) | VL | BITMAP | Portion of the volume bit map (1 to 440 bytes describing the allocated or unallocated status of 1 to 3520 tracks). |

### *Data Space Group Set of Fields Format*

Each VSAM data space on the volume is described with a Data Space Group set of fields. This set of fields is associated with a pointer that contains a type code of 6.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 2 | | Control information.<br>**Byte** **Meaning**<br>0 Count of the number of variable-length fields in this set of fields (X'00').<br>1 Hexadecimal displacement to the first variable-length field from the beginning of this set of fields (X'55'). |
| 2 (2) | 8 | DSCBTS | Format-1 DSCB time stamp, which indicates when the DSCB was created. The time stamp is part of the name given to the Format-1 DSCB. |
| 10 (A) | 5 | DSCBPTR | CCHHR of the Format-1 DSCB. |
| 15 (F) | 1 | SPHDFLG | Data-space flags. |
| | 1... .... | | Unique data space, that is, this data space contains all or part of only one VSAM object. |
| | 0... .... | | Shared data space, that is, this data space contains all or part of two or more VSAM objects. |
| | .1.. .... | | Automatically built data space, that is, this data space was built during end-of-volume processing. |
| | .... 1... | | This data space was built when the user issued an Access Method Services DEFINE CATALOG command. |
| | ..xx .xxx | | Reserved. |
| 16 (10) | 1 | NODSPEXT | Number of extents in this data space. |
| 17 (11) | 1 | DSPSOPT | Data-space creation space options. |
| | 10.. .... | | Track request, which indicates that primary space allocation is specified in tracks. |
| | 11.. .... | | Cylinder request, which indicates that primary space allocation is specified in cylinders. |
| | ..xx xxxx | | Reserved |
| 20 (14) | 3 | DSPSSQ | Secondary space allocation quantity by which space is to be extended if required. This value is taken either from an Access Method Services DEFINE command or from the first data set on this volume that caused space to be used. |
| 23 (17) | 64 | SPEXTENT | Sixteen 4-byte extent descriptors in the form TTNN:<br>TT—starting track number of the extent (relative to the beginning of the volume).<br>NN—number of tracks in the extent. |

## Derived Data Space Information

The following field names identify information that is expected, but not contained in, the Data Space Group set of fields. The information is derived from fields in the volume catalog record.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| | 1 | SPHDFLG | Data space flags: |
| | ..1. .... | | A user catalog has extents within this data space—each CAXWA associated with the user contains the volume serial number of its catalog. |
| | ...1 .... | | A master catalog has extents within this data space—the master catalog's volume is identified by a Data Set Directory Entry set of fields that contains a control interval value of 002. |
| | 2 | NODSDSP | Number of data sets in the data space—this information is derived by searching each data set and index catalog record (pointed to by Data Set Directory Entry sets of fields and Cluster catalog records) for a volume information set of fields that contains the volume's serial number. Each set of fields so identified is searched to determine if the data set or index has been allocated space in one of the data space's extents. |
| *The following field names refer to information about an extent of the data space:* | | | |
| | 2 | TRKSUSED | Number of allocated tracks in the extent—the Space Map set of fields is scanned to determine the number of allocated tracks, based on the extent's starting track number and total number of tracks (contained in SPEXTENT). |
| | 4 | EXTSTART | Cylinder and track on which the extent begins—the extent's TT value (contained in SPEXTENT) is converted to a CCHH value. |
| | 2 | NOTRKEXT | Number of tracks in the extent—the extent's NN value (contained in SPEXTENT). |
| | 2 | SNSPHD | Sequence number of the set of fields that describes the extent's data space—the sequence number of the Data Space Group set of fields. |
| | VL | SPACEMAP | A variable-length space map that defines the allocated and unallocated space in the extent—the Space map set of fields is converted to the format of this variable-length field based on the extent's starting track number and total length (contained in SPEXTENT). |

## *Data Set Directory Entry Set of Fields Format*

Each data set that resides in a VSAM data space on the volume is described with a Data Set Directory Entry set of fields. This set of fields is associated with a pointer that contains a type code of 8.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 2 | | Control information.<br>**Byte — Meaning**<br>0 — Count of the number of variable-length fields in this set of fields (X'00').<br>1 — Hexadecimal displacement to the first variable-length field from the beginning of this set of fields (X'05'). |
| 2 (2) | 3 | DSIDNO | Control interval number of the Data or Index catalog record that describes this data set or index. |
| 5(5) | 4 | DSCRETS | Data/index identifier creation timestamp. Initialized by DEFINE and never altered. Consists of the high-order 4 bytes of the TOD clock value. |
| 9(9) | 3 | DSSUMTT | Sum of TT values converted from starting CCHHs of all extents of this data set on this volume. |

### Derived Data Set Information

The following field names identify information that is expected, but not contained in, the Data Set Directory Entry set of fields. The information is derived from fields in the volume catalog record.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| | 1 | MODSEXT | Number of data set extents on this volume. |
| | 1 | DSDIRFLG | Flags: |
| | 1... .... | | The Data or Index catalog record identifies the volume as a candidate volume—the Data or Index catalog record was searched and its volume information set of fields had zero extents. |
| | .xxx xxxx | | Reserved. |
| | VL | DSSPSN | A variable-length collection of 3-byte fields that identify each data space within which the data set has extents allocated to it—this information is obtained by converting each volume information set of fields' extent descriptor's (EXTENT) SS value (data space extent's sequence number) so that the resulting 3-byte field is:<br>**Byte — Meaning**<br>0-1 — Sequence number of the Data Space Group set of fields.<br>2 — Number of extents (groups of contiguous tracks) assigned to the data set or index from the data space (limits: 1—255). |

# Extension Catalog Record Format

The Extension record contains overflow information from another catalog record.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0(0) | 1 | | Zeros. |
| 1(1) | 3 | ENTIDNO | Control interval number of this record. |
| 4(4) | 1 | RELIND | Release indicator:<br>0 = Nonenhanced VSAM;<br>1 = Enhanced VSAM. |
| 5(5) | 6 | CRAVOL | CRA volume serial.[1] |
| 11(B) | 3 | CRAIDNO | CRA control interval number. |
| 14(E) | 4 | CRADEVT | CRA device tye.[1] |
| 18(12) | 4 | CRACRETS | CRA creation time stamp. |
| 22(16) | 22 | | Zeros. |
| 44 (2C) | 1 | ENTYPE | Record type—a "W" if this Extension record is an extension of a Volume record; an "E" if this Extension record is an extension of any other record.[2] |
| 45 (2D) | 2 | | Record length. |
| 47 (2F) | 1 | | Number of variable-length fields that precede the pointer to an Extension record. Always zero. |
| 48 (30) | 1 | | Length of the fixed-length fields in the header fields, excluding any fixed length fields following displacement 48 (30). This value is always equal to the displacement from the beginning of the record to the extension record's pointer. |
| *The following 6-byte entry contains control information for the sets of fields that follow it.* | | | |
| 49 (31) | 5 | | Pointer to horizontal Extension record. If this record is not continued on an Extension record, this field contains zeros. |
| 54 (36) | 1 | | The number of set-of-fields pointers that follow.[3]<br><br>**Note:** The first set-of-fields pointer contains sequence number = 0 and type code = 0. When bytes 0 to 2 are nonzero, this set-of-fields pointer points to the first (vertical) extension record that contains free space. The set-of-fields pointer is called the ASPT—available-space pointer. When bytes 0 to 2 are zero, the pointer is a dummy set-of-fields pointer—not available to be used to point to a set of fields. |
| 55 (37) | VL | | 5-byte pointers to sets of fields within the record.[4]<br><br>**Byte** **Meaning**<br>0 Reserved.<br>1-2 Displacement of the set of fields from the beginning of all sets of fields in this record. |

## Extension Catalog Record Format

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| | | | 3 Bits 0 and 1 are set to zero. If bit 1 is on, the set of fields associated with this pointer has been deleted; the code in bits 2 through 7 of this byte and the sequence number in byte 4, however, are kept. Bits 2 through 7 contain a code describing the set of fields pointed to.[4] |
| | | | 4 Sequence number of the set of fields pointed to. |
| | | | *If the record type is "W," the sequence number length is two bytes. If the record type is not "W," the sequence number length is one byte.* |
| | | VL | 5-byte pointers to sets of fields contained in vertical Extension records.[4] |
| | | | **Byte Meaning** |
| | | | 0-2 Control interval number of the Extension record that contains this set of fields. |
| | | | 3 Bits 0 and 1 are set to B'10'. Bits 2 through 7 contain a code that describes the set of fields pointed to.[4] |
| | | | 4 Sequence number of the set of fields pointed to by code. [4] |
| | | | *If the record type is "W," the sequence number length is two bytes. If the record type is not "W," the sequence number length is one byte.* |

[1] Zeros if catalog is not recoverable or if there is no associated CRA volume.

[2] The sets of fields that are contained in an Extension record depend upon the kind of catalog record that is extended. The format of the remainder of an Extension record is, therefore, variable. The sets of fields in an Extension record will, however, follow the same pattern as they would in the base record.

[3] Fields describing (a) the volumes on which a data set resides, and (b) the password information associated with a data set are grouped into sets of fields. Pointers to each set of fields identify the type of information contained in each set of fields.

[4] If the pointer is associated with volume information, the code is 3; or with password information, the code is 4. Sets of fields with codes 1 and 2 are never in an extension record.

## CRA Free Record Format

The CRA Free record indicates that the control interval in which it resides is free and points to the next control interval that is free because of deletion. Note that the Free CRA record is used only to mark a record that was used and has been deleted. The free space (control intervals) in the CRA that has never been assigned is not represented by Free CRA records. Control intervals 1, 4, 6, 7, and 8 in the CRA are marked as Free records; however, their Free control interval chain field is zero.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0(0) | 1 | | Reserved. |
| 1(1) | 3 | ENTIANO | Control interval number of this record: X'00000N' where N = 1, 4, 6, 7, or 8. |
| 4(4) | 40 | | Reserved. |
| 44(2C) | 1 | ENTYPE | Record type—"F." |
| 45(2D) | 3 | | Free control interval chain field |
| 48(30) | 457 | | Reserved. |

## CRA Data Record Format

The CRA Data record describes the CRA data component of the CRA cluster. The CRA Data record, which is record type "D", occupies control interval 9 in the CRA.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0(0) | 1 | | Zeros. |
| 1(1) | 3 | ENTIDNO | Control interval number of this record: X'000000.' |
| 4(4) | 1 | RELIND | Release indicator:<br>0 = Nonenhanced VSAM;<br>1 = Enhanced VSAM. |
| 5(5) | 39 | | Zeros. |
| 44(2C) | 1 | ENTYPE | Record type—"D." |
| 45(2D) | 2 | | Record length. |
| 47(2F) | 1 | | Number of variable-length fields that precede the pointer to an extension record. Always zero. |
| 48(30) | 1 | | Length of nonrepeating, fixed-length fields. Always X'8F.' |
| 49(31) | 44 | ENTNAME | Name of the catalog that owns this CRA volume. |
| 93(5D) | 8 | OWNERID | Initialized to all X'FF.' |
| 101(65) | 3 | DSETCRDT | Date CRA was created. In packed decimal form YDD. |
| 104(68) | 3 | DSETEXDT | Expiration date. Initialized to X'00000F.' |
| 107(6B) | 1 | ATTR1 | Data set attributes: X'00.' |
| 108(6C) | 1 | ATTR2 | Data set attributes: X'A0.' |
| 109(6D) | 1 | OPENIND | Open indicator: X'00.' |
| 110(6E) | 4 | BUFSIZE | Minimum buffer size: X'800.' |

**CRA Data Record Format**

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 114(72) | 3 | PRIMSPAC | Primary space. Initialized to number of tracks per cylinder. |
| 117(75) | 3 | SCONSPAC | Secondary space. Initialized to number of tracks per cylinder. |
| 120(78) | 1 | SPACOPTN | Space option: X'80.' |
| 121(79) | 4 | HURBADS | High-used RBA. |
| 125(7D) | 4 | HARBADS | High-allocated RBA. |
| 129(81) | 4 | LRECL | Logical record size: X'IF9.' |
| 133(85) | 2 | USERINFO | DOS user information: X'00.' |
| 135(87) | 8 | EXCPEXIT | Exception exit: initialized to all X'FF.' |
| *The following entries contain control information for repeating fields:* | | | |
| 143(8F) | 5 | | Pointer to an extension record. Always zeros. |
| 148(94) | 1 | | The number of set-of-field pointers that follow. |
| 149(95) | VL | | Five-byte pointers to sets of fields. |

## *AMDSB (Access Method Data Statistics Block) Set of Fields Format*

The AMDSB set of fields contains a copy of the AMDSB control block that is updated each time the CRA is closed. This set of fields is associated with a pointer that contains a type (group) code of 1.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0(0) | 2 | | Control information. |
| | | | **Byte** — **Meaning** |
| | | | 0 — Count of number of variable-length fields in this set of fields. |
| | | | 1 — Displacement (X'62') to the first variable-length field from the beginning of this set of fields. |
| 2(2) | 96 | AMDSBCAT | Copy of the AMDSB. See "Data Areas" for detailed information about the AMDSB. |

## *Association (Cluster) Set of Fields Format*

There is one association in this entry, and it has a group occurrence sequence number of 1.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0(0) | 2 | | Control information: X'0006.' |
| 2(2) | 1 | TYPE | Record type—'C.' |
| 3(3) | 3 | NAME | Control interval number: X'000002.' |

### *Volume Information Set of Fields*

There is one volume information set of fields in this entry. Its group occurrence sequence number is 1, and it resides in control interval number 5.

## CRA Cluster Record Format

This record describes the CRA cluster. It is record type "C," and it occupies control interval 2 in the CRA.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0(0) | 1 | | Zeros. |
| 1(1) | 3 | ENTIDND | Control interval number of this record: X'000002.' |
| 4(4) | 1 | RELIND | Release indicator:<br>0 = Nonenhanced VSAM;<br>1 = Enhanced VSAM. |
| 5(5) | 39 | | Zeros. |
| 44(2C) | 1 | ENTYPE | Record type—'C.' |
| 45(2D) | 2 | | Record length. |
| 47(2F) | 1 | | Number of variable-length fields that precede the pointer to an extension record. Always zero. |
| 48(30) | 1 | | Length of nonrepeating fixed-length fields: X'6C.' |
| 49(31) | 44 | ENTNAME | Name of the catalog that owns this CRA volume. |
| 93(5D) | 8 | OWNERID | Initialized to all X'FF.' |
| 101(65) | 3 | DSETCRDT | Date CRA was created. In packed-decimal form YDD. |
| 104(68) | 3 | DSETEXDT | Expiration date. Initialized to X'00000F.' |
| 107(6B) | 1 | CATTR | Cluster attributes: X'00.' |
| *The following entries contain control information for repeating fields.* | | | |
| 108(6C) | 5 | | Pointer to an extension record. Always X'00.' |
| 113(71) | 1 | | The number of set of field pointers that follow. |
| 114(72) | VL | | Five-byte pointers to sets of fields. |

### *Association (Data) Set of Fields Format*

There is one association in this entry, and its group occurrence sequence number is 1.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0(0) | 2 | | Control information: X'0006.' |
| 2(2) | 1 | TYPE | Record type –"D." |
| 3(3) | 3 | NAME | Control interval number: X'000000.' |

## CRA Catalog Control Record Format

The catalog control record is used to manage CRA control interval allocation. It is record type "L," and it occupies control interval 3 in the CRA.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0(0) | 1 | | Zeros. |
| 1(1) | 3 | ENTIDNO | Control interval number of this record: X'000003.' |
| 4(4) | 1 | RELIND | Release indicator:<br>0 = Nonenhanced VSAM.<br>1 = Enhanced VSAM. |
| 5(5) | 39 | | Zeros. |
| 44(2C) | 1 | ENTYPE | Record type—"L." |
| 45(2D) | 3 | | Number of the highest control interval within the current extents. |
| 48(30) | 3 | | Number of the next free control interval that has not been previously assigned. |
| 51(33) | 3 | | Number of deleted control intervals. |
| 54(36) | 3 | | First deleted control interval in a chain of control intervals that are free because of deletion. |
| 57(39) | 4 | | Reserved. |
| 61(3D) | 4 | | CRA data high-used RBA. |
| 65(41) | 4 | | CRA data high-allocated RBA. |
| 69(45) | 436 | | Reserved. |

## CRA Data Extension Record Format

The Data Extension record is the extension of the CRA Data Record in control interval 0 of the CRA. The Data Extension record is record type "E," and it occupies control interval 5 in the CRA.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0(0) | 1 | | Zeros. |
| 1(1) | 3 | ENTIDNO | Control interval number of this record: X'000005.' |
| 4(4) | 1 | RELIND | Release indicator:<br>0 = Nonenhanced VSAM;<br>1 = Enhanced VSAM. |
| 5(5) | 39 | | Zeros. |
| 44(2C) | 1 | ENTYPE | Record Type—"E." |
| 45(2D) | 2 | | Record length. |
| 47(2F) | 1 | | Number of variable length field that precede the pointer to an extension record. Always zero. |
| 48(30) | 1 | | Length of non-repeating fixed length fields: X'6C' |
| *The following entries contain control information for repeating fields:* | | | |
| 49(31) | 5 | | Pointer to an extension record. Always zeros. |

**CRA Data Extension Record Format**

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 54(36) | 1 | | The number of sets-of-fields pointers that follows. |
| 55(37) | VL | | 5-byte pointers to sets of fields within the record. |

## *Volume Information Set of Fields Format*

There is one volume information set of fields in this entry, and its group occurrence sequence number is 1.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0(0) | 2 | | Control information: X'0327.' |
| 2(2) | 4 | DEVTYP | Device type. |
| 6(6) | 6 | VOLSER | Volume serial number. |
| 12(C) | 2 | FILESEQ | File sequence number. |
| 14(E) | 1 | VOLFLG | Volume flags: X'80.' |
| 15(F) | 1 | NOEXTNT | Number of extents. |
| 16(10) | 4 | HKRBA | High-key RBA. |
| 20(14) | 4 | HURBA | High-used RBA. |
| 24(18) | 4 | HARBA | High-allocated RBA. |
| 28(1C) | 4 | PHYBLKSZ | Block size: X'200.' |
| 32(20) | 2 | NOBLKTRK | Number of blocks per track. |
| 34(22) | 2 | NOTRKAU | Number of tracks per allocation unit. Initialized to number of tracks per cylinder. |
| 36(24) | 1 | ITYPEXT | Type-of-extent indicator. |
| 37(25) | 2 | DSDIRSN | Data set directory sequence number. |
| 39(27) | 2 | LOKEYV | Low key value on volume. |
| 41(29) | 2 | HIKEYV | High key value on volume. |
| 43(2B) | 2 | EXTENT | Length of extent information |
| 45(2D) | VL | | 20-byte extent descriptors. |

# Field-Name Dictionary

The field-name dictionary is an internal data area that provides a map between field names and fields within catalog records, as well as information that is not within catalog records. The dictionary also allows the dictionary user to specify values (for example, the number of sets of fields to be processed) by associating them with a dictionary name. A field name is specified in a CTGFL (field parameter list); for a description of the CTGFL, see "Data Areas." The catalog-management modules reference the field-name dictionary for the location, length, and type of fields.

The field-name dictionary is a series of 8-byte entries. In addition, there is an index of combination field names. Each combination field name allows catalog management to locate more than one field at a time.

The field-name dictionary is located in module IGG0CLAY.

| Offset | Bytes and Bit Pattern | Description |
|---|---|---|
| 0 | 4 | Shortened field name: the first, second, fifth, and sixth characters of the eight-character field name. |
| 4 | 1 | Flags that describe the field: |
| | 000. .... | The field is fixed-length and appears in the header portion of a record (before the Extension record pointer). |
| | 001. .... | A combination field name.[1] |
| | 010. .... | The field is fixed-length and is part of a set of fields that follows the Extension record pointer. |
| | 100. .... | The field is variable-length and appears in the header portion of a record (before the Extension record pointer). |
| | 110. .... | The field is variable-length and is part of a set of fields that follows the Extension record pointer. |
| | 011. .... | Special field[2] |
| | 111. .... | Special field[2] |
| | ...0 .... | Not a flag field, which means that a CLC (compare logical character) instruction can be used to test this field. |
| | ...1 .... | Flag field, which means that a TM (test under mask) instruction can be used to test this field. |
| | .... 1... | A fixed-length field within a variable-length field in a set of fields. |
| | .... 0... | Not a fixed-length field within a variable-length field in a set of fields. |
| | .... .1.. | CRA updates are to be suppressed. |
| | .... .0.. | CRA updates are not suppressed. |
| | .... ..1. | This field must be retrieved from the upgrade set. |
| | .... ..0. | This field has no special retrieval characteristics. |
| | .... ...x | Reserved |
| 5 | 1 | Bytes that identify the location of the field: |

| Type of Field Name: | Contents of This Byte: |
|---|---|
| Fixed-length: | Displacement in bytes from the |
| In the header: | Beginning of the record. |
| In a set of fields: | Beginning of the set of fields. |
| In a group of fixed-length fields within a variable-length field: | Length of the group of fixed-length fields. |
| Variable-length: | Zero. |

**Field-Name Dictionary**

| Offset | Bytes and Bit Pattern | Description |
|---|---|---|
| | | Combination: Index value in the combination-name index. |
| 6 | 1 | Bytes that identify the location of the field (continued): |
| | | **Type of Field Name:** / **Contents of This Byte:** |
| | | Fixed-length: Length of the field (in bytes). |
| | | Variable-length: Sequence number of the field. |
| | | Combination: Number of fields identified by the combination name. |
| 7 | 1 | A code that indicates which group of data (the kind of catalog record and the set of fields) this field is in. |

| Type Code: | Description: |
|---|---|
| 0 | Header field |
| 1 | AMDSB |
| 2 | Association |
| 3 | Volume information |
| 4 | Password |
| 5 | Space map [3] |
| 6 | Data space group [3] |
| 7 | Reserved |
| 8 | Data set directory[3] |

[1] "Combination field name" indicates that the name supplied is a name that allows catalog management to locate a group of related fields.

[2] The field-name dictionary permits catalog management to locate information that is not contained in catalog records.

[3] This set of fields is contained only in a Volume catalog record.

Bytes 4 through 7 of the field-name dictionary record describe the field. When a caller specifies catalog information with an CTGFL, dictionary information is moved into the CTGFL.

To clarify the use of the dictionary as a means of gaining access to catalog information, refer to the examples that follow.

## *Combination Field Names*

A combination field name identifies a group of related fields. When a catalog management user requires information from many catalog record fields (for example, all fields in a set of fields), the user builds a CTGFL that contains a combination field name. The combination field name in the CTGFL identifies an entry in the field name dictionary (in module IGG0CLAY). The entry identifies the field name as a combination field name, specifies the number of fields contained in the combination, and points to the starting point for the combination in the combination field name index. The combination field name index contains a group of 1-byte entries. Each entry points to an entry in the field name dictionary, as shown in Figure 60, Resolution of a Combination Field Name. The entry in the field name dictionary describes one of the fields identified by the combination.

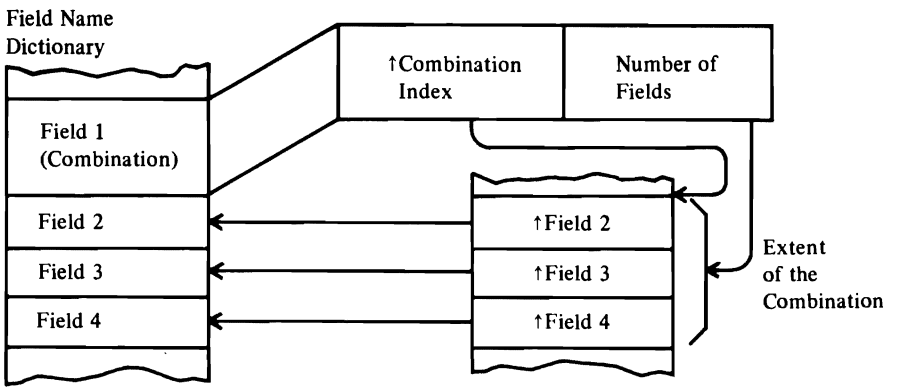



Figure 60. Resolution of a Combination Field Name

## *Field Name Dictionary Entries*

This section lists the field name dictionary entries in alphabetic order. IGG0CLAY lists the field name dictionary entries in alphabetic order, too, but that list is ordered by *abbreviated* field name. The field's name, length, location, and description are listed here.

The length is a number of bytes. If the field is variable-length, the code 'VL' is used. If the name is a combination name, the length might be indeterminate. The field's length (Len) is:

- 'n'—a number of bytes.
- 'VL'—indeterminate, because the field is a variable-length field.
- '—'—indeterminate, because the combination-name group of fields includes one or more variable-length fields.

The field's location is coded as follows:

| Code | Location description |
|---|---|
| AMDSB | The AMDSB set of fields |
| Assoc | The association set of fields |
| Combin | A combination field name |
| Header | The catalog record's header fields. Note: some header fields appear in all catalog records; other header fields appear only in one (or more) type of catalog record. |
| Password | The password set of fields |
| Special | A piece of information that is derived from information in catalog record fields or catalog control blocks, but is not stored in a catalog record |
| Vol Info | The volume information set of fields |
| Vol–Dir | The data set directory entry set of fields in the volume catalog record |
| Vol–DSG | The data space group set of fields in the volume catalog record |
| Vol–SM | The space map set of fields in the volume catalog record |

**Notes:**

- The list of field names following each combination name in this section is ordered alphanumerically. See module listing IGG0CLAY for the actual order of the combination's field names.

- Each variable-length field contains two bytes of control information followed by a number of data bytes. The 2-byte control information specifies the total length of the field (the number of data bytes).

| Field Name | Len | Location | Description |
|---|---|---|---|
| AMDCIREC | 8 | AMDSB | AMDSB control-interval size and maximum logical record size (Show Catalog support) |
| AMDKEY | 4 | AMDSB | AMDSB relative key position and key length (Show Catalog support) |
| AMDSBCAT | 96 | AMDSB | Copy of the AMDSB control block |
| AMDSBSC | 12 | Combin | AMDSB fields for Show Catalog<br>Includes fields: AMDCIREC, AMOKEY |
| AMDSB1 | 6 | AMDSB | Part of AMDSB |
| AMDSB2 | 10 | AMDSB | Part of AMDSB |
| AMDSB3 | 68 | AMDSB | Part of AMDSB |
| ASSOCSC | 8 | Combin | Twin associations for Show Catalog support |
| ATTR1 | 1 | Header | Data set attributes |
| ATTR2 | 1 | Header | Data set attributes |
| BITMAP | VL | Vol–SM | Volume space map showing the allocated and unallocated tracks on a direct-access volume |
| BUFSIZE | 4 | Header | Minimum buffer size |
| CATACB | 4 | Special | Address of the catalog's ACB control block |
| CATTR | 1 | Header | Cluster attributes |
| CATVOL | 15 | Combin | Volume information set of fields for a nonVSAM data set's catalog record<br>Includes fields: DEVTYP, FILESER, RELREPNO, VOLFLG, VOLSER |
| CNTREPNO | 2 | Special | Maximum number of RELREPNOs to be processed |
| CRACRETS | 4 | Header | CRA creation timestamp |
| CRADEVT | 4 | Header | CRA device type |
| CRADIRCT | 12 | Combin | Data set directory fields for Catalog Recovery<br>Includes fields: DSCRETS, DSIDNO, DSSUMTT, RELREPNO |
| CRADITS | 4 | Header | Data/index identifier creation timestamp |
| CRAIDNO | 3 | Header | CRA control interval number |
| CRAVOL | 6 | Header | CRA volume serial |
| DATASPAC | 85 | Combin | Data space group set of fields<br>Includes fields: DSCBPTR, DSCBTS, DSPSOPT, DSPSSQ, NODSPEXT, SPEXTENT, SPHDFLG, RELREPNO |
| DEVTYP | 4 | Vol Info | Device type |
| DEXTENTS | 68 | Combin | All fields in the data space group set of fields required by Suballocate processing<br>Includes fields: NODSPEXT, SPEXTENT, SPHDFLG, RELREPNO |
| DIRECTRY | 5 | Combin | Data set directory entry set of fields<br>Includes fields: DSIDNO, RELREPNO |

## Field Name Dictionary Entries

| Field Name | Len | Location | Description |
|---|---|---|---|
| DSATRO | 3 | Combin | All data set attributes flags fields |
| | | | Includes fields: ATTR1, ATTR2, OPENIND |
| DSATTR | 2 | Combin | All data set attributes flags fields, except the open indicator |
| | | | Includes fields: ATTR1, ATTR2, OPENIND |
| DSCBPTR | 5 | Vol–DSG | TTR of the DSCB that describes the data space in the volume's VTOC |
| DSCBTS | 8 | Vol–DSG | Data space timestamp |
| DSCBTTR | 3 | Vol Info | TTR of the format 1 (identifier) DSCB that describes the space allocated to a nonVSAM data set |
| DSCRETS | 4 | Vol-Dir | Data/index identifier timestamp |
| DSDIRECT | — | Combin | Data set directory entry set of fields, including some of the related derived fields |
| | | | Length: 7 + l'DSSPSN |
| | | | Includes fields: DSDIRFLG, DSIDNO, DSSPSN, NODSEXT, RELREPNO |
| DSDIRFLG | 1 | Special | Data set directory flags (derived) |
| DSDIRSN | 2 | Vol Info | Data set directory sequence number in the volume informationset of field's extent descriptor |
| DSETCRDT | 3 | Header | Data set creation date |
| DSETEXDT | 3 | Header | Data set expiration date |
| DSIDNO | 3 | Vol–Dir | Control interval number of the data set directory entry's object's catalog record |
| DSPDSCRP | 13 | Combin | Space descriptor set of fields (a group of derived fields) |
| | | | Includes fields: EXTSTART, NOTRKEXT, RELREPNO, SNSPHD, SPACEMAP, TRKSUSED |
| DSPSOPT | 1 | Vol–DSG | Space options for a data space |
| DSPSSQ | 3 | Vol–DSG | Secondary data space quantity |
| DSSPSN | VL | Special | Data space sequence numbers for the data set directory entry set of fields (derived) |
| DSTYPNAM | 4 | Combin | Catalog record type and dsname |
| | | | Includes fields: ENTIDNO, ENTYPE |
| DSSUMTT | 3 | Vol-Dir | Sum of starting tracks of all extents of data set on this volume |
| ENTASSOC | 8 | Combin | Association set of fields |
| | | | Includes fields: NAME, RELREPNO, TYPE |
| ENTIDNO | 3 | Header | Control interval number of the catalog record |
| ENTNAME | 44 | Header | Dsname in the catalog record |
| ENTUPGD | 22 | Combin | Combination for associations in the upgrade set |
| | | | Includes fields: NAME, NAME2, RELREPNO, TYPE, TYPE2 |
| ENTVOL | 37 | Combin | Volume information set of fields |

## Field Name Dictionary Entries

| Field Name | Len | Location | Description |
|---|---|---|---|
| | | | Includes fields: DEVTYP, FILESEQ, HARBA, HKRBA, HURBA, ITYPEXT, NOBLKTRK, NOEXTNT, NOTRKAU, PHYBLKSZ, VOLFLG, VOLSER, RELREPNO, DSDIRSN, EXTENT, HIKEYV, LOKEYV |
| ENTYPE | 1 | Header | Catalog record type identifier |
| EXCPEXIT | 8 | Header | Exception exit |
| EXTENT | VL | Vol Info | Extent descriptors |
| EXTSTART | 4 | Special | Starting point in the list of data space extents in the data space group set of fields (derived) |
| EXTVOL | — | Combin | Volume information set of field's fields required for VSAM End of Volume processing |
| | | | Length: 7 + l'EXTENT |
| | | | Includes fields: DEVTYP, EXTENT, ITYPEXT, RELPERNO |
| FILESEQ | 2 | Vol Info | File sequence number for a nonVSAM data set |
| | | | Includes fields: NAME, TYPE |
| GENDSP | 44 | Special | Generated data space dsname |
| HARBA | 4 | Vol Info | High-allocated RBA on the volume for the data set |
| HARBADS | 4 | Header | High-allocated RBA for the data set (not always the same as HARBA if the data set resides on several volumes) |
| HIKEYV | 2 | Vol Info | Key-sequenced data set's high key value on a volume or, if the data set is divided into key ranges, the key range's high key value |
| HKRBA | 4 | Vol Info | RBA of the record containing the high key of a key-sequenced data set on a volume or, if the data set is divided into key ranges, the key range's high key value |
| HKURBA | 8 | Combin | Data set's high-key and high-used RBAs |
| | | | Includes fields: HKRBA, HURBA |
| HURBA | 4 | Vol Info | High-used RBA on the volume for the data set |
| HURBADS | 4 | Header | High-used RBA for the data set (not always the same as HURBA if the data set resides on several volumes) |
| ITYPEXT | 1 | Vol Info | Type of extent indicator |
| LOKEYV | VL | Vol Info | Key-sequenced data set's low-key value on a volume or, if the data set is divided into key ranges, in the key range |
| LRECL | 4 | Header | Average logical record size |
| MAPSPACE | — | Combin | Volume catalog record's space map set of fields |
| | | | Length: 2 + l'BITMAP |
| | | | Includes fields: BITMAP, RELREPNO |
| NAME | 3 | Assoc | Control interval number |
| NAME2 | 3 | Assoc. | Control interval number of index in twin association of upgrade set |
| NAMEDS | 4 | Combin | Association set of fields |

## Field Name Dictionary Entries

| Field Name | Len | Location | Description |
|---|---|---|---|
| | | | Includes fields: NAME, TYPE |
| NOBLKTRK | 2 | Vol Info | Number of blocks per track |
| NOBYTTRK | 4 | Vol Info | Number of bytes per track |
| NODSDSP | 2 | Special | Number of data sets in a data space (derived) |
| NODSET | 2 | Special | Number of data sets on a volume (derived) |
| NODSEXT | 1 | Special | Number of data set directory extents (derived) |
| NODSPACE | 2 | Header | Number of data spaces on a volume (derived) |
| NODSPEXT | 1 | Vol–DSG | Number of data space extents |
| NOEXTNT | 1 | Vol Info | Number of volume information set of field's extents |
| NONVOL | 16 | Combin | NonVSAM data set's volume information set of fields |
| | | | Includes fields: DEVTYP, DSCBTTR, FILESEQ, RELREPNO, VOLFLG, VOLSER |
| NOTRKAU | 2 | Vol Info | Number of tracks per allocation unit |
| NOTRKEXT | 2 | Special | Number of tracks in a data space's extent (derived) |
| OPENCNT | 1 | Header | Open count |
| OPENC2 | 61 | Combin | Catalog fields required by OPEN |
| | | | Includes fields: BUFSIZE, ENTNAME, EXCPEXIT, HURBADS, SPACOPTN |
| OPENIND | 1 | Header | Open indicator |
| OPNCALL1 | 53 | Combin | All header fields required by VSAM Open processing |
| | | | Includes fields: BUFSIZE, ENTNAME, HURBADS, SPACOPTN |
| OWNERID | 8 | Header | Owner identification number |
| PASSATMP | 2 | Password | Number of attempts the operator has to supply the correct password |
| PASSPRMT | 8 | Password | Data set's prompting name (codename) for security verification |
| PASSWALL | — | Combin | Password set of fields |
| | | | Length: 50 + l'USERAREC |
| | | | Includes fields: PASSATMP, PASSPRMT, PASSWORD, USERAREC, USVRMDUL |
| PASSWORD | 32 | Password | All (4) 8-byte passwords |
| PHYBLKSZ | 4 | Vol Info | Physical blocksize |
| PRIMSPAC | 3 | Header | Primary space allocation amount |
| RELCRA | 14 | Combin | Release indicator and CRA header fields |
| | | | Includes fields: CRADEVT, CRAIDNO, CRAVOL, RELIND |
| RELIND | 1 | Header | Release indicator |
| RELREPNO | 2 | Special | Relative repetition number |
| REPNO | 2 | Special | Highest nondeleted sequence number |
| RGATTR | 1 | Header | Path/alternate index indicator |

## Field Name Dictionary Entries

| Field Name | Len | Location | Description |
|---|---|---|---|
| SCONSPAC | 3 | Header | Secondary space allocation amount requirement |
| SLOCVOL | 19 | Combin | Fields required by Superlocate |
| | | | Includes fields: DEVTYP, DSCBTTR, FILESEQ, ITYPEXT, RELREPNO, VOLFLG, VOLSER |
| SNSPHD | 2 | Special | Data space sequence number for a data space (derived) |
| SPACEHDR | 23 | Combin | Data space group set of fields |
| | | | Includes fields: DSCBPTR, DSCBTS, DSPSOPT, DSPSSQ, NODSDSP, NODSPEXT, RELREPNO, SPHDFLG |
| SPACEMAP | 1 | Special | Data space run length codes (derived) |
| SPACOPTN | 1 | Header | Space options |
| SPACPARM | 7 | Combin | Space allocation quantities and options |
| | | | Includes fields: PRIMSPAC, SPACOPTN, SCONSPAC |
| SPEXTENT | 64 | Vol–DSG | Data space extent descriptors |
| SPHDFLG | 1 | Vol–DSG | Data space flag (partially derived) |
| SYSEXTDS | 1 | Header | Number of extents allowed per suballocation—in the volume catalog record |
| TOENTVOL | 43 | Combin | Track overflow entire volume |
| | | | Includes fields: DEVTYP, DSDIRSN, EXTENT, FILESEQ, HARBA, HIKEYV, HKRBA, HURBA, ITYPEXT, LOKEY, NOBLKTRK, NOBYTAU, NOBYTTRK, NOEXTNT, NOTRKAU, PHYBLKSZ, RELREPNO, VOLFLG, VOLSER |
| TRBAEXT | 3 | Vol Info | Test RBA for EOV mount by RBA |
| TRKSUSED | 2 | Special | Tracks used in the data space (derived) |
| TYPE | 1 | Assoc | Association entry type |
| TYPE2 | 1 | Assoc | Entry type for second name of a twin association in the upgrade set |
| UPDVOL | — | Combin | Volume information set of fields for UPDATE–Extend processing |
| | | | Length: 43 + l'HIKEYV + l'LOKEYV |
| | | | Includes fields: DEVTYP, DSDIRSN, FILESEQ, HARBA, HIKEYV, HKRBA, HURBA, ITYPEXT, LOKEYV, NOBLKTRK, NOEXTNT, NOTRKAU, PHYBLKSZ, RELREPNO, VOLFLG, VOLSER |
| UPGRADE | 8 | Combin | Fields for a twin association in the upgrade set |
| | | | Includes fields: TYPE, NAME, TYPE2, NAME2 |
| USERAREC | VL | Password | Additional security verification data |
| USVRMDUL | 8 | Password | USVR (user security verification routine) module name |
| VOLDEV | 12 | Combin | Volume information set of fields required by VSAM Open processing |

**Field Name Dictionary Entries**

| Field Name | Len | Location | Description |
|---|---|---|---|
| | | | Includes fields: DEVTYP, NOBLKTRK, NOTRKAU, PHYBLKSZ |
| VOLDVCHR | 20 | Header | Device characteristics—in the volume catalog record |
| VOLEXT | — | Combin | Volume information set of fields |
| | | | Length: 10 + l'EXTENT |
| | | | Includes fields: DSDIRSN, EXTENT, RELREPNO, VOLSER |
| VOLFLG | 1 | Vol Info | Volume information flags |
| VOLPHY | — | Combin | Volume information set of fields required by VSAM Open processing |
| | | | Length: 23 + l'EXTENT + l'HIKEYV + l'LOKEYV |
| | | | Includes fields: EXTENT, HARBA, HIKEYV, HKRBA, HURBA, ITYPEXT, LOKEYV, NOEXTNT, RELREPNO, VOLFLG, VOLSER |
| VOLRFLG | 1 | Header | Volume catalog record flags |
| VOLSER | 6 | Vol Info | Volume serial number |
| VOLTSTMP | 8 | Header | Volume catalog record timestamp |

### *Dictionary Example 1*

The DSETCRDT (data set creation date) field appears in the dictionary, as follows:

```
DSETCRDT,0,101,3,0
```

The first 0 is the fourth byte value of the record; it indicates that DSETCRDT is (a) a fixed-length field, (b) not part of a set of fields, and (c) not a flag field.

The 101 (decimal) is the fifth-byte value of the record; it indicates, when converted to hexadecimal, that DSETCRDT is at displacement X'65' from the beginning of the record in which it appears.

The 3 is the sixth-byte value of the record; it indicates that DSETCRDT is three bytes long.

The last 0 is the seventh-byte value of the record; it is zero because DSETCRDT is not part of a set of fields and, therefore, is not associated with a set of fields code.

### *Dictionary Example 2*

The DSPSOPT (data-space-creation space options) field appears in the dictionary, as follows:

```
DSPSOPT,80,19,1,6
```

The 80 (X'50') is the fourth-byte value of the record; it indicates, when converted to binary, that DSPSOPT is (a) a fixed-length field that is part of a set of fields, (b) a flag field, and (c) not a repeating field within a variable-length field.

The 19 is the fifth-byte value of the record; it indicates, when converted to hexadecimal, that DSPSOPT is at displacement 13 from the beginning of the set of fields to which it belongs.

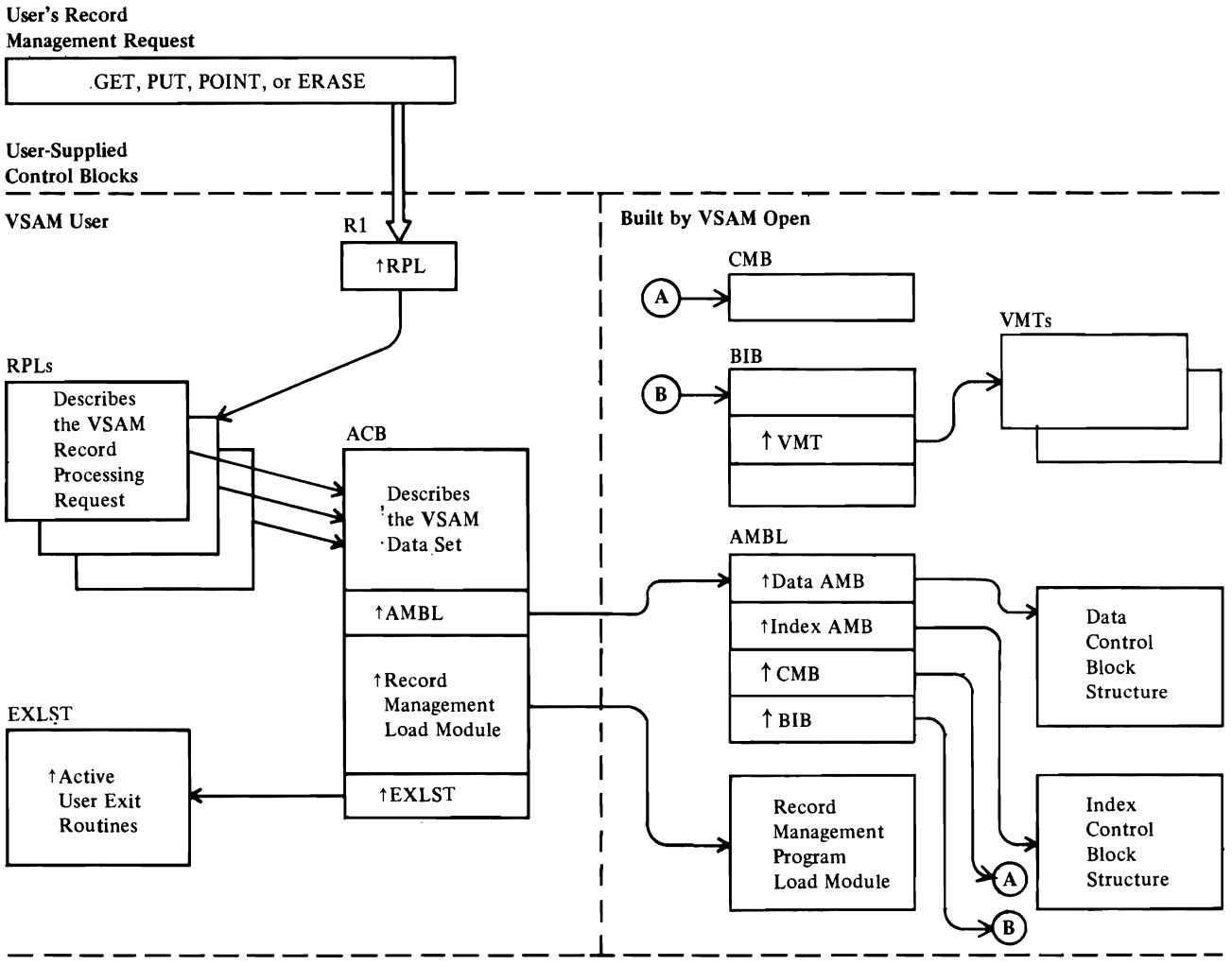
The 1 is the sixth-byte value of the record; it indicates that DSPSOPT is 1 byte long.

The 6 is the seventh-byte value of the record; it indicates that DSPSOPT is part of a set of fields associated with a code of 6, which means that it is part of a set of fields that contains VSAM data-space information.

## Control Block Interrelationships

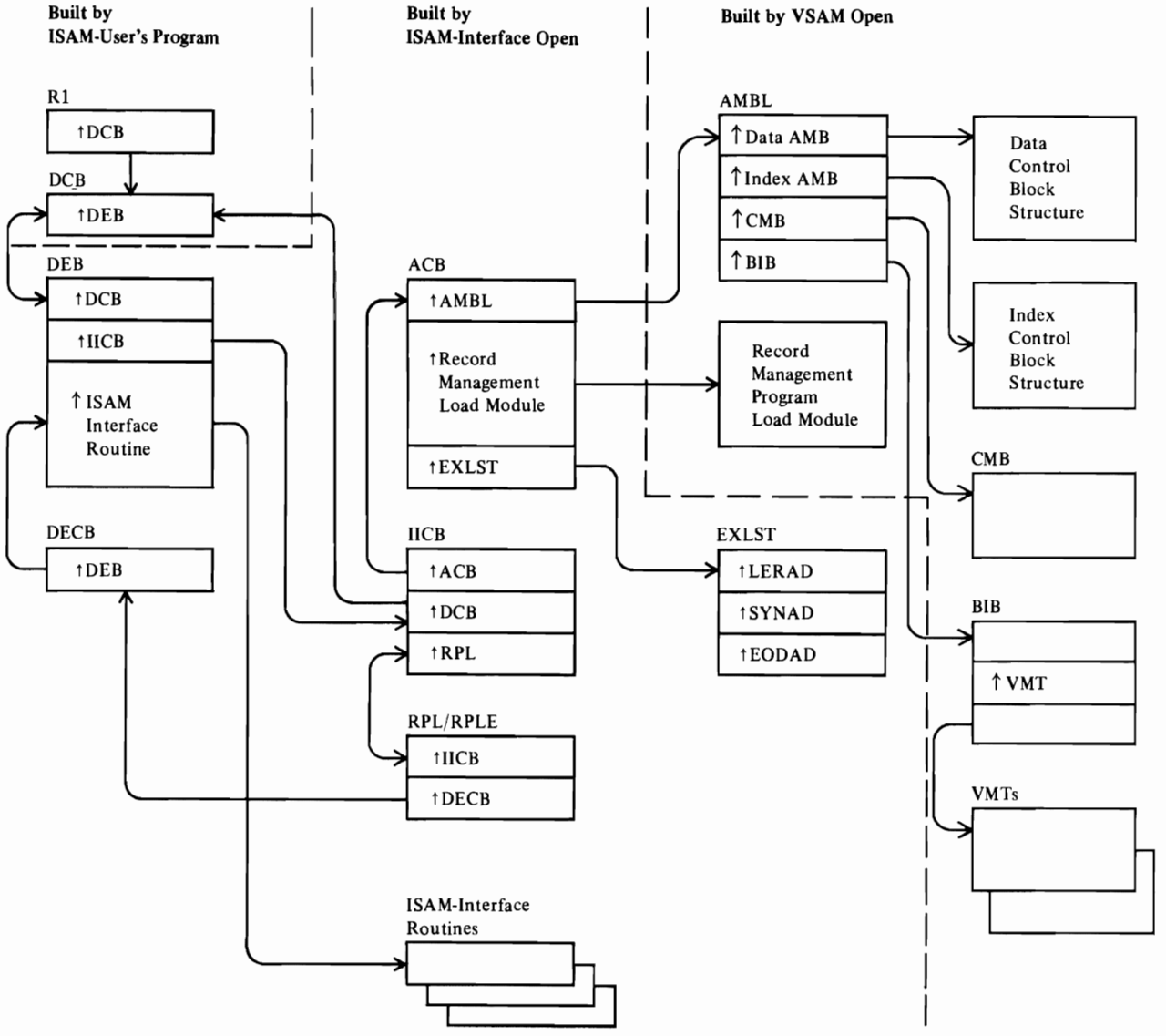
Figures 61 and 62 show the VSAM control blocks built when a key-sequenced data set is opened.

The role of the BIB and CMB in virtual-storage management is described in "Virtual-Storage Management" in "Diagnostic Aids."




**Note:** The data control block structure is shown in Figure 66. The index control block structure is shown in Figure 68.

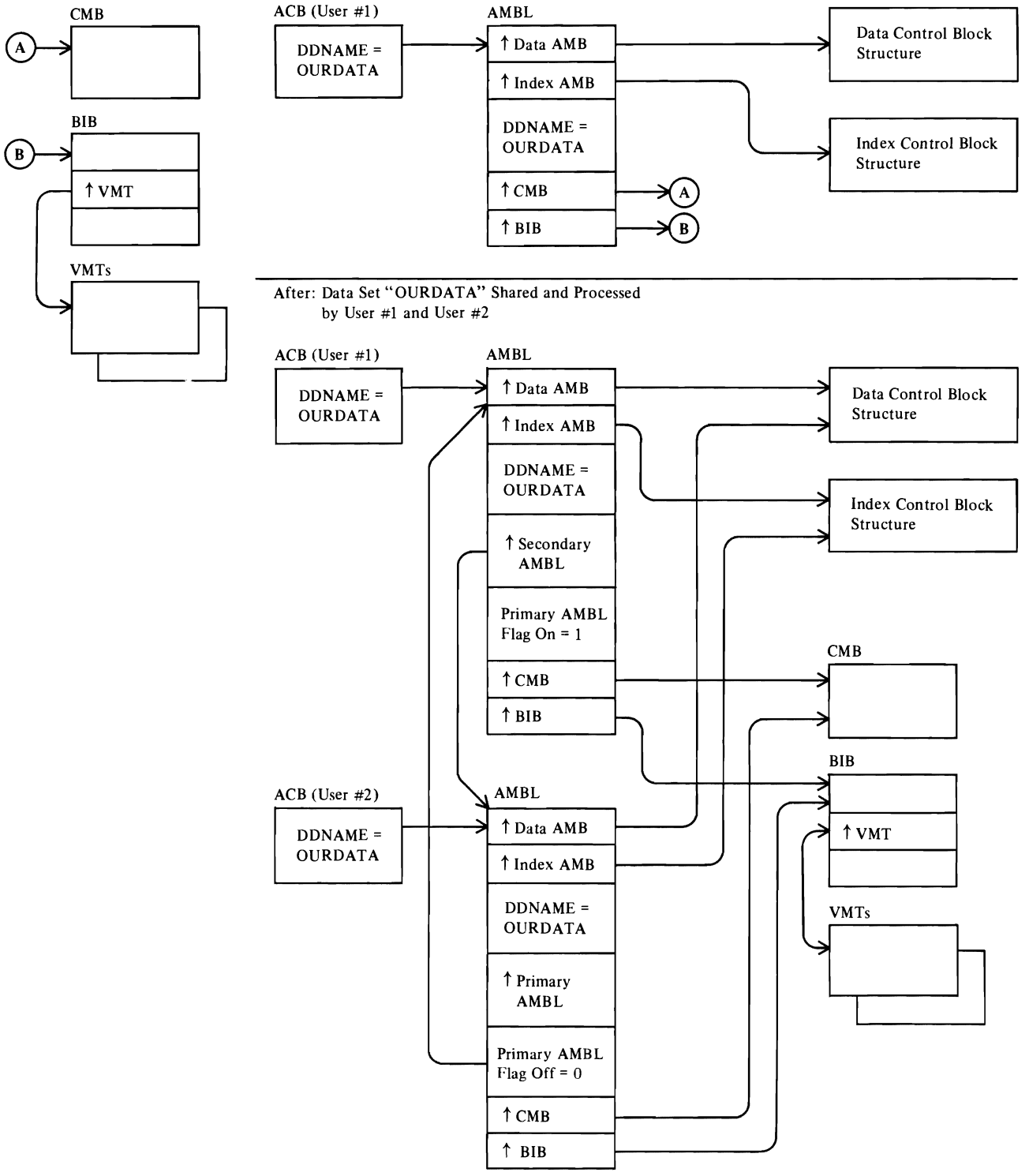
Figure 61. VSAM Control Block Structure for a Key-Sequenced Data Set (VSAM User)




**Note:** The data control block structure is shown in Figure 66.
The index control block structure is shown in Figure 68.

Figure 62. VSAM Control Block Structure for a Key-Sequenced Data Set (ISAM User)

Figure 63 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.

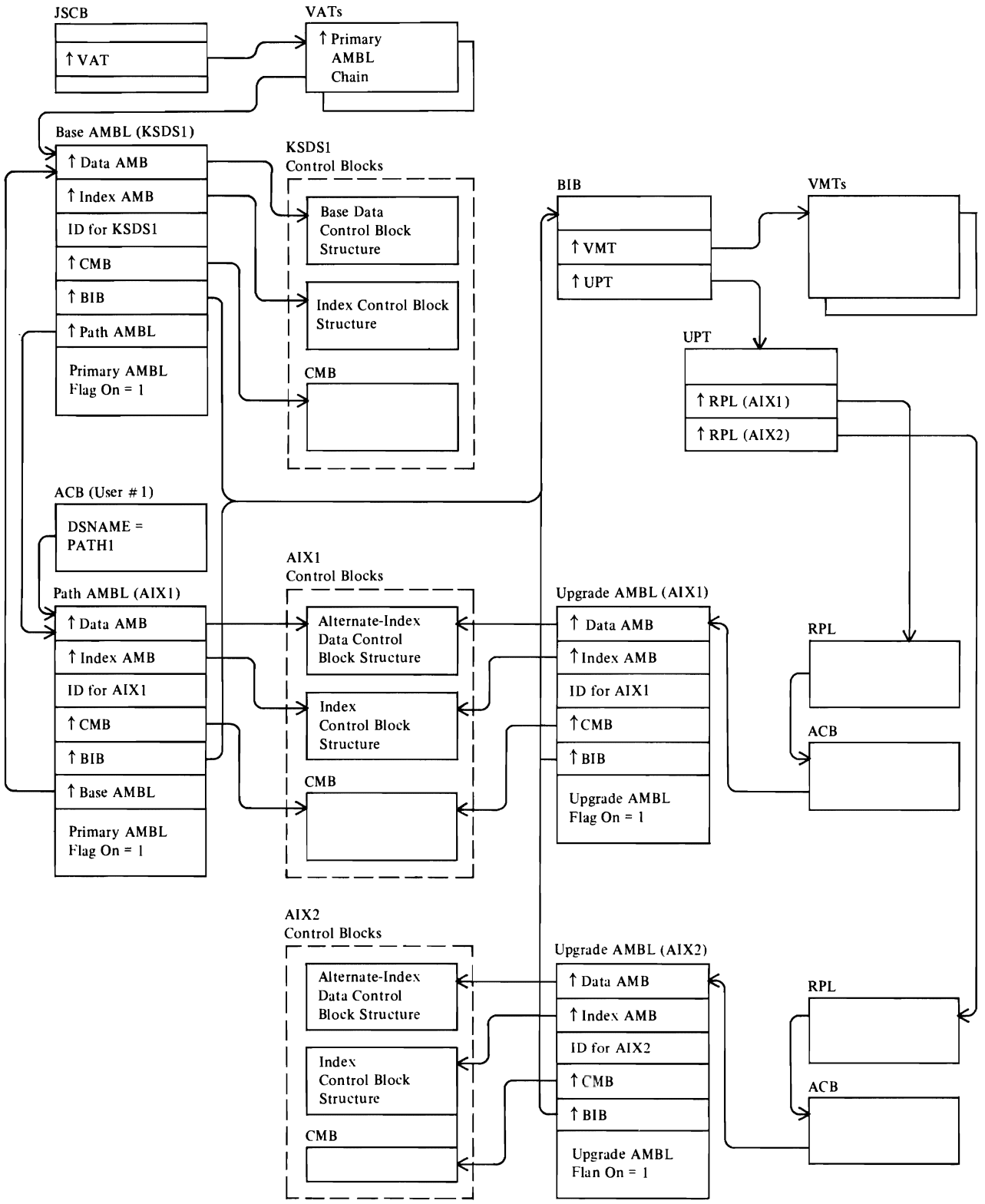



**Note:** The data control block structure is shown in Figure 66. The index control block structure is shown in Figure 68.

Figure 63. VSAM Data Set Control Blocks Before and After Data Set Sharing

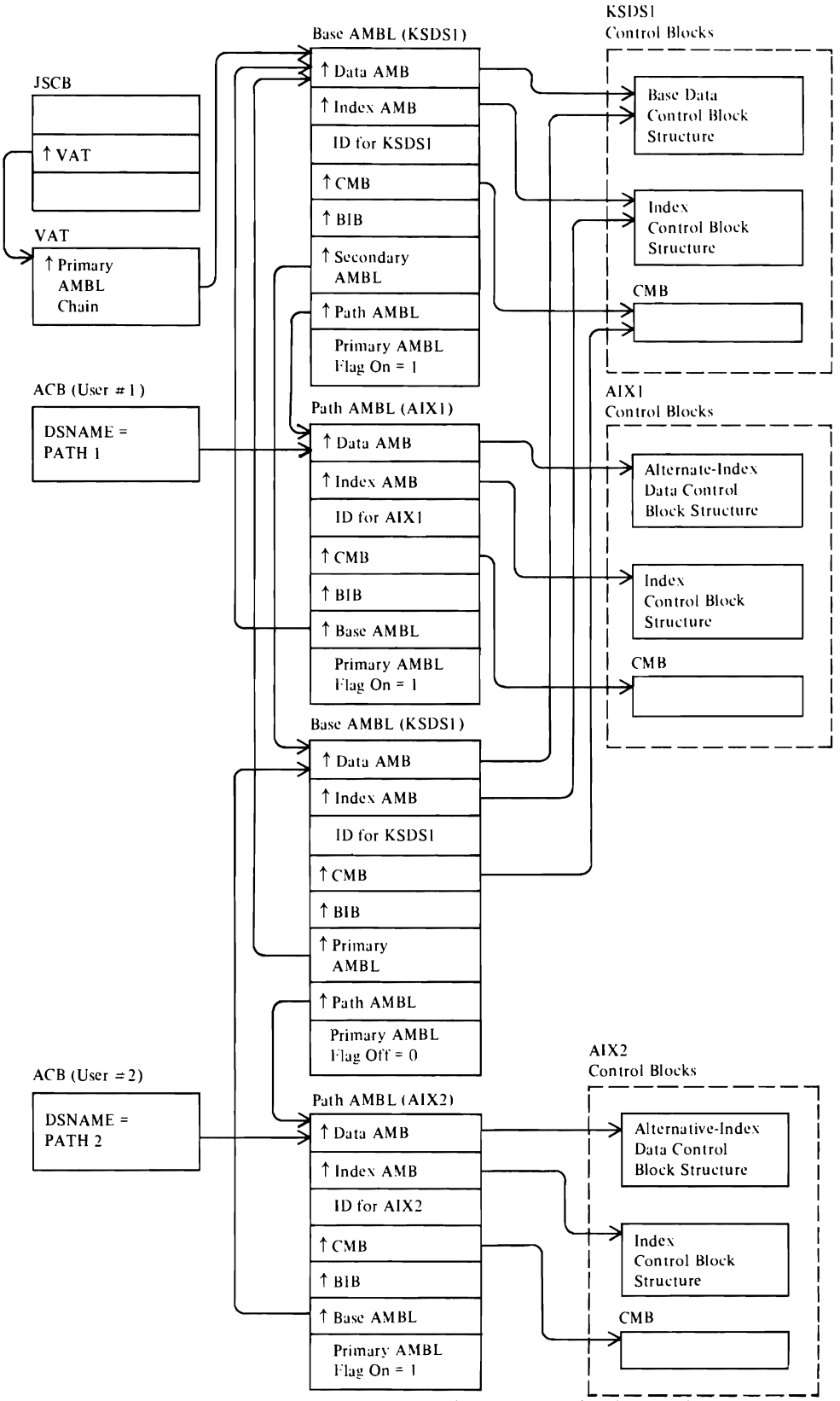
Figure 64 shows the VSAM control blocks built when a key-sequenced data set (KSDS1) is opened for access through a path (PATH1). The path alternate index (AIX1) and a second alternate index (AIX2) are members of the upgrade set for KSDS1.

Figure 65 shows the sharing of VSAM control blocks when the key-sequenced data set (KSDS1) shown in Figure 64 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, his AMBLs are deleted.



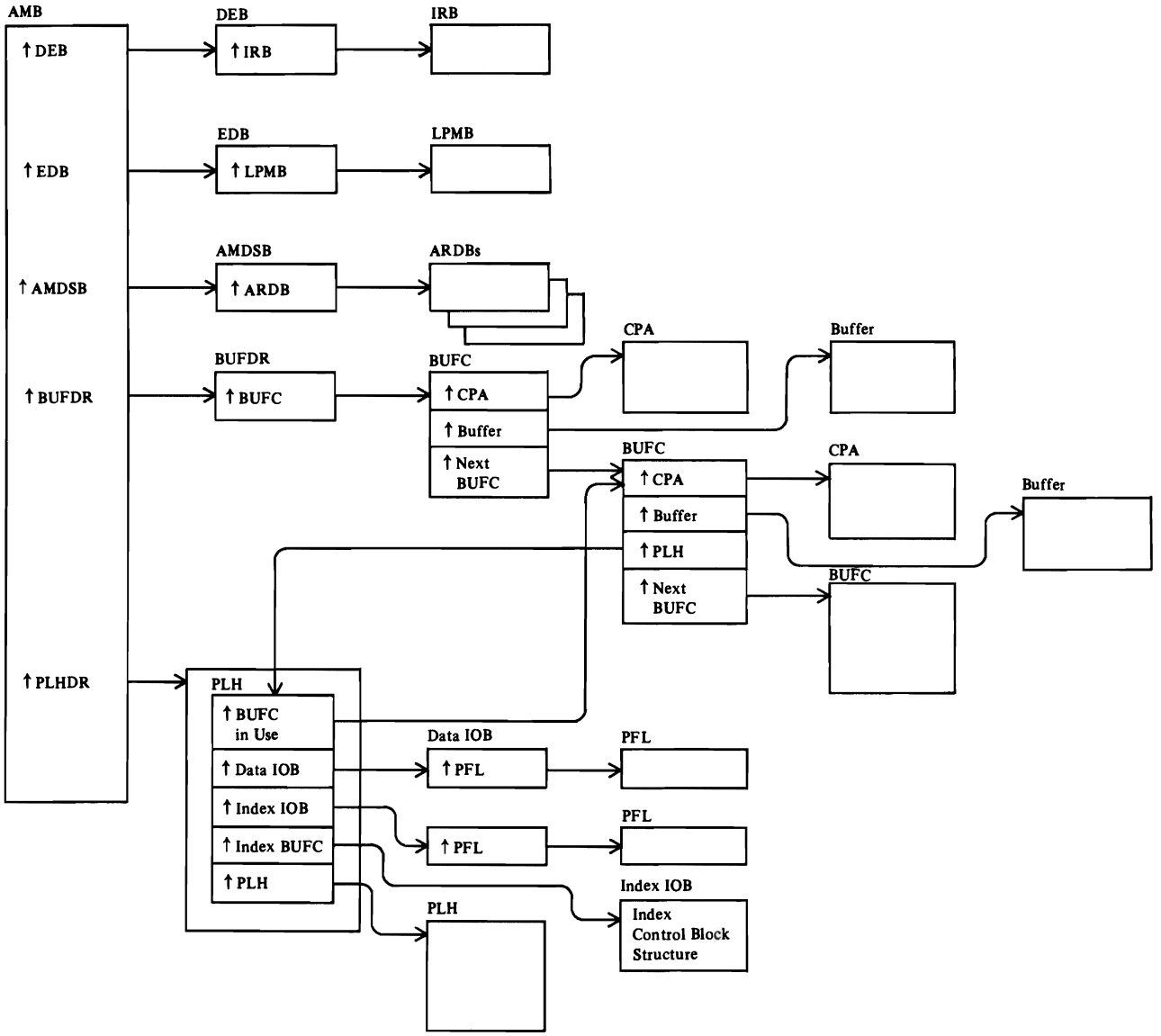

**Note:** The base data control block structure is shown in Figure 66. The alternate-index data control block structure is shown in Figure 67. The index control block structure is shown in Figure 68.

Figure 64. VSAM Control Block Structure for a Key-Sequenced Data Set Accessed through a Path




**Note:** The base data control block structure is shown in Figure 66. The alternate-index data control block structure is shown in Figure 67. The index control block structure is shown in Figure 68. The BIB-UPT-RPL-ACB-upgrade AMBL structure is shown in Figure 68. The BIB-UPT-RPL-ACB-upgrade AMBL structure (not shown) is the same as in Figure 64.

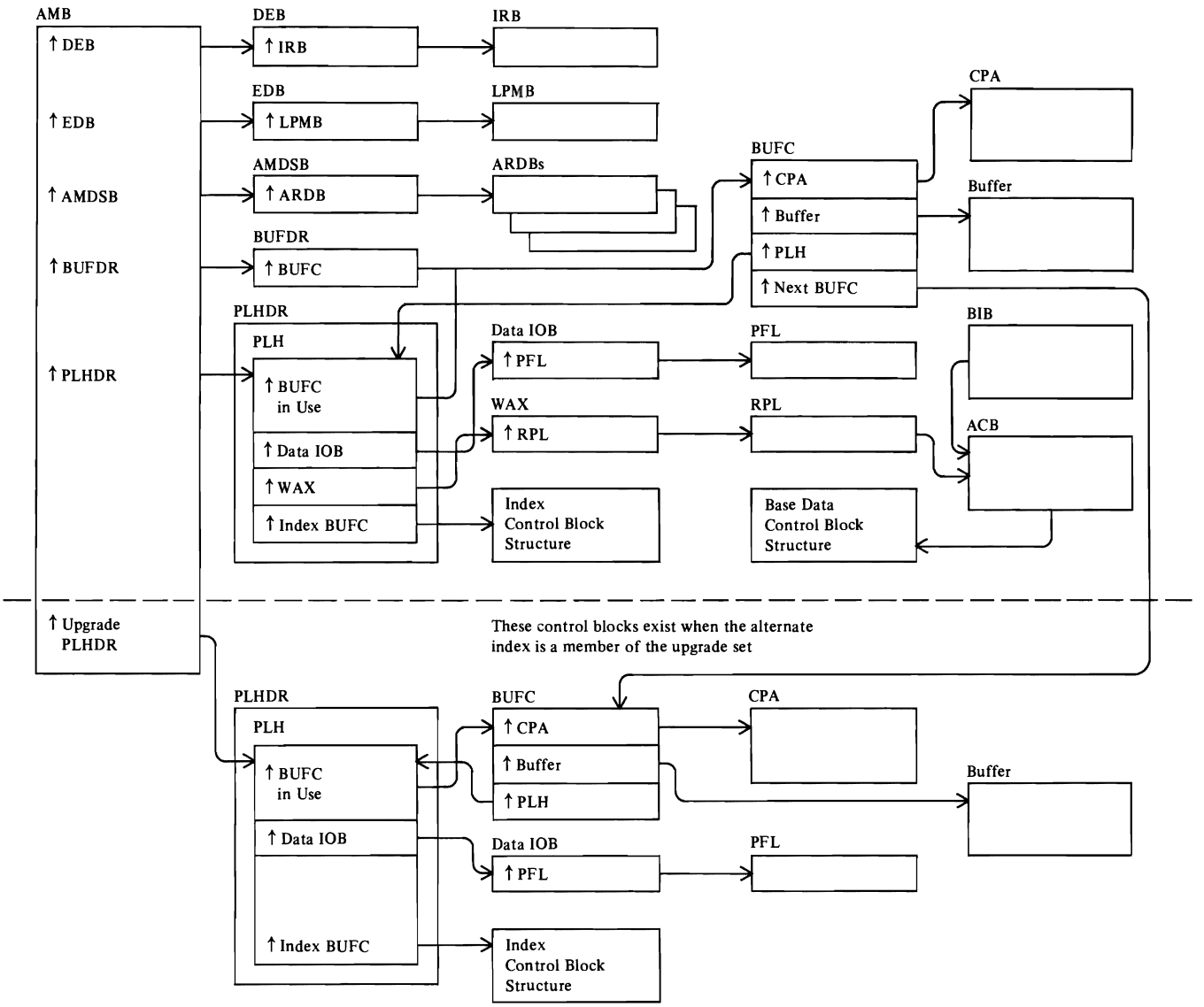
Figure 65. Shared VSAM Control Block Structure for a Key-Sequenced Data Set Accessed through Two Paths




**Note:** The index control block structure is illustrated in Figure 68.

Figure 66. Data AMB Control Block Structure

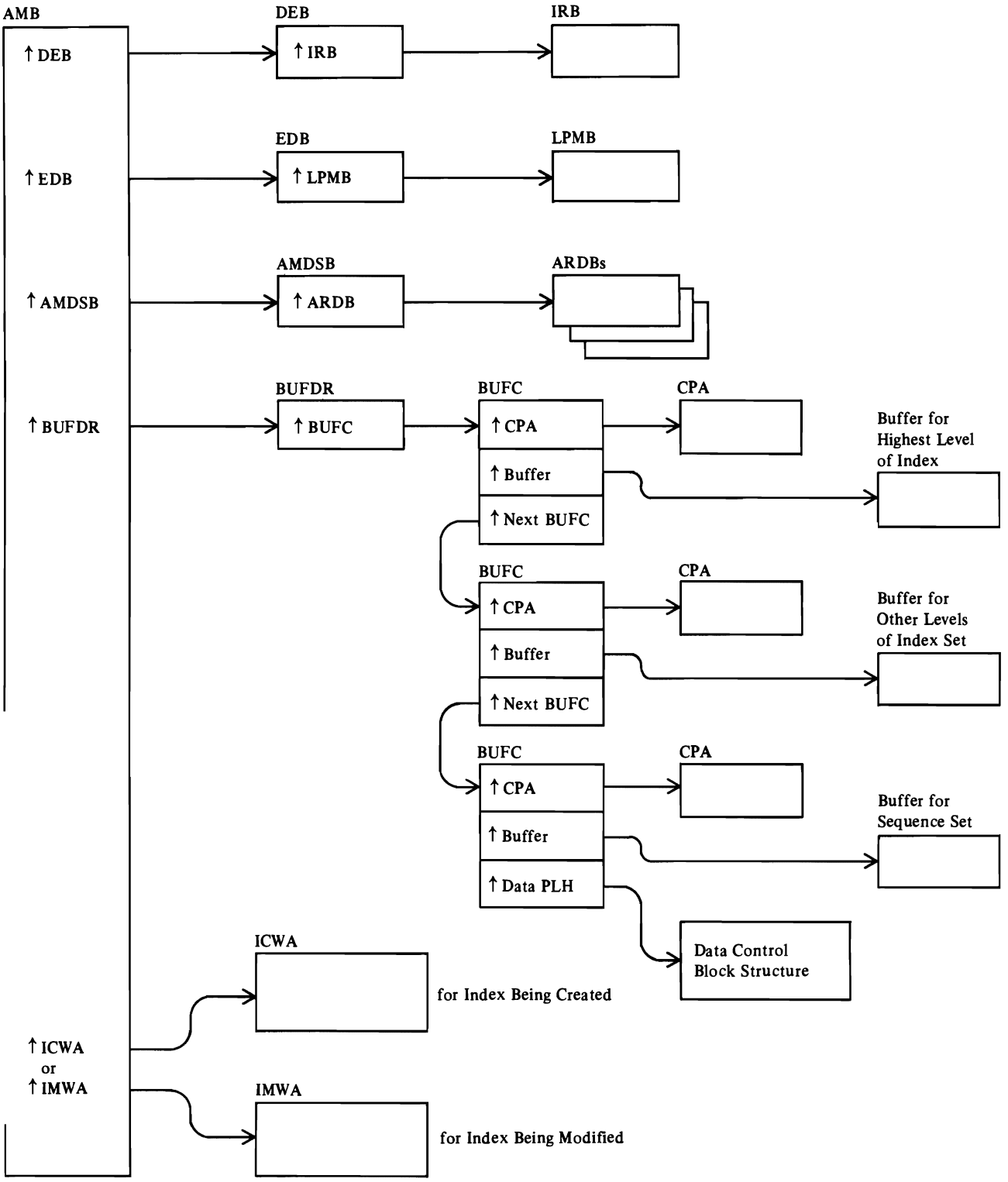
Figure 66 shows the control blocks that describe a cluster's data component to VSAM record management routines.




**Note:** The base data control block structure is shown in Figure 66. The index control block structure is shown in Figure 68.

Figure 67. Alternate-Index Data AMB Control Block Structure

Figure 67 shows the control blocks that describe an alternate index's data component to VSAM Record-Management routines.




**Note:** The data control block structure is illustrated in Figure 66.

Figure 68. Index AMB Control Block Structure

Figure 68 shows the control blocks that describe a key-sequenced cluster's index to VSAM record management routines.

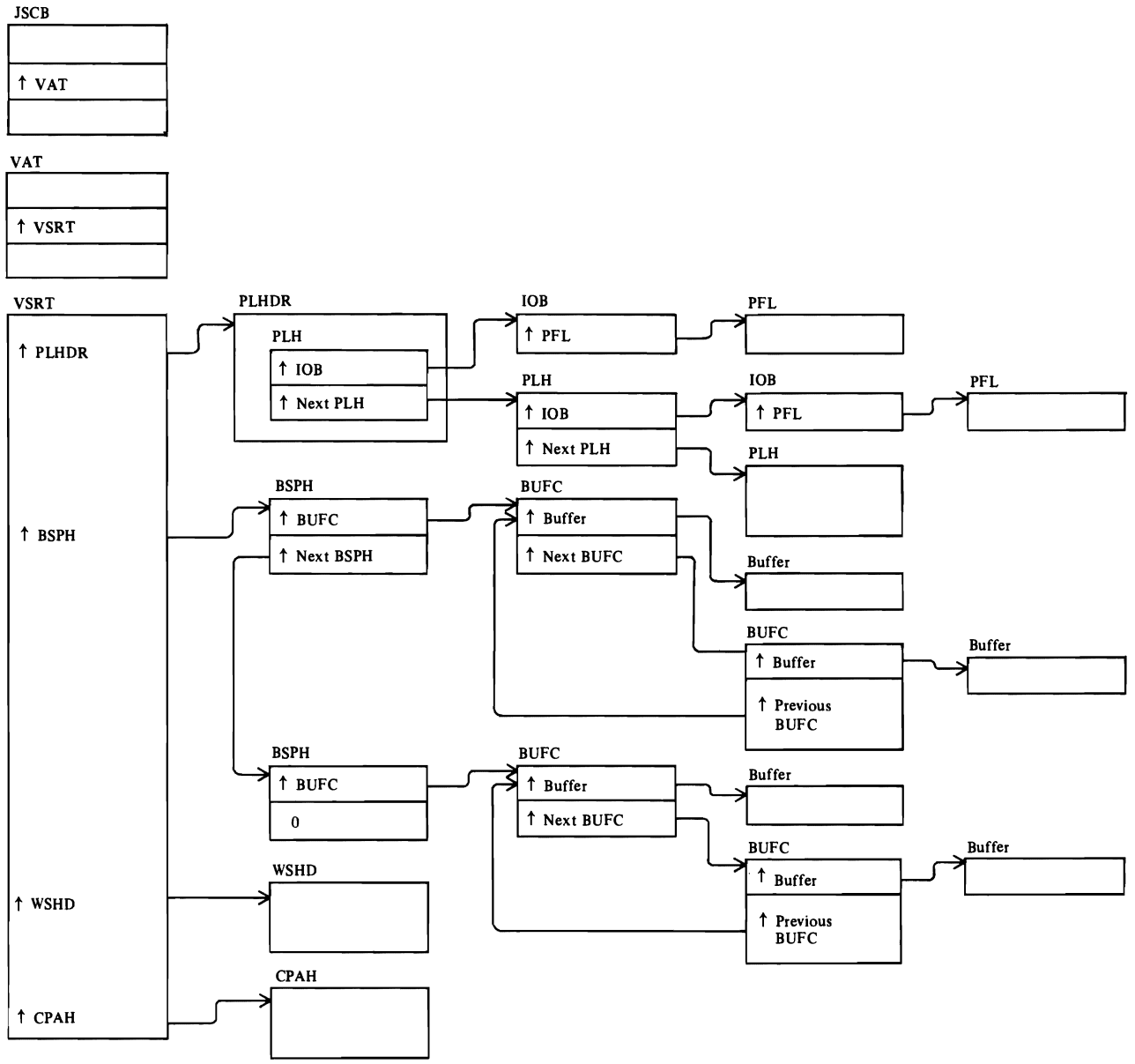



Figure 69. Local Shared Resources Control Block Structure

Figure 69 shows the VSAM control blocks built for processing with local shared resources (LSR). These control blocks describe the local VSAM resource pool.

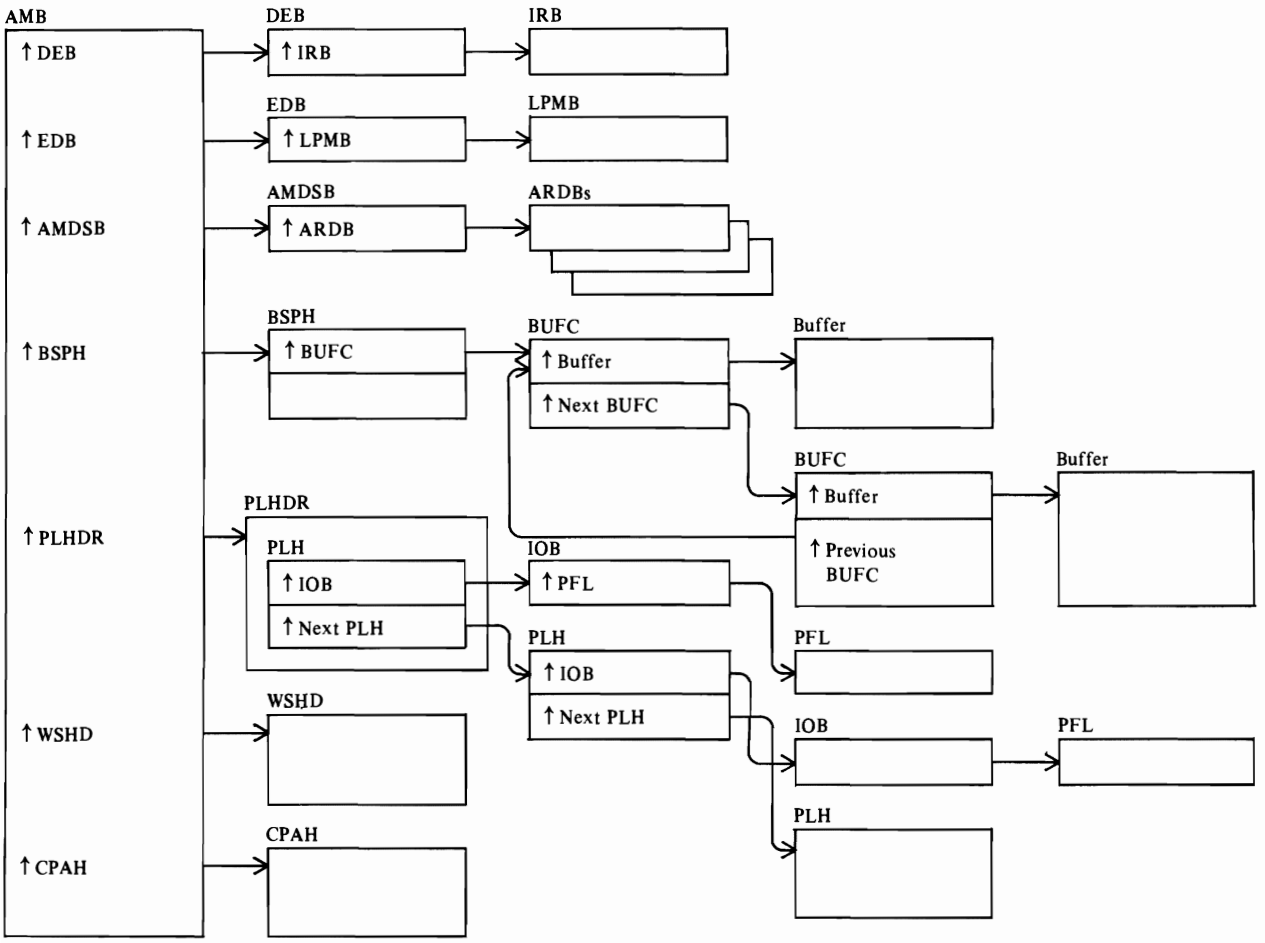


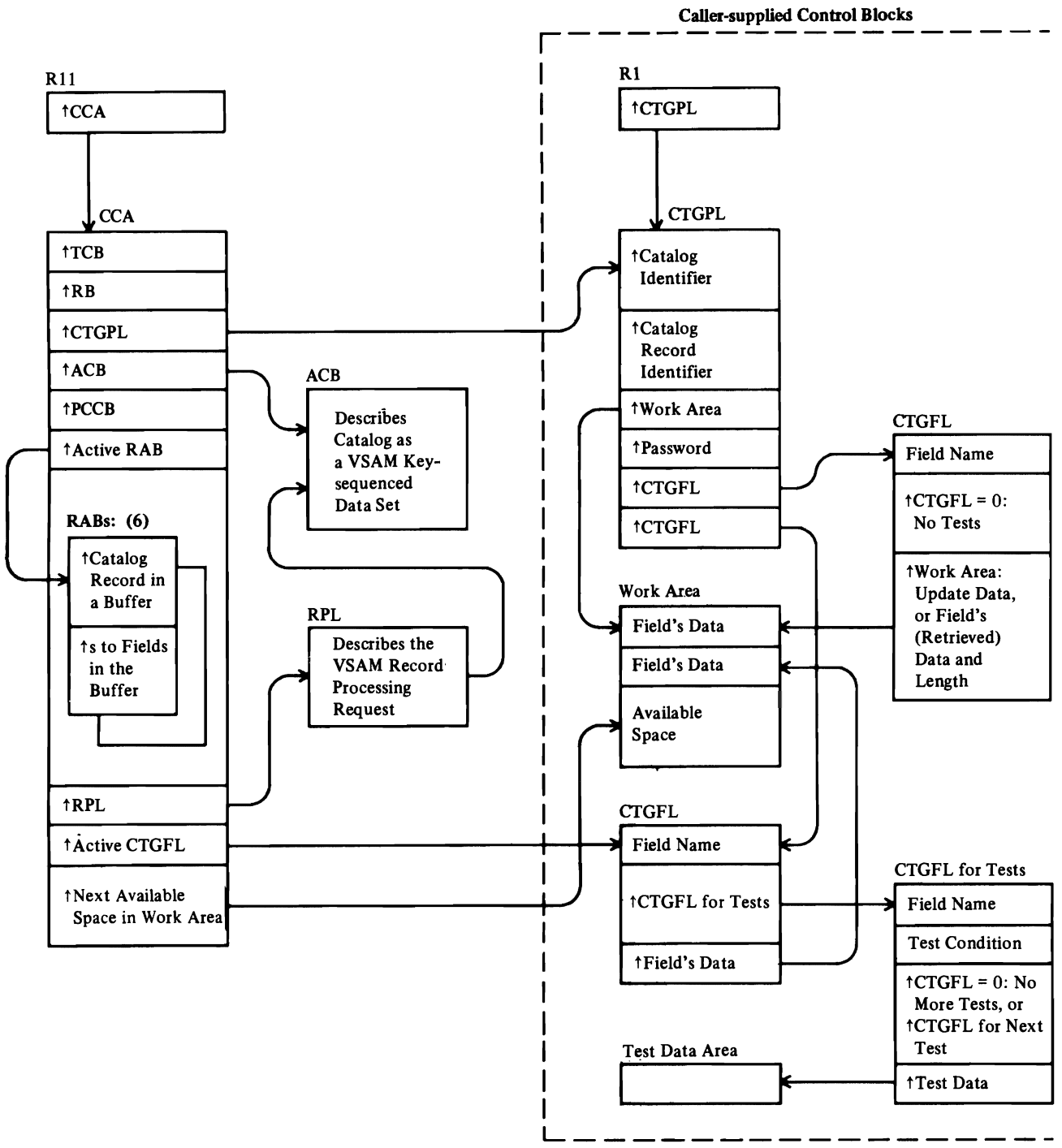

Figure 70. AMB Control Block Structure with Local Shared Resources

Figure 70 shows the AMB control block structure for processing with local shared resources (LSR). It differs from the structure for processing without shared resources, which is shown in Figures 66, 67, and 68.

## Catalog Management Control Block Interrelationships

Figure 71 shows the VSAM catalog management control blocks built when a VSAM routine calls catalog management to process a VSAM catalog record.

Figure 72, Open Catalog Control Blocks, shows the OS/VS2 system and catalog management control blocks that describe a VSAM catalog to the OS/VS2 system.




**Note:** Figure 61 illustrates the control block structure of a key-sequenced VSAM data set.

Figure 71. Catalog Management Control Blocks

Figure 73, VSAM Control Blocks That Describe a Catalog (a key-sequenced key-range VSAM data set), shows the control blocks that describe the catalog as a data set. This control block structure allows record management to read and write control intervals in the catalog, and to update the catalog's index, as required when catalog management I/O functions issue GET, PUT, and ERASE macro instructions.

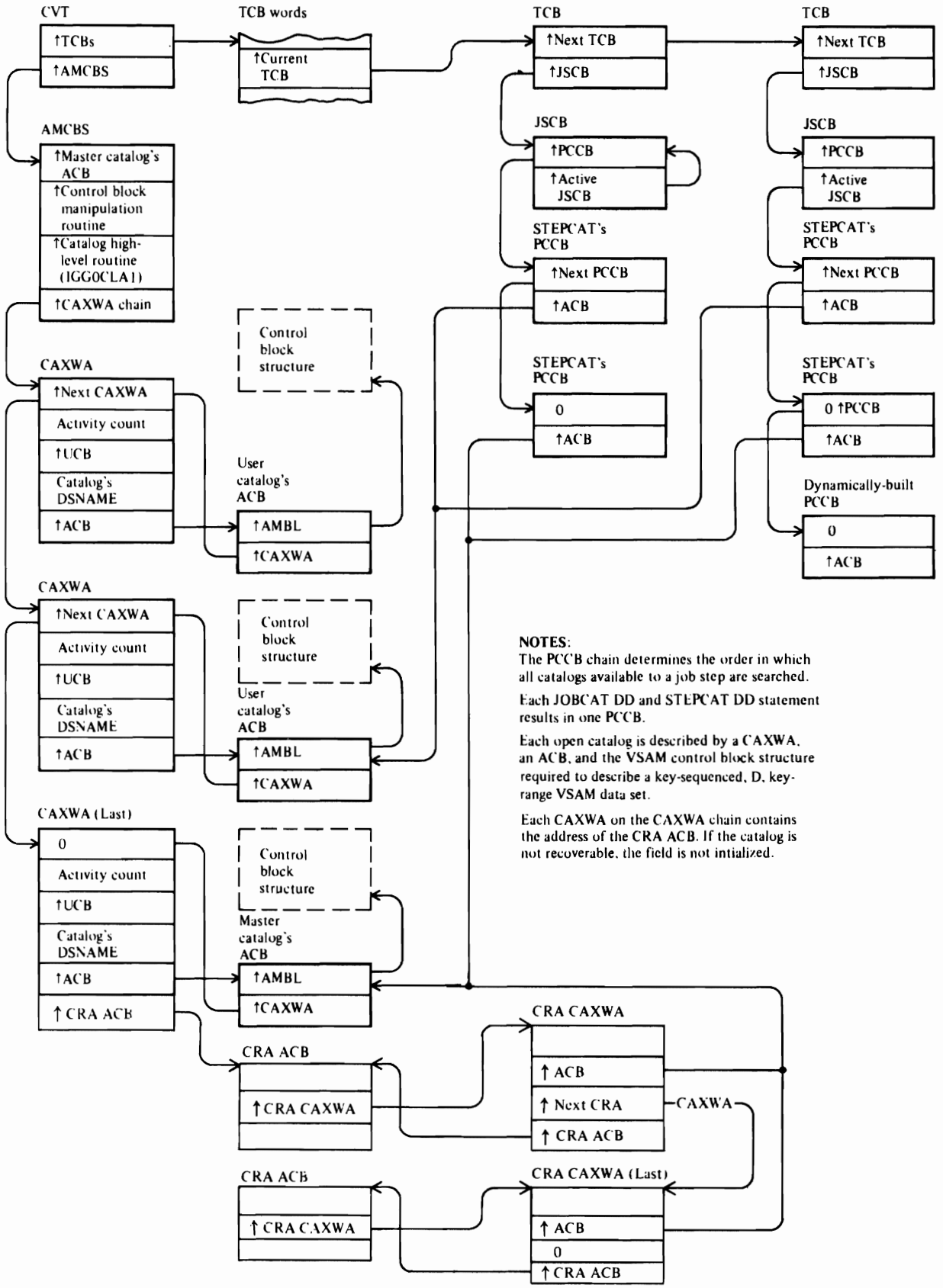



**Figure 72. Open Catalog Control Blocks**

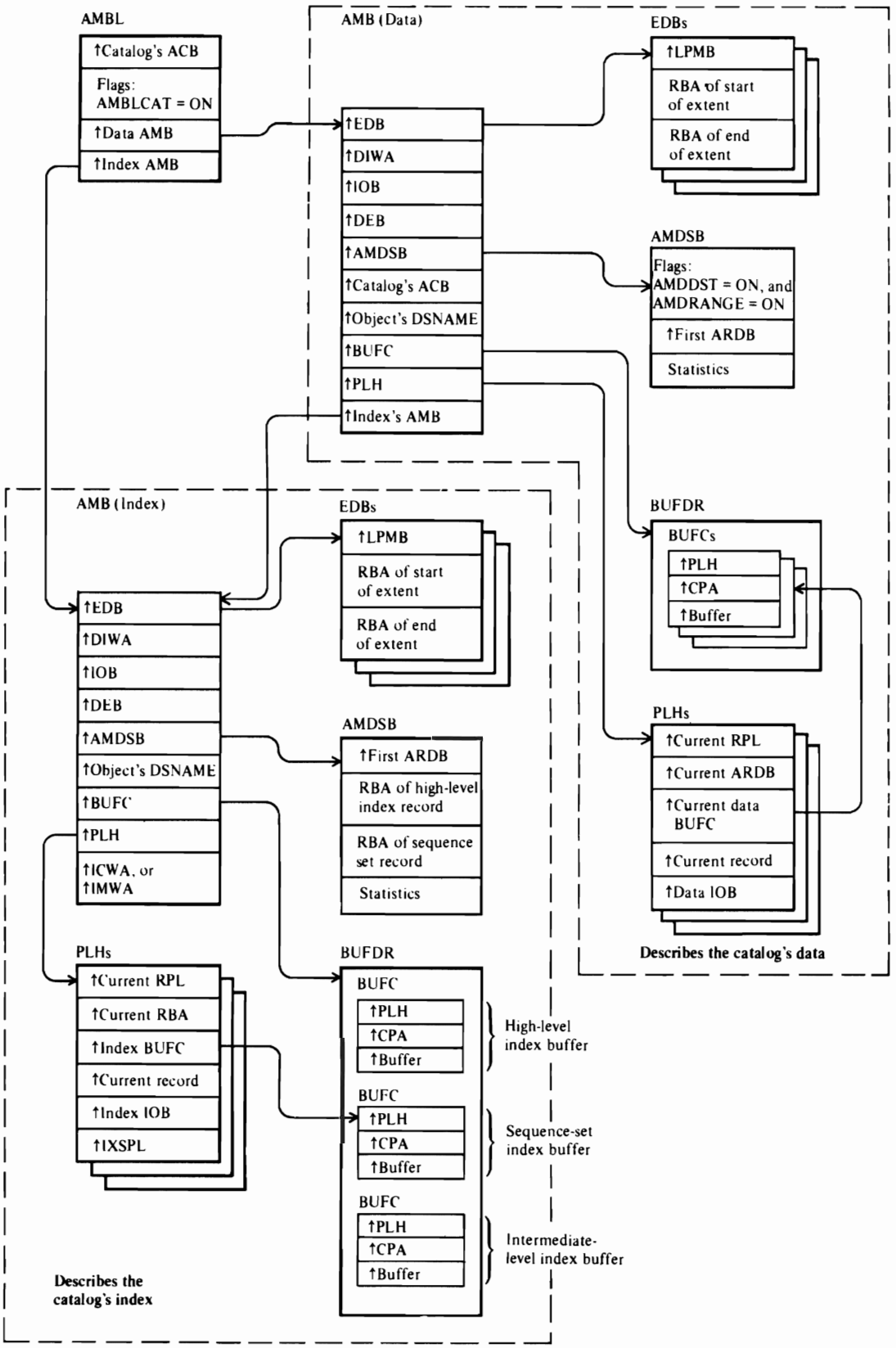



Figure 73. VSAM Control Blocks that Describe a Catalog (A Key-Sequenced, Key-Range VSAM Data Set)

# VSAM Control Block Descriptions

## *ACB—Access Method Control Block*

The VSAM ACB describes a VSAM cluster. It is built by the user's program. Before the cluster is opened, the ACB can be modified by the user's DD statements and by the ACB exit routine. After the cluster is opened, the ACB is pointed to by the RPL (RPLDACB) that describes the user's record processing request.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 1 | ACBID | Control block identifier, X'A0' |
| 1 (1) | 1 | ACBSTYP | Subtype:<br>X'10' = VSAM<br>X'20' = VTAM |
| 2 (2) | 2 | ACBLENG<br>ACBLENG2<br>ACBLEN2 | Length of the ACB |
| 4 (4) | 4 | ACBAMBL<br>ACBIXLST<br>ACBJWA<br>ACBIBCT | Address of the AMBL<br>Address of the index list |
| 8 (8) | 4 | ACBINRTN | Address of the VSAM Interface routine (IDA019R1) |
| 12 (C) | 2 | ACBMACRF | MACRF flags: |
| | Byte 1 | ACBMACR1 | MACRF flag byte 1: |
| | 1... .... | ACBKEY | The record is identified by a key—keyed processing |
| | .1.. .... | ACBADR<br>ACBADD | The record is identified by a RBA (relative byte address)—addressed processing |
| | ..1. .... | ACBCNV<br>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) | Byte 2 | 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 | Entity 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) |

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 20 (14) | 1 | ACBMACR3 | MACRF flag byte 3 |
| | .1.. .... | ACBLSR | Local shared resources |
| | ...1 .... | ACBICI | Improved control interval processing |
| | .... 1... | ACBDFR | Write operations are to be deferred |
| | .... .1.. | ACBSIS | Sequential insert strategy |
| | .... ..0. | ACBNCFX | Control blocks are not fixed |
| | .... ..1. | | Control blocks are fixed |
| | x.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 |
| | ..1. .... | ACBCPACD | Compaction table must be passed (JES/RES) |
| | ...1 .... | ACBPDIR | PDIR must be passed (JES/RES) |
| | .x.. xxxx | | Reserved |
| 25 (19) | 1 | ACBCCTYP | Control character: |
| | nn.. .... | ACBTRCID | 3800 translate table identifier |
| | .... nnnn | ACBASA | Control character type |
| | ..xx .... | | Reserved |
| 26 (1A) | 2 | ACBOPT<br>ACBDSORG | Non-user options<br>Match ACBDORGA with DCBDSORG |
| | | | Byte 1: |
| | xx.. .... | ACBCROPS | Checkpoint/restart options: |
| | 1... .... | ACBCRNCK | Restart has not checked for notification since last checkpoint |
| | .1.. .... | ACBCRNRE | Data added since last checkpoint has not been erased by restart, and no reposition to last checkpoint takes place |
| | ..1. .... | ACBDVIND<br>ACBOPTJ | 3800 device type indicator |
| | ...x xxxx | | Reserved |
| | | | Byte 2: |
| | .... 1... | ACBDORGA | 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<br>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 |

| Offset | Bytes and Bit Pattern | | Field Name | Description |
|---|---|---|---|---|
| **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 |
| | | | ACBLOCK | The ACB is locked |
| | .... | ...1 | ACBIOSFG | The Open or Close routine is in control |
| | | | ACBBUSY | The ACB is busy |
| | xx.. | .x.. | | Reserved |
| 49 (31) | | 1 | ACBERFLG | Error flags<br>**Note:** See "Diagnostic Aids: Open, Close, and End-of-Volume Error Codes" 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 storage |
| | .... | ..1. | ACBVVIC | Data set being opened is the mass storage volume inventory (MSVI data set) |
| | .... | ...1 | ACBBYPSS | Bypass security checking on Open if user is authorized |
| | 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<br>ACBMSGLN | Length of the physical DASD record<br>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 CBM work area |
| 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.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 1 | AMBID | Control block identifier, X'40' |
| 1 (1) | 1 | AMBRSC | Resource test and set byte |
| 2 (2) | 2 | AMBLEN | Length of the AMB |
| 4 (4) | 4 | AMBLINK | Address of the next AMB in the 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) |
| 20 (14) | 4 | AMBDSB | Address of the AMDSB |
| 24 (18) | 1 | AMBEOVR | End of volume request type: |
| | X'01' | | Mount by key |
| | X'81' | | Mount by RBA |
| | X'02' | | Allocate by key |
| | X'82' | | Allocate by RBA |
| 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 VSAM master catalog |
| | ...1 .... | AMBUCAT | The AMB describes a VSAM 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... | | VSAM access method |
| | xxxx .xxx | | Reserved |
| 28 (1C) | 4 | AMBIOBAD | Address of the IOB |
| 32 (20) | 3 | AMBCDSN | Data set name of the VSAM catalog |
| 35 (23) | 3 | AMBDDSN | Data set name of the object associated with the AMB |

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 38 (26) | 2 | | Reserved |
| 40 (28) | 2 | AMBTIOT | Address of the TIOT |
| 42 (2A) | 1 | AMBINFL | Indicator flags: |
| 42 (2A) | ...1 .... | AMBCAT | The AMB describes a VSAM catalog's data set or index |
| | .... 1... | AMBSCRA | Catalog recovery area 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 |
| 44 (2C) | 1 | AMBDEBPT | Address of the DEB and error flags: |
| 44 (2C) | 1 | | Error flags |
| 45 (2D) | 3 | AMBDEBAD | Address of the DEB |
| 48 (30) | 1 | AMBOFLGS | Open status flags: |
| | ...1 .... | AMBOPEN | The AMB is open |
| | .... .11. | AMBEXFG | User exit routines are active |
| | .... ...1 | AMBBUSY | Busy bit |
| | xxx. x... | | Reserved |
| 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 key or RBA to be used by the VSAM End of Volume routine |
| 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 | AMBIXP | Address of the index's AMB if this is a data AMB for a key-sequenced data set. |
| 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 | Flat byte: |
| | .1.. .... | AMBLSR | Local 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. ...x | | Reserved |
| 85 (55) | 1 | | Reserved |
| 86 (56) | 2 | AMBRDCNT | Number of control intervals read for this AMB |
| 88 (58) | 4 | AMBBM2SH | Address of PLH doing second search of subpool |

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 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 | | Reserved |

## *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).

| 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) | 4 | AMBLEOV | Work area for End of Volume and Record Management routines |
| 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 |

## Access Method Block List (AMBL)—Description and Format

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 23 (17) | 1 | AMBLQ | Qualifier: |
| | 1... .... | AMBLDDC | DD connect only |
| | ..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 |
| | .x.. .... | | Reserved |
| 24 (18) | 4 | 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 identifies 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 |
| 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 |

## Access Method Block List (AMBL)—Description and Format

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 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:<br>001 Catalog<br>101 MSVI<br>011 SCRA |
| 36 (24) | 1 | AMBLFLG2 | Flags: |
| | 1... .... | AMBLREST | A Restart routine is in control |
| | .1.. .... | AMBLCKPT | A Checkpoint routine is in control |
| | ..1. .... | AMBLRST1 | Update the high RBA for Restart |
| | ...1 .... | AMBLSTAG | The cluster is staged |
| | .... 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 key-sequenced cluster's index AMB |
| 60 (3C) | 4 | AMBLBIB | Address of the base information block |
| 64 (40) | 4 | AMBLCMB | Address of the cluster management block |

## *AMCBS—Access Method Control Block Structure Block*

The AMCBS contains information that is used by OS/VS to locate the master catalog and user catalogs. The AMCBS is completed when the master catalog is opened. IDA019C1, a VSAM module in located in the pageable nucleus, contains the AMCBS at entry point IDA019C2. The CVT (CVTCBSP) points to the AMCBS.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 2 | CBSID | AMCBS ID character |
| 2 (2) | 2 | CBSSIZ | Length of the AMCBS |
| 4 (4) | 4 | | Reserved |
| 8 (8) | 4 | CBSACB | Address of the VSAM master catalog's ACB |
| 12 (C) | 4 | CBSCBP | Address of the control block manipulation routine |
| 16 (10) | 4 | CBSCMP | Address of the catalog management high-level routine (IGG0CLA1) |
| 20 (14) | 4 | CBSCAXCN | Address of the CAXWA chain |
| 24 (18) | 4 | CBSCRACA | Address of the CRA CAXWA chain |
| 28 (1C) | 4 | CBSCRTCB | Address of CRA task TCB |
| 32 (20) | 8 | CBSVSRT | CDS (compare and swap double) word for VSRT (VSAM shared resource table) |
| | | CBSVUSE | VSRT use count |
| | | CBSVPTR | Address of the VSRT |

## *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.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 1 | AMDSBID | AMDSB 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 is replicated. |
| | ...1.... | AMDREPL | All index records are replicated—duplicated around the track. |
| | .... 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. | AMDRRDS | 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 |

## Access Method Data Set Statistics Block (AMDSB)—Description and Format

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 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:<br>0 Unique keys<br>1 Nonunique keys |
| | .x..... | AMDFAULT | The data set is staged:<br>0 At open time, if required<br>1 By cylinder fault |
| | ..x..... | AMDBIND | The data set is:<br>0 Not bound<br>1 Staged and bound |
| | ...x.... | AMDWAIT | After destaging is begun, control is returned to the program that is closing the data set:<br>0 Immediately<br>1 After destaging is finished |
| | .... x... | AMDLM | 0 Load mode, or data set is not loaded<br>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 |

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 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 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).

| 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 (1) | 1... .... | ARDKR | One key range of a key-range data set |
| | .1.. .... | ARDHLI | The total index of a key-sequenced data set, or<br>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... | ARDEOD | The key range containing the highest data RBA in the data set |
| | .... .1.. | ARDUSED | The ARDHRBA field's initial value has changed |
| | ...x ..xx | | Reserved |
| 2 (2) | 2 | ARDLEN | Length of the ARDB |
| 4 (4) | 4 | ARDNPTR | Address of the next 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 |

### Address Range Definition Block (ARDB) Description and Format

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 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).

| 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 |
| | .xxx 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 |
| 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) | 16 | BIBRTNS | Addresses of Record-Management routines: |
| 48 (30) | 4 | BIBINTRF | VSAM interface (IDA019R1) |
| 52 (34) | 4 | BIBCEAPP | Channel end appendage |
| 56 (38) | 4 | BIBASYRT | Asynchronous Routine |
| 60 (3C) | 4 | BIBSIOAP | Start-I/O appendage |

## *BLPRM—Resource Pool Parameter List*

BLPRM is created by the BLDVRP and DLVRP macros. It is used by Record Management for dynamic string additiona dn by data set management 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 | ACBID—X'AO' |
| 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 | BULPIOPLH | 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 |
| 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) |

**Resource Pool Parameter List (BLPRM)—Description and Format**

| Offset | Bytes and Bit Patterns | Field Name | Description |
|---|---|---|---|
| 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:*

| | | | |
|---|---|---|---|
| 0 (0) | 4 | BLPBUFSZ | Buffer size |
| 4 (4) | 1 | BLPBRLFG | Buffer list flags: |
| | 1... .... | BLPBFLST | Last buffer request |
| | .xxx xxxx | | 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.

| 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 |
| 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. The BUFC is the interface between the I/O Manager routine (IDA019R3) and the Buffer Manager routines (IDA019R2 and its procedures). The BUFC is pointed to by the PLH (PLHBUFC points to the data BUFC; PLHIBUFC points to the index BUFC).

The buffer header and the buffer control entries 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.

| 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 BUFC entry |
| 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. |
| 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** | | | |
| 0 (0) | 1 | BUFCAVL | Test-and-set byte for the buffer. |
| 0 (0) | 1 | BUFCUCNT | Use count |

## Buffer Control Block (BUFC)—Description and Format

| Offset | Bytes and Bit Pattern | | Field Name | Description |
|---|---|---|---|---|
| 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. |
| | ...x | .... | BUFCREAL | Reserved. |
| | .... | 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. |
| | .... | ...x | BUFCFIX | Reserved. |
| 3 (3) | | 1 | BUFCFLG2 | Flag byte 2: |
| | ..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 |
| | xx.. | 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 | BUFCDD | 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. |

**Buffer Control Block (BUFC)—Description and Format**

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 28(1C) | 8 | | Reserved |
| 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 |

## *CAXWA—Catalog Auxiliary Work Area*

The CAXWA is built when a VSAM master or user catalog is opened or is being created. The CAXWA contains the addresses of control blocks and work areas needed when a catalog is being opened or created, such as the alternate TIOT, the DRWA, and the UCB. The CAXWA also contains flags that indicate the type of processing being performed on the catalog and the OS/VS component that invoked the processing. The CAXWA is pointed to by the ACB (ACBUAPTR). The AMCBS (CBSCAXCN) contains the address of the CAXWA chain.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 1 | CAXID | Control block identifier, X'CA' |
| 1 (1) | 3 | | Reserved |
| 4 (4) | 4 | CAXCHN | Address of the next CAXWA in the chain |
| 8 (8) | 1 | CAXFLGS | Flags: |
| | 1... .... | CAXBLD | Build request |
| | .1.. .... | CAXOPN | The catalog is being opened |
| | ..1. .... | CAXCLS | The catalog is being closed |
| | ...1 .... | CAXEOV | An End of Volume routine is in control |
| | .... 1... | CAXCMP | Open/Close/EOV processing is complete |
| | .... .1.. | CAXMCT | Identifies the VSAM master catalog |
| | .... .0.. | | Identifies a VSAM user catalog |
| | .... ..1. | CAXCMR | Catalog management (SVC 26) has been called by a catalog management routine |
| | .... ...1 | CAXSCR | Catalog Management (SVC 26) has been called by the OS/VS Scheduler |
| | .... ..00 | | Catalog management (SVC 26) has been called by an Access Method Services procedure |

## Catalog Auxiliary Work Area (CAXWA)—Description and Format

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 9 (9) | 1 | CAXFLG2 | Flags: |
| | 1... .... | CAXF2DT | The catalog has been deleted |
| | | | The following flags are set by IFG0191X and IFG0200N: |
| | .1.. .... | CAXF2NDD | No DDNAME found |
| | ..1. .... | CAXF2NCR | Unable to obtain virtual storage with a GETMAIN request |
| | ...1 .... | CAXF2IOE | I/O error |
| | .... 1... | CAXF2CLR | RPL cleanup request |
| | .... .1.. | CAXF2CA | If an error occurs, free the CAXWA |
| | .... ..1. | CAXF2REC | Recoverable catalog |
| | .... ...1 | CAXF2VTU | Volume timestamp updated |
| 10 (A) | 1 | CAXFLG3 | Flags: |
| | 1... .... | CAXF3AT | CRA alternate TIOT exists |
| | .1.. .... | CAXF3ANE | CRA does not exist |
| | .... 00.. | CAXF3B5 | Password not read |
| | .... 11.. | | No passwords |
| | .... 01.. | CAXF3B6 | Master password; no update password |
| | .... 10.. | | Master and update passwords |
| | ..xx ..xx | | Reserved |
| 11 (B) | 1 | CAXACT | Catalog activity count |
| 12 (C) | 4 | CAXATIOT | Address of the alternate TIOT |
| 16 (10) | 4 | CAXSCHWA | Address of the Scheduler work area |
| 20 (14) | 4 | CAXDRWP | Address of the catalog's DRWA |
| 24 (18) | 4 | CAXACB | Address of the catalog's ACB |
| 28 (1C) | 4 | CAXUCB | Address of the UCB |
| 32 (20) | 12 | CAXCCR | Catalog control record information |
| 32 (20) | 3 | CAXHACI | Control interval number of the highest allocated control interval in the catalog |
| 35 (23) | 3 | CAXNFCI | Control interval number of the next free control interval in the catalog |
| 38 (26) | 3 | CAXCDCI | Number of deleted control intervals |
| 41 (29) | 3 | CAXFDCI | Control interval number of the first deleted control interval in the catalog |
| 44 (2C) | 2 | | Reserved |
| 46 (2E) | 2 | CAXRPLCT | Number of RPLs associated with the CAXWA |
| 48 (30) | 4 | CAXRPL | Address of the first RPL in the CAXWA's RPL chain |
| 52 (34) | 44 | CAXCNAM | Catalog's dsname |
| 52 (34) | 6 | CAXVOLID | CRA volume serial |
| 58 (3A) | 4 | CAXRACTS | CRA creation timestamp |
| 62 (3D) | 4 | CAXRATEP | CRA's TIOT entry address |
| 66 (42) | 8 | CAXRADDN | CRA's DDNAME |
| 74 (4A) | 22 | | Reserved for CRA |

**Catalog Auxiliary Work Area (CAXWA)—Description and Format**

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 96 (60) | 4 | CAXOPLST | Open/Close parameter list: |
| 96 (60) | 1 | COPTS | Option flags: |
| | 1... .... | CENLST | End of list indicator |
| | .xxx xxxx | | Reserved |
| 97 (61) | 3 | COPACB | Address of the catalog's ACB |
| 100 (64) | 4 | CAXOPEWA | Address of the Open/Close/End of Volume work area |
| 104 (68) | 4 | CAXCCA | Address of the CCA |
| 108 (6C) | | CAXJDE | Address of the JDE |
| 112 (70) | | CAXCRACB | Address of the CRA's ACB |

## *CCA—Catalog Communications Area*

The CCA is built each time an OS/VS component issues the CATLG macro instruction (SVC 26) to process a catalog record. The CCA contains information about the catalog being processed, and about the catalog record and its extensions contained in each of the six buffers available to process the user's request. The CCA is used to pass information between catalog management procedures. Register 11 contains the address of the CCA.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 2 | CCAID | Identifier, X'ACCA' |
| 2 (2) | 2 | CCASZ | Length of CCA |
| 4 (4) | 4 | CCAPROB | Problem determination: |
| 4 (4) | 2 | CCAMODID | Error module ID |
| 6 (6) | 2 | CCAERRCD | Error code |
| 6 (6) | 1 | CCAREASN | Set reason code |
| 6 (6) | 1 | CCACDR | Refer reason code |
| 7 (7) | 1 | CCARETRN | Set return code |
| 7 (7) | 1 | CCACD1 | Refer return code |
| 8 (8) | 5 | | Reserved |
| 13 (D) | 1 | CCACD2 | Return code 2 |
| | | | **Note:** See "Diagnostic Aids: Catalog Management Error Codes" for a list of the VSAM Catalog Management return codes and error codes. |

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 14 (E) | 1 | CCAFLG1 | Flags Byte 1: |
| | 1... .... | CCAF1LPS | Stop the loop |
| | .x.. .... | | Reserved |
| | ..1. .... | CCAF1LRD | Catalog control record read into virtual storage |
| | ...1 .... | CCAF1KEY | Retrieve the catalog record based on a DSNAME value |
| | .... 1... | CCAF1KGE | Retrieve the next catalog record |
| | .... .1.. | CCAF1CR | A checkpoint of the catalog control record is required |
| | .... ..1. | CCAF1UP | GET for update |
| | .... ...1 | CCAF1DK | When the caller is renaming a data set, this flag indicates that the data set's true name record is to be deleted, but the data set's catalog record is not to be deleted. |
| 15 (F) | 1 | CCAFLG2 | Flags Byte 2: |
| | 1... .... | CCAF2SYS | The caller is an OS/VS system module |
| | .1.. .... | CCAF2NVC | No validity check on the caller's CTGFL or work area is required |
| | ..0. .... | CCAF2CCT | Search all catalogs available to the caller. |
| | ..1. .... | | Search the first VSAM catalog specified by the caller's STEPCAT statement. If the caller's JCL doesn't include a STEPCAT statement, search the VSAM master catalog. In either case search only one catalog. |
| | ...1 .... | CCAF2XEQ | Exclusive enqueue |
| | ...0 .... | | Shared enqueue |
| | .... 1... | CCAF2RHS | When a catalog management routine calls the VSAM Open routines to open a newly created catalog, and the Open routines call VSAM Catalog Management routines to obtain information about the catalog to be opened, the situation is called a "recursive call." The catalog cannot be dequeued when the Catalog Management routines return to the caller (VSAM Open routines). |
| | .... .xx. | CCAF2COB | Both catalog open and build: |
| | .... .1.. | CCAF2CO | Catalog is being opened |
| | .... ..1. | CCAF2CB | Catalog is being created |
| | .... ...1 | CCAF2SMO | Search the master catalog only |
| 16 (10) | 1 | CCAFLG3 | Flags Byte 3: |
| | 1... .... | CCAEXGR1 | Exit indicator |
| | .1.. .... | CCAGC4 | The catalog record contains a password information set of fields, identified by Type Code 4 (detected during IGGPSCNC processing) |
| | ..1. .... | CCAGDSP | Caller specified the GENDSP option (detected during IGGPSCNC processing) |
| | ...1 .... | CCAEXGR2 | Exit indicator |
| | .... 1... | CCANF | The set of fields cannot be found |
| | .... .1.. | CCAELC2 | Exit indicator |
| | .... ..1. | CCALFT | First time |
| | .... ...1 | CCAEGREC | Exit indicator |

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 17 (11) | 1 | CCAFLG4 | Flags Byte 4: |
| | 1... .... | CCAF4DRQ | The catalog must be dequeued after the request completes |
| | .1.. .... | CCAF4BYS | Bypass the security verification |
| | ..1. .... | CCAGVNC | The required variable-length field is not completely contained in the record currently in the buffer |
| | ...1 .... | CCAGVNF | The set of fields identified by the caller-specified sequence number cannot be found |
| | .... 1... | CCAGVNBS | There is no buffer space available to contain an extension record |
| | .... .1.. | CCAGVEX | Exit indicator |
| | .... ..1. | CCAGVNE | The field does not exist in the located set of fields |
| | .... ...1 | CCATCOMP | Test complete: all set-of-fields pointers have been examined and all designated fields have been tested |
| 18 (12) | 1 | CCAFLG5 | Flags Byte 5: |
| | 1... .... | CCAMEX2 | Exit indicator |
| | .1.. .... | CCAMEX | Exit indicator |
| | ..1. .... | CCAMEX1 | Exit indicator |
| | ...1 .... | CCAMODPA | The catalog record's base record must be written (using IGGPPAD) into the catalog |
| | .... 1... | CCATHIT | Successful test: a set of fields has been found that satisfies the test conditions |
| | .... .1.. | CCATEX | Exit indicator |
| | .... ..1. | CCATEX1 | Exit indicator |
| | .... ...1 | CCATEX2 | Exit indicator |
| 19 (13) | 1 | CCAFLG6 | Flags Byte 6: |
| | 1... .... | CCAMCODR | The catalog must be dequeued when the request completes |
| | .1.. .... | CCADELP | A deleted set-of-fields pointer was found |
| | ..1. .... | CCAMNOSP | The catalog record's free space isn't large enough to contain all the new catalog information during the set-of-fields move operation |
| | ...1 .... | CCAINIT | Insert switch for variable-length field being retrieved |
| | .... 1... | CCASUPFD | Suppress password field information during field retrieval |
| | .... .1.. | CCAREUSE | The contents of the caller's record areas (buffers) can be used by IGGPEXT and IGGPMOD |
| | .... ..1. | CCAEXT | Set when a catalog management routine calls the Extract routine (IGGPEXT) |
| | .... ...1 | CCAMOD | Set when a catalog management routine calls the Modify routine (IGGPMOD) |
| 20 (14) | 4 | CCATCB | Address of the TCB |
| 20 (14) | 4 | CCALBCYL | Address of label cylinder data |
| 24 (18) | 4 | CCARB | Address of the RB |
| 24 (18) | 4 | CCADPL | Address of DADSM parameter list |
| 28 (1C) | 4 | CCACPL | Address of the caller's CTGPL |
| 32 (20) | 4 | CCAACB | Address of the catalog's ACB |
| 36 (24) | 4 | CCANPCCB | Address of the next PCCB |

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 40 (28) | 4 | CCAURAB | Address of the record area block (RAB) currently in use |
| 44 (2C) | 44 | CCASRCH | Search argument (dsname of a cluster, data set, index, catalog, or nonVSAM data set, or a volume serial number) |
| 44 (2C) | 3 | CCASRID<br>CCASRCIN | Control interval number |
| 47 (2F) | 41 | | Reserved |
| 88 (58) | 20 | CCARAB0 | Record Area Block 0: Each record area block describes the catalog record contained in one of the six catalog management buffers available for the request. RAB's 1 through 5 are identical in format to RAB 0.<br><br>**Note:** 'x' in each field name is replaced by '0' through '5' to indicate a particular RAB's field. |
| 88 (58) | 1 | CCARxFLG | Flags:<br><br>The following flag is used by IGGPEXT and IGGPMOD: |
| | 1... .... | CCARxUR | The RAB is in use. It cannot be used by IGGPEXT or IGGPMOD |
| | .1.. .... | CCARxU1 | The RAB is temporarily in use by IGGPEXT or IGGPMOD. It cannot be overlaid. |
| | ..1. .... | CCARxU2 | (Same as CCARxU1) |
| | ...1 .... | CCARxWR | The buffer must be written before another catalog record can be read into it |
| | .... 1... | CCARxPA | The buffer contains a new catalog record—PUT-add is required to write the record into the catalog |
| | .... .xx. | | Reserved |
| | .... ...1 | CCARxUPD | The buffer contains a GET-for-update record |
| 89 (59) | 1 | CCARORPL | Last assigned RPL index |
| 90 (5A) | 2 | | Reserved |
| 92 (5C) | 4 | CCARxREC | Address of the record in the buffer |
| 96 (60) | 12 | CCARxSEG | Addresses of parts of the catalog record: |
| | 4 | CCACPE2x | Address of the first byte after the fixed-length header fields. |
| 100 (64) | 4 | CCACPE3x | Address of the first set of fields |
| 104 (68) | 4 | CCACPE4x | Address of the first free-space byte in the record |
| 108 (6C) | 20 | CCARAB1 | Record Area Block 1 (See RAB 0 description) |
| 128 (80) | 20 | CCARAB2 | Record Area Block 2 (See RAB 0 description) |
| 148 (94) | 20 | CCARAB3 | Record Area Block 3 (See RAB 0 description) |
| 168 (A8) | 20 | CCARB4 | Record Area Block 4 (See RAB 0 description) |
| 188 (BC) | 20 | CCARAB5 | Record Area Block 5 (see RAB 0 description) |
| 208 (D0) | 1 | CCARPLK | Work area for IGG0CLAG and IGG0CLAM |

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 209 (D1) | 1 | CCARPLF | (Same as CCARPLK) |
| 210 (D2) | 1 | CCARPLX | (Same as CCARPLK) |
| 211 (D3) | 1 | CCARPLT | (Same as CCARPLK) |
| 212 (D4) | 6 | CCARPLAA | Indexes to assigned RPLs |
| 218 (DA) | 2 | | Reserved |
| 220 (DC) | 4 | CCARPL1 | Address of the RPL in use |
| 224 (E0) | 44 | CCADESA | Save area for the extent information returned by OS/VS DADSM and VSAM Catalog Management: Suballocate |
| 224 (E0) | 1 | CCANDEXT | Number of extents |
| 225 (E1) | 1 | CCAIXEXT | Extent index value |
| 226 (E2) | 2 | CCASSVOL | Sequence number of the data set directory entry in the volume catalog record |
| 228 (E4) | 40 | CCAEXTDE | Five 8-byte extent descriptors: |
| 228 (E4) | 2 | CCAEXTSS | Sequence number of the Data Space Group set of fields that this extent's space is a part of. |
| 230 (E6) | 4 | CCAEXTAD | The extent's starting physical address: |
| 230 (E6) | 2 | CCAEXTCC | Cylinder number CC |
| 232 (E8) | 2 | CCAEXTHH | Head number HH |
| 234 (EA) | 2 | CCAEXTTH | Number of tracks in the extent |
| 268 (10C) | 1 | CCAASCIK | Number of control intervals required to satisfy the caller's request |
| 269 (10D) | 1 | CCACRRP | RPL used to read the CCR |
| 270 (10E) | 1 | CCAASCIX | Used by the ASSIGN functions—points to the element in CCAASCI currently being processed |
| 271 (10F) | 9 | CCAASCI | Number of each assigned control interval |
| 280 (118) | 16 | CCAEQDQ | ENQ/DEQ parameter list: |
| 280 (118) | 1 | CCAEDXFF | "End of parameter list" indicator—X'FF' |
| 281 (119) | 1 | CCAEDRLN | Length of minor name |
| 282 (11A) | 1 | CCAEDOPT | ENQ/DEQ options |
| | 1... .... | CCAEDSHR | Shared |
| | 0... .... | | Exclusive |
| | .xxx xxxx | | Reserved |
| 283 (11B) | 1 | CCAEDRCD | ENQ/DEQ return code |
| 284 (11C) | 4 | CCADEQNM | Address of the major name |
| 288 (120) | 4 | CCAEDRNM | Address of the minor name |
| 292 (124) | 4 | CCAEDUCB | Address of the UCB |
| 296 (128) | 4 | CCAMLRET | Address of the caller's save area |
| 300 (12C) | 12 | CCAMSSPL | Storage management work area: |
| 300 (12C) | 4 | CCAMNLEN | Number of bytes to process |
| 304 (130) | 4 | CCAMNPTR | Address of the return address |
| 308 (134) | 1 | | Reserved |

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 309 (135) | 1 | CCAMNSPL | Required subpool number |
| 310 (136) | 2 | | Reserved |
| 312 (138) | 4 | CCARPRM | Return parameters |
| 316 (13C) | 8 | CCACMS | Catalog Management Services work area: |
| 316 (13C) | 4 | CCACMSWA | Address of the catalog management services calling routine's work area |
| 320 (140) | 4 | CCAEXCMS | Address of a secondary catalog management services work area |
| *The following fields are set and used by IGGPLOC, IGGPEXT, IGGPMOD, and IGGPTSTS, and catalog management subfunctions that these procedures call:* | | | |
| 324 (144) | 4 | CCACPE5<br>CCALUME | Address of a selected set of fields pointer |
| 328 (148) | 4 | CCACPE51 | (Same as CCACPE5) |
| 332 (14C) | 4 | CCACPE52 | (Same as CCACPE5) |
| 336 (150) | 4 | CCACPE53 | (Same as CCACPE5) |
| 340 (154) | 4 | CCACPE6 | Address of a selected set of fields |
| 344 (158) | 4 | CCACPE61 | (Same as CCACPE6) |
| 348 (15C) | 4 | CCACPE7<br>CCAIDPT | Address of a selected retrieved field<br>(Same as CCACPE7) |
| 352 (160) | 4 | CCACPE71 | (Same as CCACPE7) |
| 356 (164) | 2 | CCAGOPLN | Length of the set-of-fields pointer |
| 358 (166) | 2 | CCASL | Number of bytes for the sequence number |
| 360 (168) | 4 | CCAILNG | Length of the selected retrieved field |
| 364 (16C) | 4 | CCAFLPT<br>CCATFLPT | Address of the requested-field CTGFL<br>Address of the CTGFL-for-tests |
| 368 (170) | 4 | CCARABPT | Address of the record area block |
| 372 (174) | 4 | CCADICT | Dictionary information to describe the field, based on its field name |
| 376 (178) | 4 | CCAXCPL<br>CCAMCPL | Address of the CTGPL built when IGGPEXT and IGGPMOD are called, so that information in the caller's CTGPL is not altered |
| 380 (17C) | 4 | CCARABB | Address of the RAB that identifies the base catalog record |
| 384 (180) | 4 | CCARABF | Address of the RAB that identifies the first record area (buffer) that can be used by IGGPEXT or IGGPMOD |
| 388 (184) | 4 | CCARABL | Address of the RAB that identifies the last record area (buffer) that can be used by IGGPEXT or IGGPMOD |
| 392 (188) | 3 | CCACBASE | The control interval number of the base catalog record |
| 395 (18B) | 1 | CCAGC | Type code of the requested set of fields |
| 396 (18C) | 2 | CCALREL<br>CCALREL1 | Relative repetition number of a selected set of fields |
| 398 (18E) | 2 | CCASN<br>CCASN1 | Sequence number of a selected set of fields |

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 400 (190) | 2 | | Reserved |
| 402 (192) | 2 | CCAIXFPL | Index to the current CTGFL being processed |
| 404 (194) | 2 | CCAIXREL | Index for CCATREL |
| 406 (196) | 2 | CCATNREL | The sequence number of the next set of fields to perform tests against if CCATREL is full or if there are no buffers available to contain the catalog record's next extension |
| 408 (198) | 2 | CCATNUM | Number of successful relative repetition numbers (cannot exceed 16) |
| 410 (19A) | 32 | CCATREL | Successful relative repetition numbers |
| 442 (1BA) | 2 | CCATNO | Total number of successful relative repetition numbers (might exceed 16) |
| 444 (1BC) | 4 | CCATEST | Address of the test CTGFL |
| 448 (1C0) | 20 | CCARBA | Work area for extent descriptors: |
| 448 (1C0) | 6 | CCACRAVL | CRA volume serial number |
| 448 (1C0) | 2 | CCASS | Sequence number of the Data Space Group set of fields that contains the extent |
| 450 (1C2) | 4 | CCACCHH1 | Physical address—CCHH—of the extent's first track |
| 454 (1C6) | 4 | CCACRADT | CRA device type |
| 454 (1C6) | 4 | CCACCHH2 | Physical address—CCHH—of the extent's last track |
| 458 (1CA) | 2 | CCATT | Number of tracks in the extent |
| 460 (1CC) | 4 | CCARBA1 | Low relative byte address (RBA) |
| 464 (1D0) | 4 | CCARBA2 | High relative byte address (RBA) |
| 468 (1D4) | 2 | CCATLNG | The total length of the extent information that has been processed |
| | | CCATLEN | Total length of the scanned field so far |
| 470 (1D6) | 2 | CCARBAL | RBA extent balance |
| 472 (1D8) | 2 | CCACNIX | Combination name index |
| 474 (1DA) | 2 | CCASMFIX | The type of SMF record to be written when a cluster or catalog is defined (SMF record for the cluster, data set, or index) |
| 476 (1DC) | 4 | CCAIDPT2 | Address of the available space in the caller's work area or of the caller-supplied update information |
| 480 (1E0) | 4 | CCAIDPT3 | Address of the length field of a variable-length field in the user's return area |
| 484 (1E4) | 2 | CCAGVCT | Number of set-of-fields pointers processed so far |
| 486 (1E6) | 2 | CCANEVV | If the requested variable-length field is non-existant, this field is set to binary zero |
| 488 (1E8) | 3 | CCAGVEXT | Control interval number of the record's next extension record (not yet in a buffer) |
| 491 (1EB) | 1 | CCANEFV | If the requested fixed-length field is nonexistant,existant, this byte is set to X'FF' |
| 492 (1EC) | 1 | | Reserved |

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 493 (1ED) | 1 | CCAGRGC | Group code of the requested set of fields |
| 493 (1ED) | 1 | CCARCDID | Record ID |
| 494 (1EE) | 2 | CCAGRHI<br>CCAGRHI1 | Sequence number of the requested set of fields |
| 494 (1EE) | 2 | CCASSEQ | Save sequence number |
| 496 (1F0) | 2 | CCAIXTPL | Index to test CTGPLs |
| 498 (1F2) | 2 | CCADLEN | Number of bytes to be deleted from the catalog record |
| 500 (1F4) | 2 | CCADIFF | The difference between the insert length and the delete length (can be a negative number) |
| 502 (1F6) | 2 | CCAREPCT | Number of relative repetition numbers processed so far |
| 504 (1F8) | 2 | CCADISP | Displacement into variable-length field to the delete/insert location |
| 506 (1FA) | 3 | CCASVCI | Save area for the control interval number of the base catalog record |
| 509 (1FD) | 3 | CCASVCI1 | Save area for the control interval number |
| 512 (200) | 4 | CCADTA | Address of the dictionary |
| 516 (204) | 4 | CCACDTA | Address of the index combination table |
| 520 (208)) | 2 | CCADTCT | Number of dictionary entries |
| 522 (20A) | 2 | CCACDTCT | Number of index combination entries |
| 524 (20C) | 4 | CCACWAP | Address of the controller work area |
| 528 (210) | 4 | CCAMNADR | Address of the virtual storage obtained by a GETMAIN request |
| 532 (214) | 4 | CCAILNG3 | Save area for the insertion length |
| 536 (218) | 4 | CCAILNG2 | Length of the user-supplied insert data |
| 540 (21C) | 4 | CCAALPTR | Address of the space management work area |
| 544 (220) | 4 | CCASMFPT | Address of the SMF record chain |
| 548 (224) | 4 | CCALCPL | Address of the CTGPL used when a catalog management routine issues the LSPACE macro instruction |
| 552 (228) | 1 | CCAFLG7 | Flag byte 7: |
| | 1... .... | CCALSP | LSPACE has been called by a catalog management routine |
| | .1.. .... | CCASMFEX | SMF exit indicator |
| | ..1. .... | CCASMFA | Perform SMF alter processing during IGGPEXT and IGGPMOD processing |
| | ...1 .... | CCASMFBR | The base catalog record is in a record area (buffer) for SMF processing |
| | .... 1... | CCAONCE | Move only one set of fields |
| | .... .1.. | CCAROREQ | Request-type is read-only |
| | .... ..1. | CCAFEOV | Force end of volume request |
| | .... ...1 | CCACRABU | A CRA is being built |
| 553 (229) | 1 | CCAFLG8 | Flag byte 8: |
| | 1... .... | CCADSRCL | Recursive call for DEFINE space |
| | .1.. .... | CCAVBUFI | Number of volume records buffered |
| | ..1. .... | CCASCRA | Suppress CRA updates |

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| | ...1 .... | CCASCICK | Suppress CRA control interval check |
| | .... 1... | CCALPIND | Loop control in buffer scan for GETs |
| | .... .1.. | CCAVRIND | Volume record buffer chain is to be checked |
| | .... ..1. | CCALEOD | End-of-file on low keys |
| | .... ...1 | CCAAUCAT | Volume has a user catalog |
| 554 (22A) | 1 | CCAFLG9 | Flags byte 9: |
| | 1... .... | CCARABYC | Bypass catalog I/O |
| | .1.. .... | CCARAEOV | CRA end-of-volume |
| | ..1. .... | CCARALRD | CRA CCR has been read |
| | ...1 .... | CCARACR | CRA CCR checkpoint requested |
| | .... 1... | CCAUCRA | Use CRA CI number translate table before CRA GET |
| | .... .1.. | CCARAACT | CRA is active |
| | .... ..1. | CCARAICI | Inhibit catalog I/O |
| | .... ...1 | CCARESUM | Replace sum |
| | .... ...0 | | Increment sum |
| 555 (22B) | 1 | CCANORBA | Number of RBAs needed |
| 556 (22C) | 4 | CCASMFRD | Address of the SMF record |
| 560 (230) | 2 | CCASMFCT | Number of catalog records in the SMF record chain to be used in creating the SMF ALTER record |
| 562 (232) | 2 | CCASMFLG<br>CCASMFG1 | SMF record flags: |
| *Before calling the SMF-record-writing routine, the caller sets these bits to indicate what it has done or intends to do:* | | | |
| | 1... .... | CCASMFUC<br>CCASMFDF | Uncatalog—write SMF record type 67<br>DEFINE—write SMF record type 63 |
| | .1.. .... | CCASMFSR | Data set, index, or nonVSAM data set has been scratched—write SMF record type 67 |
| | | CCASMFAL | ALTER—write SMF record type 63 |
| | ..xx xxxx | | Reserved |
| 563 (233) | 1 | CCASMFG2 | Reserved |
| 564 (234) | 2 | CCASMFLN | Amount of virtual storage obtained for the SMF record |
| 566 (236) | 3 | CCACHAIN | Control interval number save area |
| 569 (239) | 3 | CCACI1 | Control interval number save area |
| 572 (23C) | 3 | CCACI2 | (Same as CCACI1) |
| 575 (23F) | 3 | CCACI3 | (Same as CCACI1) |
| 578 (242) | 2 | CCAVARLN | Number of bytes to be inserted into the record |
| 580 (244) | 4 | CCARRAB | Address of the RAB containing the set-of-fields pointers where delete/insert processing is to begin. |
| 584 (248) | 4 | CCARBASE | (Same as CCARRAB) |
| 588 (24C) | 4 | CCAVARPT | Address of the information to be inserted into the record |
| 592 (250) | 2 | CCADELN | Number of bytes to be deleted from the record |

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 594 (252) | 20 | CCAVAR | Insert information save area |
| 614 (266) | 20 | CCAVAR1 | (Same as CCAVAR) |
| 634 (27A) | 3 | CCADEL1 | The control interval number of the first record in a series of records to be deleted |
| 637 (27D) | 3 | CCADEL2 | The control interval number of the last record in a series of records to be deleted |
| 640 (280) | 40 | CCAXLATE | Temporary compiler work area |
| 680 (2A8) | 4 | CCAR14S | IGG0CLC9 Register 14 save area |
| 684 (2AC) | 60 | CCABMxxx | Bit manipulation routine (IGG0CLBR) work area |
| 684 (2AC) | 2 | CCABMTRK | Starting track number |
| 686 (2AE) | 2 | CCABMLIM | Ending track number, or upper limit |
| 688 (2B0) | 2 | CCABMMIN | Minimum number of tracks required |
| 690 (2B2) | 1 | CCABMFLG | State and function code: |
| | 1... .... | CCABMST | State to set or condition to check |
| | .1.. .... | CCABMCHK | Perform the check |
| | ..1. .... | CCABMSET | Set the state |
| | ...1 .... | CCABMCCK | Perform a conditional check |
| | .... 1... | CCABMLST | Last set required |
| | .... .xxx | | Reserved |
| 691 (2B3) | 1 | | Reserved |
| 692 (2B4) | 5 | CCABMOUT | Output parameters for the bit manipulation routine (IGG0CLBR) |
| 692 (2B4) | 2 | CCABMONN | Number of tracks |
| 694 (2B6) | 2 | CCABMOTR | Starting track number |
| 696 (2B8) | 1 | CCABMOFG | Output flags: |
| | 1... .... | CCABMOST | State of bits |
| | .xxx xxxx | | Reserved |
| 697 (2B9) | 2 | | Reserved |
| 699 (2BB) | 1 | CCABMPAD | Padding character |
| 700 (2BC) | 4 | CCABMGOP | Address of the current space map set of fields |
| 704 (2C0) | 4 | CCABMPTR | Address of the current bit mask byte |
| 708 (2C4) | 4 | CCABMEND | Address of the end of the current space map set of fields |
| 712 (2C8) | 2 | CCABMBT1 | Number of bits—first byte |
| 714 (2CA) | 2 | CCABMBTL | Number of bits—last byte |
| 716 (2CC) | 2 | CCABMBYT | Number of full bytes |
| 718 (2CE) | 2 | CCABMSTR | Current bit mask starting track |
| 720 (2D0) | 4 | CCABMWK1 | Work area |
| 724 (2D4) | 4 | CCABMWK2 | Work area |
| 728 (2D8) | 4 | CCADMWK3 | Work area |
| 732 (2DC) | 4 | CCABMWK4 | Work area |
| 736 (2E0) | 4 | CCABMRB1 | Address of the first RAB that points to the space map set of fields |

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 740 (2E4) | 4 | CCABMRB2 | Address of the second RAB that points to the space map set of fields |
| 744 (2E8) | 40 | CCATEMPS | Compiler "TEMPS" work area |
| 784 (310) | 348 | CCAREGS | Register 12, 13, and 14 save area (See "Diagnostic Aids" for details about the CCA register save area) |
| 784 (310) | 4 | | Address of user save area |
| 788 (314) | 8 | CCAMODNM | Load module name |
| 1132 (46C) | 4 | CCABZSAV | Address of the save area for IGG0CLBZ |
| 1132 (46C) | 4 | CCADSPWA | Address of DEFINE SPACE work area |
| 1136 (470) | 4 | | Reserved |
| 1140 (474) | 4 | CCACUMPL | Address of the catalog management upgrade parameter list |
| 1144 (478) | 1 | | Reserved |
| 1145 (479) | 3 | CCASBASE | Save base control interval for upgrade process |
| 1148 (47C) | 4 | CCACRACI | Address of the CRA record pointer array |
| 1152 (480) | 4 | CCARAACB | Address of the CRA's ACB |
| 1156 (484) | 4 | CCARARPL | Address of the CRA RPL |
| 1160 (488) | 4 | CCARARBA | Relative byte address of the CRA |
| 1164 (48C) | 4 | CCARAREC | Address of a CRA record |
| 1168 (490) | 4 | CCARALSA | Address of the CRA local save area |
| 1168 (490) | 2 | CCACRABT | Block/track value for CRA record construction |
| 1170 (492) | 2 | | Reserved |
| 1172 (494) | 1 | CCAFLG10 | Flag byte 10: |
| | 1... .... | CCAINCPL | Catalog parameter list is invalid |
| | .1.. .... | CCAPDMW | Problem determination message |
| | ..1. .... | CCACATAC | Catalog is active |
| | ...1 .... | CCARAFEV | Forced end of volume |
| | .... 1... | CCARARTC | Recovery exit - return to caller |
| | .... .1.. | CCAPRANX | Prime CRA gone |
| | .... ..xx | | Reserved |
| 1173 (495) | 3 | CCASUMTT | Sum of the tracks in the CRA |
| 1176 (498) | 4 | CCADICTS | Data/index timestamp |
| 1180 (49C) | 8 | CCAPANCA | Start, end addresses of normal record buffer chain |
| 1188 (4A4) | 8 | CCARAVCA | Start, end addresses of volume record buffer chain |
| 1196 (4AC) | 8 | CCAVTS | Volume timestamp |
| 1204 (4B4) | 4 | CCAREWKA | Address of reusable data set processing workarea |
| 1208 (4B8) | 4 | CCASMFP | Save area for SMF problem determination |
| 1208 (4B8) | 2 | CCASMFMD | Module ID |
| 1210 (4BA) | 1 | CCASMFRC | Reason code |
| 1211 (4BB) | 1 | CCASMFCD | Return code |

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 1212 (4BC) | 4 | CCAPROBX | Save area for CCAPROB |
| 1212 (4BC) | 2 | CCAMODDX | Error module ID |
| 1214 (4BE) | 2 | CCAERCDX | Error codes |
| 1214 (4BE) | 1 | CCARESNX | Reason code |
| 1215 (4BF) | 1 | CCARETRX | Return code |

## *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 Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 4 | CLWCOMWK | Address of common work area |
| 4 (4) | 4 | CLWAMBPT | Address of current AMB |
| 8 (8) | 12 | CLWSFI | Subfunction information area |
| 20 (14) | 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."

| 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 |

**Cluster Management Block (CMB)—Description and Format**

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 4 (4) | 1 | CMBFLGS | Flags: |
| | 1... .... 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 |
| 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 |

## *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 module listing IDA019R3 for channel program building and execution details. The formats of four channel programs follow this description of the CPA.

| 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 |

## Channel Program Area (CPA)—Description and Format

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 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 |
| 39 (27) | 4 | CPAWCHR | Cylinder and head address (CCHH value) |
| 43 (2B) | 1 | | 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 | CPARCHR | Cylinder and head address (CCHH value) |
| 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 |
| | .xxxxxxx | | 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 | CPANXTI | Address of the static CPA chain |
| 92 (5C) | 4 | CPACPCHN | Address of the dynamic CPA chain |

## *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.

| CCW Number | Command Code Hex | Description | Address | Flags 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] CPABLKSZ | 06[3] | Read data | IDAL | 40 | CC | |
| | 86[4] CPABLKSZ | M-T read data | IDAL | 40 | CC | |
| 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).

| CCW Number | Command Code Hex | Description | Address | Flags 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] | 1D | Write C,K,&D | CPAFWCNT | 80 | CC | 8 |
| FW6[2] | 1D | 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).

| CCW Number | Command Code Hex | Description | Address | Flags 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.

| CCW Number | Command Code Hex | Description | Address | Flags Hex | Description | Count |
|---|---|---|---|---|---|---|
| WC1 | 1B | Seek head | CPAWSEEK | 40 | CC | 6 |
| WC2 | 23[1] | Set sector | CPAWSECT | 60 | CC, SLI | 1 |
| WC3 | 31 | Search ID eq. | CPAFWCTN[2]<br>CPAWSID[3] | 40 | CC | 5 |
| 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 pointed to by OPWA (called the ACB work area). Additional CSLs are chained as required.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| **CSL Header** | | | |
| 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: 'bIDACSLb' |
| 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) | 12×32 | 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 header element in the HEB for the virtual-storage area, or zero |

## *CTGCV—VSAM Catalog Control Volume List*

The CTGCV is built by the Scheduler to contain the volume and name of the OS/VS system catalog CVOL entry for a SUPERLOCATE request. CTGCV is mapped by IEZCTGCV and is pointed to by CTGPL.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 6 | CTGCVVOL | CVOL volume serial |
| 6 (6) | 4 | CTGCVDEV | CVOL device type |

## *CTGFL—Field Parameter List*

The CTGFL is built before an OS/VS component issues the CATLG macro instruction (SVC 26) to process a catalog record. The CTGFL defines one of the catalog record's fields or a group of logically related fields (identified by a combination name). The CTGFL is used in two situations:

- It identifies catalog record information to retrieve or update. The CTGPL contains the address of each CTGFL used in this way.
- It identifies catalog record information to compare against caller-supplied data. This is a "test" CTGFL and is addressed by another CTGFL.

When a catalog management routine is processing a CTGFL, the CTGFL's address is in the CCA (CCAFLPT or CCATEST).

**CTGFL—Field Parameter List**

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 1 | CTGFLDNO | Number of entries in CTGFLDAT |
| 1 (1) | 1 | CTGFLDCD | Test condition: |
| | X'00' | | The CTGFL describes a field to be updated or retrieved. The CTGFL is pointed to by the caller's CTGPL (CTGFIELD entry). |
| | X'non00' | | The CTGFL describes a test condition. The CTGFL is pointed to by another CTGFL. |
| | | | Test condition: |
| | X'80' | | Equal |
| | X'70' | | Not equal |
| | X'20' | | Greater than |
| | X'40' | | Less than |
| | X'A0' | | Greater than or equal |
| | X'C0' | | Less than or equal |
| | X'80' | | Test under mask for zeros |
| | X'10' | | Test under mask for ones |
| | X'40' | | Test under mask for mixed |
| 2 (2) | 1 | CTGFLDGC | Type code number |
| 3 (3) | 1 | CTGFLDRE | Test results: |
| | xxxx xxx. | | Reserved |
| | .... ...0 | | Successful test |
| | .... ...1 | | Test failed |
| 4 (4) | 4 | CTGFLDWA | Work area: contains information about the catalog record's field name from the dictionary |
| 8 (8) | 4 | CTGFLDNM | Address of the field name |
| 12 (C) | 4 | CTGFLCHN | Address of a CTGFL-for-tests, or 0 |
| 16 (10) | (CTGFLDNO × 8) | CTGFLDAT | Address and length, in the caller's work area, of:<br>• Each field that was retrieved, if the request was LOCATE or LISTCAT.<br>• New data to replace or add to data in the catalog record, if the request was UPDATE, DEFINE, or ALTER.<br>• Data used to compare to catalog record fields, if the CTGFL is a CTGFL-for-tests. |

## *CTGFV—Field Vector Table*

The CTGFV is built by the Access Method Services (AMS) utility programs and contains addresses of user-supplied information fields and lists. The CTGFV is built when the user issues a DEFINE or ALTER command. If the user is creating a cluster, a CTGFV is built for each catalog record that will be built to describe the cluster: that is, Access Method Services builds a cluster CTGFV, a data CTGFV, and, if the cluster is key-sequenced, an index CTGFV. The CTGFV is pointed to by the CTGPL (CTGFVT). If Access Method Services builds more than one CTGFV, the cluster CTGFV is pointed to by the CTGPL (CTGFVT) and the data and index CTGFVs are pointed to by the cluster CTGFV.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 1 | CTGFVTYP | The CTGFV contains information used by the catalog management service's DEFINE routines to build a catalog record of the type: |
| | C'A' | | NonVSAM |
| | C'C' | | Cluster |
| | C'D' | | Data |
| | C'G' | CTGFVAIX | Alternate index |
| | C'I' | | Index |
| | C'R' | CTGFVPTH | Path |
| | C'V' | | Volume |
| 1 (1) | 1 | CTGFVPRO | Catalog management services processing option flags: |
| | 1... .... | CTGFVAVL | ALTER: Add volumes |
| | .1.. .... | CTGFVRVL | ALTER: Remove volumes |
| | ..1. .... | CTGFVNDC | No device-type conversion |
| | ...1 .... | CTGFVDRC | DEFINE a recoverable catalog |
| | .... xxxx | | Reserved |
| 2 (2) | 1 | CTGFVELM | Element number of CMSPCATR |
| 3 (3) | 1 | | Reserved |
| 4 (4) | 4 | CTGFVDCH | Address of the cluster's data CTGFV |
| 8 (8) | 4 | CTGFVICH | Address of the cluster's index CTGFV |
| 12 (C) | 4 | CTGFVVCH | Address of the space vector table |
| 16 (10) | 4 | CTGFVIND | Address of the associated DD statement |
| 20 (14) | 4 | CTGFVENT | Address of the entry name CTGFL |
| 24 (18) | 4 | CTGFVSTY | Address of the security information CTGFL (passwords, codeword, and number of tries) |
| 28 (1C) | 4 | CTGFVOWN | Address of the owner identification CTGFL |
| 32 (20) | 4 | CTGFVEXP | Address of the expiration date CTGFL |
| 36 (24) | 4 | CTGFVCRE | Address of the creation date CTGFL |
| 40 (28) | 4 | CTGFVVLT | Address of the volume serial number list |
| 44 (2C) | 4 | CTGFVRNG | Address of the key range list |
| 48 (30) | 4 | CTGFVDVT | Address of the device type CTGFL (for NonVSAM DEFINE only) |
| 52 (34) | 4 | CTGFVSPC | Address of the space allocation information CTGFL |
| 56 (38) | 4 | CTGFVAMD | Address of the AMDSB CTGFL (if VSAM DEFINE) |
| | | CTGFVFSN | Address of the file sequence number (if NonVSAM DEFINE) |

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 60 (3C) | 4 | CTGFVATR | Address of the data set attributes CTGFL |
| 64 (40) | 4 | CTGFVBUF | Address of the buffer size CTGFL |
| 68 (44) | 4 | CTGFVLRS | Address of the average record size CTGFL |
| 72 (48) | 4 | CTGFVEXT | Address of exception exit parameter list |
| 76 (4C) | 4 | CTGFVUPG | Address of alternate index/path parameter list |
| 80 (50) | 4 | CTGFVNAM | Address of the related name |
| 84 (54) | 4 | CTGFVPWD | Address of related object's password |
| 88 (58) | 4 | CTGFVWKA | Address of the CRA feedback area |

## *CTGPL—Catalog Parameter List*

The CTGPL is built before an OS/VS component issues the CATLG macro instruction (SVC 26) to process a catalog record. The CTGPL defines the catalog management request and its options, the catalog record to be processed, and the VSAM catalog that contains the record. The CTGPL is pointed to by register 1. When the catalog management routines build a CCA to support the request, the address of the CTGPL is put into the CCA (CCACPL).

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 1 | CTGOPTN1 | First option byte: |
| | 1... .... | CTGBYPSS | Bypass the catalog management security verification processing |
| | .1.. .... | CTGMAST | Check the master password |
| | ..1. .... | CTGCI | Check the control interval password |
| | ...1 .... | CTGUPD | Check the update password |
| | .... 1... | CTGREAD | Check the read password |
| | .... .1.. | CTGNAME | The CTGENT field contains the address of a 44-byte dsname, or a 6-byte volume serial number (padded with binary 0s) |
| | .... .0.. | | The CTGENT field contains the address of a 3-byte control interval number |
| | .... ..1. | CTGCNAME | The CTGCAT field contains the address of a 44-byte catalog dsname or the address of an OS/VS system catalog CVOL entry |
| | .... ..0. | | The CTGCAT field contains the address of a 4-byte field that contains the address of a VSAM catalog's ACB |
| | .... ...x | | Reserved |
| 1 (1) | 1 | CTGOPTN2 | Second option byte: |
| | 1... .... | CTGEXT | Extend option (with UPDATE) |
| | .1.. .... | CTGERASE | Erase option (with DELETE) |
| | | CTGSMF | Write SMF record option (with LSPACE) |
| | ..1. .... | CTGPURG | Purge option (with DELETE) |
| | | CTGVMNT | The caller is VSAM Open/Close/EOV: Volume Mount and Verify routine (IDA0192V) |
| | ...1 .... | CTGGTNXT | Get-next option (with LISTCAT |
| | .... 1... | CTGDISC | Disconnect option (with DELETE) |
| | .... .1.. | CTGOVRID | Erase override option (with DELETE) |
| | .... ..1. | CTGSCR | Scratch space option (with DELETE) |
| | .... ...x | | Reserved |

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 2 (2) | 1 | CTGOPTN3 | Third option byte: |
| | xxx. .... | CTGFUNC | Specifies the caller-requested function: |
| | 001. .... | | LOCATE |
| | 010. .... | | LSPACE |
| | 011. .... | | UPDATE |
| | 100. .... | | A Catalog Management Services function (see CTGOPTNS) |
| | ...1 .... | CTGSUPLT | Super-Locate function |
| | .... x... | | Reserved |
| | .... .1.. | CTGSRH | Search the OS/VS system catalog first |
| | .... .0.. | | Search the VSAM catalog first (specified by CTGCAT or, if CTGCAT = 0, search the VSAM catalogs available to the caller) |
| | .... ..1. | CTGNUM | Search only one catalog (specified by CTGCAT) |
| | .... ..0. | | Search as many catalogs as are available |
| | .... ...1 | CTGAM0 | The call is a VSAM catalog management request |
| | .... ...0 | | The call is a nonVSAM request |
| 3 (3) | 1 | CTGOPTN4 | Fourth option byte: |
| | xxxx .xxx | | Reserved |
| | .... 1... | CTGBYPMT | Bypass security prompting to system operator |
| 4 (4) | 4 | CTGENT | Address of the catalog record identifier, as defined in CTGOPTN1. |
| | | CTGFVT | Address of the caller's CTGFV |
| 8 (8) | 4 | CTGCAT | Address of the catalog's dsname or a 4-byte field that contains the address of the specified in CTGOPTN1 |
| | | CTGCVOL | Address of an OS/VS system catalog CVOL entry, if the request is SUPERLOCATE |
| 12 (C) | 4 | CTGWKA | Address of the caller's work area |
| 16 (10) | 2 | CTGDSORG | Data set organization, if the request is SUPERLOCATE |
| 16 (10) | 1 | CTGOPTNS | Catalog Management Services request options: |
| | 0000 1... | | DEFINE |
| | 0001 0... | | ALTER |
| | 0001 1... | | DELETE |
| | 0010 0... | | LISTCAT |
| | 0011 0... | | CONVERTV |
| | .... .xxx | | Reserved |
| 17 (11) | 1 | | Reserved |

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 18 (12) | 1 | CTGTYPE | Type of catalog record: |
| | C'D' | | Data |
| | C'I' | | Index |
| | C'A' | | NonVSAM |
| | C'U' | | User catalog |
| | C'V' | | Volume |
| | C'C' | | Cluster |
| | C'M' | | Master catalog |
| | C'G' | | Alternate index |
| | C'R' | | Path |
| | C'Y' | | Upgrade |
| | C'F' | | Free |
| 19 (13) | 1 | CTGNOFLD | Number of entries contained in CTGFIELD |
| 20 (14) | 4 | CTGDDNM | Address of the DD statement, if one is associated with this request |
| | | CTGNEWNM | Address of the new dsname, if the request is ALTER and the object's name is being changed |
| | | | If the request is Super-Locate: |
| 20 (14) | 2 | CTGFDBK | Feedback area |
| 22 (16) | 1 | CTGFBFLG | Flags: |
| | 1... .... | CTGPAR | Parallel mount |
| | .1.. .... | CTGKEEP | Forced keep |
| | ..xx xxxx | | Reserved |
| 23 (17) | 1 | | Reserved |
| 24 (18) | 4 | CTGJSCB | Address of the JSCB |
| | | CTGPSWD | Address of the caller-supplied password |
| 28 (1C) | VL | CTGFIELD | The address of each CTGFL, to specify each catalog field to be processed. The length of CTGFIELD is four times the CTGNOFLD value. |

## *CTGVL—Volume List*

The CTGVL is built by the issuer of a locate request for a data-set name. Catalog management uses the CTGVL to return to the caller the volume serial numbers of the volumes on which space is allocated to the data set. For superlocate requests, the CTGWA points to the CTGVL.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 6 | CTGVLVOL | Volume serial number |
| 6 (6) | 4 | CTGVLDEV | Device type |
| 10 (A) | 2 | CTGVLSEQ | File sequence number |
| 12 (C) | 3 | CTGVLTTR | For a single-volume data set, the TTR of its DSCB |
| 12 (C) | VL | | For a multi-volume data set, a repetition of CTGVLVOL, CTGVLDEV and CTGVLSEQ for the rest of the volumes |

## *CTGWA—Work Area*

The CTGWA is built by the caller of catalog management for most requests. The CTGPL points to the CTGWA.

The work area has one format for a superlocate request and another format for all other requests.

**Format for a Request Other Than Superlocate**

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 2 | CTGWALNG | Length of the work area |
| 2 (2) | VL | * | Data returned for DEFINE, DELETE, GENDSP, LISTCAT, LOCATE, LSPACE |

**Format for a Superlocate Request**

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 4 | CTGWAVL | Address of the CTGVL (volume list) |
| 4 (4) | 2 | CTGWALV | Length of the volume list |
| 6 (6) | 2 | CTGWAVCT | Number of volume serial numbers returned in the volume list |
| 8 (8) | 2 | CTGWAUCT | Minimum number of volumes that must be mounted |

## *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).

| 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: |
| | 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 |

**Data Insert Work Area (DIWA)—Description and Format**

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| | .xxxxxxx | | Reserved |
| 6 (6) | 2 | | Reserved |
| 8 (8) | 4 | DIWLRBA | Address of the first control interval in a control area which is being split |
| 12 (C) | 4 | DIWHRBA | Address of the last control interval in a control area which 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 which 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 |

## *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 pointed to by OPWA (called the ACB work area). Additional DSLs are chained as required.

| 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: 'bIDADSLb' |
| 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) | 4x16 | 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).

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| **EDB Header Definition** | | | |
| 0 (0) | 1 | EDBIDID | Control block identifier, X'90' |
| 1 (1) | 1 | EDBNO | Number of EDB entries—one EDB per logical extent |
| 2 (2) | 2 | EDBLEN | Length of an EDB entry in bytes |
| 4 (4) | 4 | EDBLPMBC | Address of first LPMB |
| **EDB Entry Definition** | | | |
| 0 (0) | 3 | | Reserved |
| 3 (3) | 1 | EDBM | Extent number; specifies the relative location of an extent entry in a DEB |
| 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 |

## *ESL—Enqueue Save List*

The ESL contains up to 16 entries that describe ENQ requests that have been completed during Open. It enables Open to dequeue the indicated resources if an error prevents them from being dequeued normally.

The ESL is pointed to by OPWA (called the ACB work area). Additional ESLs are chained as required.

| 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: 'bIDAESLb' |
| 12 (C) | 4 | ESLNXPTR | Address of the next ESL (zero for the last ESL in the chain) |
| 16 (10) | 2 | ESLACTEN | Number of active entries |
| 18 (12) | 2 | | Reserved |
| 10 (14) | 9 x 16 | ESLENTRY | Entries for resources enqueued: |
| 20 (14) | 1 | ESLENQOP | The ENQ option that was used for this resource |
| 21 (15) | 8 | ESLRNAME | ENQ resource name (minor) that identifies this resource: |
| 21 (15) | 3 | ESLCINBR | Control-interval number for the resource |

**ESL—Enqueue Save List**

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 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:<br>I input<br>0 output |

## *EXLST—Exit List*

The EXLST contains addresses for the user-exit processing routines EODAD, SYNAD, LERAD, and JRNAD. The address of the EXLST is in the ACB (ACBEXLST).

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 1 | EXLID | Control block identifier, X'81' |
| 1 (1) | 1 | EXLSTYP | Subtype identifier:<br>X'10' = VSAM<br>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) | 1 | EXLUPADF | Entry description |
| 21 (15) | 4 | EXLUPADP | Address of the UPAD exit routine |
| 25 (19) | 5 | | 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 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.

| 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 in use |
| 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 |
| 1 (1) | 3 | HEBLN | Length of the storage block |
| 4 (4) | 4 | HEBBLKPT | Address of the storage block |
| 8 (8) | 1 | HEBFLAGS | Flags: |
| | 1... .... | HEBPGBDY | The storage block is on a page boundary |
| | .xxx xxxx | | 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 ICWA is pointed to by the index AMB (AMBIWA).

| Offset | Bytes and 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 work area |
| | ..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 |

## Index Create Work Area (ICWA)—Description and Format

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 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 |
| 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 (3B) | 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.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 1 | IICBID | IICB 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 work area (scan mode). |
| 16 (10) | 2 | IISAVLRL | Previous DCBLRECL value (load mode). |
| 18 (12) | 2 | IIMAXLRL | Maximum DCBLRECL value (load mode). |
| 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. |
| | .... .1.. | IIFTEST | Loop test bit. |
| | .... ..x. | | Reserved |
| | .... ...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. |
| | .xxxxxxx | | Reserved. |
| 37 (25) | 3 | IIMSGL | Message area length. |

### ISAM Interface Control Block (IICB)—Description and Format

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 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. |

## *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).

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 1 | IMWID | IMWA 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. |
| | ..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 control block description). |
| 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. |

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 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. |

## *IOB Extension to Support VSAM Processing*

The VSAM extension to the basic direct-access IOB (offset is 40 bytes (X'28') from the start of the IOB) contains information used by VSAM I/O Management for OS/VS I/O Supervisor to process VSAM I/O requests.

See *OS/VS1 System Data Areas* for basic direct-access IOB details.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| **Note:** These fields are located in the IOB at X'28', + X 'offset value'. | | | |
| 0 (0) | 4 | IOBBUFCS | Address of the BUFC for current I/O processing |
| 4 (4) | 1 | IOBIOMF | Flags: |
| | 1... .... | IOBAMUSE | The IOB is in use |
| | .1.. .... | IOBMCSW | The address in the CSW is not in a VSAM channel program |
| | ..1. .... | IOBEOVW | The VSAM End of Volume routine is waiting |
| | ...1 .... | IOBEOVTS | The VSAM End of Volume routine set the IOBLOCK field |
| | .... xxxx | | Reserved |
| 5 (5) | 1 | IOBNMOD | Number of modules in the virtual storage list |
| 6 (6) | 1 | IOBLOCK | Process lock |
| 7 (7) | 1 | | Reserved |
| 8 (8) | 2 | IOBNBUF | Number of buffers reserved for this request |
| 10 (A) | 2 | IOBNSEG | Number of channel program segments for this request |
| 12 (C) | 4 | IOBPLH | Address of the PLH |
| 16 (10) | 4 | IOBR14 | Save area that contains the return address to the I/O Supervisor when an appendage module is called |
| 20 (14) | 4 | IOBVSL | Address of the virtual storage list |

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 24 (18) | 24 | IOBMSAV1 through IOBMSAV6 | Six 4-byte work areas |
| 48 (30) | 4 | REG1 | Register save area |
| 52 (34) | 4 | IOBIQE | Address of the IQE |

## *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).

| 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 |
| 14 (E) | 3 | | Reserved |
| 17 (11) | 1 | IXSBFLG | Flags: |
| | 1... .... | IXSSSRH | Used by the Search Compressed Index Search routine: 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 preceeding 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 table 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, RPL, and EXLST), 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) |
| 9 (9) | 1 | The index point for MODCB of the NIB |
| 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) |
| • | | |
| • | | |
| • | | |
| 255 (FF) | 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, using information in the data set's catalog record. The EDB (EDBLPMBA) contains the address of the LPMB.

| Offset | Bytes and 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... | LPMBSSTH | The Set Sector table is included at the end of the LPMB |
| | ...x .xxx | | Reserved |
| 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 component, this field is the control area 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 |

## *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 |

**Open Work Area (OPW)—Description and Format**

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 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 |
| | .... xxx. | | Reserved |
| | .... ...1 | OPWNOPFL | Page fix list does not exist |
| 14 (E) | 1 | OPWFLGS3 | Flags for IDA0192B: |
| | 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 |
| | .... xxxx | | Reserved |
| 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 | OPWCURPT | Address of cluster being processed. This field can point to OPWBSECL (548(224)), OPWPTAIX (556(22C)), OPWUPAIX (568(238)), and every 8 bytes thereafter, since OPWUPAIX is a repeating field. The format of current cluster information is described below by OPWCURCL. |
| 20 (14) | 4 | OPW2YPLH | Address of first PLH |
| 24 (18) | 4 | OPWCAMBL | The address of the existing AMBL for connecting to an existing structure |
| 28 (1C) | 4 | OPWBCON | Address of base AMBL connecting to |
| 32 (20) | 4 | OPWPCON | Address of path AMBL connecting to |
| 36 (24) | 4 | OPWBAMBL | Address of AMBL for base |
| 40 (28) | 4 | OPWPAMBL | Address of AMBL for path |
| 44 (2C) | 6 | OPWCRA | CRA volume serial number |
| 50 (32) | 2 | | Reserved |
| 52 (34) | 4 | OPWBIB | Address of the BIB |
| 56 (38) | 4 | OPWUPT | Address of the upgrade table |
| 60 (3C) | 4 | OPWUACB | Address of the user ACB |
| 64 (40) | 20 | OPWSAVE | Addresses of save lists |
| 64 (40) | 4 | OPWCSL | Address of core save list |
| 68 (44) | 4 | OPWESL | Address of ENQ save list |
| 72 (48) | 4 | OPWPSL | Address of the page-fix save list |
| 76 (4C) | 4 | OPWDSL | Address of the DEB save list |
| 80 (50) | 4 | OPWSSL | Address of the swap save list |
| 84 (54) | 4 | OPWWRKPT | Address of current AMB work area |
| 88 (58) | 4 | OPWDTWRK | Address of data AMB work area |
| 92 (5C) | 4 | OPWIXWRK | Address of index AMB work area |
| 96 (60) | 4 | OPWCTCB | Address of current TCB |

**Open Work Area (OPW)—Description and Format**

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 100 (64) | 4 | OPWJSTCB | Address of job step TCB |
| 104 (68) | 4 | OPWTIOT | Address of TIOT entry |
| 108 (6C) | 4 | OPWCOMWA | Address of Open common work area |
| 112 (70) | 4 | OPWBUFND | Number of data buffers |
| 116 (74) | 4 | OPWBUFNI | Number of index buffers |
| 120 (78) | 1 | OPWCSTRN | Current string number |
| 121 (79) | 1 | OPWSTRNO | Path string number, if path processing; otherwise, base string number |
| 122 (7A) | 1 | OPWBSTRN | Base string number, if base processing |
| 123 (7B) | 1 | | Reserved |
| 124 (7C) | 8 | OPWIDF | Cluster identifier |
| 124 (7C) | 4 | OPWCACB | Address of catalog ACB |
| 128 (80) | 3 | OPWDCI | Control interval number of data component |
| 131 (83) | 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 |
| 132 (84) | 16 | OPWVSMPL | O/C/EOV Virtual Storage Manager parameter list |
| 132 (84) | 4 | OPWVMANC | Address of anchor block |
| 136 (88) | 1 | OPWVMSP | Subpool for direct request |
| 136 (88) | 1 | OPWVMTYP | Request type |
| 137 (89) | 3 | OPWVMLNG | Amount of storage requested |
| 140 (8C) | 4 | OPWVMADR | Address of storage acquired (zero, if storage not obtained) |
| 144 (90) | 1 | OPWVMFLG | Flag byte: |
| | 1... .... | OPWVMPGB | Get storage on a page boundary |
| | .1.. .... | OPWVMNSL | Do not build a CSL for this request |
| | ..1. .... | OPWVMDIR | Direct request |
| | | | Reserved |
| 145 (91) | 3 | | Reserved |
| 148 (94) | 76 | OPWVSMWA | O/C/EOV Virtual Storage Manager work area |
| 148 (94) | 4 | OPWVANCP | Pointer to the address of the first HEB header element associated with this request |
| 152 (98) | 4 | OPWVTBLP | Address of the request table used by GETSPACE routine |
| 156 (9C) | 4 | OPWVCSLE | Address of core save list entry |
| 160 (A0) | 4 | OPWVHDRE | Address of header element |
| 164 (A4) | 16 | OPWVSAVE | Save area for IDA0192M processing |

**Open Work Area (OPW)—Description and Format**

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 180 (B4) | 36 | | The following 12-byte field is repeated 3 times: |
| 0 (0) | 12 | OPWVGSPL | Get space parameter lists |
| 0 (0) | 1 | OPWVGSSP | Subpool number |
| 1 (1) | 3 | OPWVGETL | Length of acquired storage |
| 4 (4) | 4 | OPWVGSPT | Address of required storage |
| 8 (8) | 1 | OPWVGFLG | Flags for Get space |
| | 1... .... | OPWVGPGB | Get storage on a page boundary |
| | .1.. .... | OPWVGNSL | Do not build a CSL for this request |
| | ..xx xxxx | | Reserved |
| 9 (9) | 3 | OPWVREQL | Length of request |
| 216 (D8) | 12 | OPWVGMPL | GETMAIN parameter list |
| 228 (E4) | 8 | OPWVFMPL | FREEMAIN parameter list |
| 228 (E4) | 1 | OPWVFMSP | FREEMAIN subpool number |
| 229 (E5) | 3 | OPWVFMLN | FREEMAIN length |
| 232 (E8) | 4 | OPWVFMPT | FREEMAIN address |
| 236 (EC) | 52 | OPWDACB | Dummy ACB for opening base |
| 288 (120) | 12 | OPWSFI | Subfunction information |
| 300 (12C) | 256 | OPWERMAP | Map of return codes to ACBERFLG, where return code *rc* is defined in *OS/VS Message Library: VS1 System Messages* for messages IEC070I, IEC161I, IEC251I, and IEC252I. |
| 556 (22C) | 4 | OPWSAVEA | Return address save area |
| 560 (230) | 8 | OPWBSECL | Base cluster information |
| 560 (230) | 1 | | Reserved |
| 561 (231) | 3 | OPWBDTCI | Base data control interval number |
| 564 (234) | 1 | | Reserved |
| 565 (235) | 3 | OPWBIXCI | Base index control interval number |
| 568 (238) | 8 | OPWPTAIX | Path alternate index information |
| 568 (238) | 1 | | Reserved |
| 569 (239) | 3 | OPWPDTCI | Path alternate index data control interval number |
| 572 (23C) | 1 | | Reserved |
| 573 (23D) | 3 | OPWPIXCI | Path alternate index control interval number |
| 576 (240) | 1 | OPWNOUPG | Number of upgrade alternate indexes |
| 577 (241) | 3 | | Reserved |
| 580 (244) | 8 | | The following 8-byte field, pointed to by OPWCURPT (16(10)), 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 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 (16(10)) 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.. .... | OPWPATII | 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 | OPWCDTCI | 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 |

## *PCCB—Private Catalog Control Block*

A PCCB describes each user's catalog to the OS/VS system. The PCCB is built when the user's catalog is opened. If the catalog is already open for another user, a PCCB is built when the user's JCL DD STEPCAT or JOBCAT statement specifies the catalog. The JSCB associated with the user task's TCB points to the first PCCB. Other PCCBs available to the user's task are chained form the first PCCB.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 4 | PCCBNXT | Address of the next PCCB |
| 4 (4) | 4 | PCCBACB | Address of the ACB for the VSAM user's catalog |
| 8 (8) | 4 | PCCBTCE | Address of the TIOT chaining element |
| 12 (C) | 4 | PCCBCORE | Information used with GETMAIN and FREEMAIN requests: |
| 12 (C) | 1 | PCCBSBP | Subpool number |
| 13 (D) | 3 | PCCBLNG | Amount of virtual storage for the PCCB, TCE, and AT |
| 16 (10) | 4 | PCCBSTS | Status bytes |
| 16 (10) | 1 | PCCBST1 | Status Flags: |
| | 1... .... | PCCBOPN | The VSAM user's catalog was opened |
| | .xxx xxxx | | Reserved |
| 17 (11) | 3 | | Reserved |

## *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. When a record management routine is processiug a PLH, the PLH's address is in register 2 (RPLH).

| 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 |
| 2 (2) | 1 | PLHFLG1 | Process flag 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 | PLHFLG2 | Process flag 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 | Test whether ECB is posted |
| 4 (4) | 2 | PLHEFLGS | Exception flags: |
| | | | Byte 1: |
| | 1..... | PLHNOSPC | End of Volume could find no more space for creation |
| | .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 | The PLH is invalid |
| | .......1 | PLHDSCAN | Scan data after read exclusive |

**Placeholder (PLH)—Description and Format**

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| | | | 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 |
| | .....1.. | PLHEOVDF | End of Volume synchronization flag |
| | ......xx | | 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 |
| 8 (8) | 4 | PLHACB | Address of the caller's ACB |
| 12 (C) | 1 | PLHDSTYP | Data set type: |
| | | | Data—X'01'<br>Index—X'02' |
| 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<br>PLHJORBA | Current RBA<br>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. See *VSAM Programmer's Guide* for a list of entry codes. |
| 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 |

**Placeholder (PLH)—Description and Format**

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 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<br>PLHIIOB | Address of the data IOB<br>Address of the index IOB |
| 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 save area for IDADRQ and IDATJXIT |
| 112(70) | 4 | PLHCHAIN | Address of the next PLH in the chain |
| 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<br>PLHDDDD | Address of the RBA provided by the End of Volume routine<br>RBA of the previous request |
| 184 (B8) | 4 | PLHNRBA | Next RBA |
| 188 (BC) | 4 | PLHIBUFC | Address of the index BUFC |
| 192 (C0) | 4 | PLHRBUFC | Save area for register RBUFC for IDADRA and IDATJXIT |
| 196 (C4) | 4 | PLHISPLP | Address of the IXSPL |
| 200 (C8) | 32 | PLHIXSPL | Space for one IXSPL |
| 200 (C8) | 4 | PLHSSRBA<br>PLHHIREC | RBA of the sequence-set control interval<br>RBA of the highest record |
| 204 (CC) | 4 | PLHIXBFC | Address of a BUFC for index search |
| 208 (D0) | 24 | | |
| 232(E8) | 4 | PLHWAX<br>PLHXPLH | Address of the work area for path processing<br>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 |

**Placeholder (PLH)—Description and Format**

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 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 | 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 | | Reserved |
| 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 |

## *RPL—Request Parameter List*

The RPL contains user-request information and error feedback information. It also maintains information required by GET and PUT macro instructions. The RPL is created by the user with the RPL macro instruction.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 4 | RPLIDWD | Identification word of the RPL: |
| 0 (0) | 1 | RPLID | RPL identifier, X'00' |
| 1 (1) | 1 | RPLSTYP | RPL subtype: |
| | | | X'10' = VSAM<br>X'20' = VTAM |
| 2 (2) | 1 | RPLREQ | Request type—when the user issues a VSAM macro-instruction, 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) | | Search buffer request |
| | 14 (E) | | Mark buffer request |
| | 15 (F) | | Write buffer request |
| 3 (3) | 1 | RPLLEN<br>RPLLEN2 | 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 word |
| 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<br>RPLERREG | RPL return code (See "Error Codes" in "Diagnostic Aids" section for a description of RPL return codes.) |
| | X'00' | | Normal return |

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| | X'04' | | Request not accepted because the RPL indicated for this request was active for another request |
| | 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<br>RPLKEYL | Key length |
| 18 (12) | 2 | RPLSTRID | RPL transaction 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 |
| 36 (24) | 4 | | Address of SETPRT parameter list: |
| 36 (24) | 2 | RPLSAF | Source address field |
| 38 (26) | 2 | RPLDAF | Destination address field |
| 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<br>RPLECBIN | 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<br>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 |
| | ...0 .... | | Process in forward direction |
| | .... 1... | RPLLRD | Locate or retrieve the last record in the data set |
| | .... 0... | | Locate, retrieve, or store the record identified by the user's argument |
| | .... .1.. | RPLWAITX | Take UPAD exit before WAIT |
| | .... ..1. | RPLUPD | Update processing |
| | .... ...1 | RPLNSP | Note the string position |

| 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 | Option flag byte 4: |
| | 1... .... | RPLENDTR | 3800 end of transmission |
| | .1.. .... | RPLMKFRM | 3800 mark form |
| | ..1. .... | RPLPDIC | PDIC passed |
| | .... ...1 | RPL1COPY | Pass only one copy of the data set |
| | ...x xxx. | | Reserved |
| 44 (2C) | 4 | RPLNXTRP<br>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: |
| | 1... .... | RPLAXPKP | Relative byte address |
| | 0... .... | | Prime-key pointer |
| | .xxx xxxx | | Reserved |
| 63 (3F) | 1 | | Reserved |
| 64 (40) | 4 | RPLDDDD | Relative byte address |
| 68 (44) | 1 | RPLEXTDS | Exit definition flags: |
| | | RPLEXTD1 | |
| | 1... .... | RPLEXSCH | An exit is scheduled |
| | .1.. .... | RPLNEXIT | No exit is specified |
| | ..1. .... | RPLEXIT | Asynchronous exit |
| | .... .1.. | RPLNIB | Argument has a pointer to the NIB |
| | .... ..1. | RPLBRANC | Branch entry to a macro |
| | ...x x..x | | 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—RPL Extension*

An RPLE is built and appended to each RPL built for an ISAM Interface user 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.

| 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 |

### *RWA—Reposition Work Area*

The RWA is built and freed by IDA0C06C and IDA0A05B. It is used for saving data needed to reposition the user's data sets at restart. VSAM I/O operations necessary during the checkpoint process use the RPL, PLH, and BUFC poritions of the work area. It is pointed to by the VRCWA (VRCWARWA).

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 1 | RWAFLAG1 | Save area for AMBLFLG1 |
| 1 (1) | 1 | RWAMBFLG | Save area for AMBFLG1 |
| 2 (2) | 2 | | Reserved |
| 4 (4) | 4 | RWAPSAV | Address of previous save area |
| 8 (8) | 4 | | Reserved |
| 12 (C) | 60 | RWARGSAV | Register save area |
| 72 (48) | 4 | RWARBA | RBA argument used by GET |
| 76 (4C) | 4 | RWABUFC | VSAM buffer pointer used by GET |
| 80 (50) | 4 | RWAEXLST | Save area for ACB exit list address |
| 84 (54) | 76 | | RPL used for GET |
| 160 (A0) | 280 | | PLH save area |

## *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.

The SSL is pointed to by OPWA (called the ACB work area). Additional SSLs are chained as required.

| 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: 'bIDASSLb' |
| 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 swap word for the Compare-and-Swap instruction |
| 24 (18) | 4 | SSLSWAP | Value to replace original in Compare-and-Swap |

## *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).

| 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 | UPTFLGO | 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 |

**UPT—Upgrade Table**

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| **UPT Header** | | | |
| 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:<br>0 Old<br>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 |
| | ...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 ofRPLAREA 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).

| 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 | | Reserved |
| 32 (20) | 1 | VATAMBLI | Zero. This field is used by VSAM Checkpoint/Restart to identify enhanced VSAM |
| 33 (21) | 3 | | Reserved |
| 36 (24) | 4 x 16 | VATAMBL | Addresses of VALID AMBLs |

## *VCRT—VSAM Checkpoint/Restart Table*

The VSAM Checkpoint/Restart Table (VCRT) is used by VSAM Checkpoint/Restart while processing the alternate-index environment introduced with enhanced VSAM. The VCRT contains a count, by entry type, of each entry appended to the VCRT. There are four types of VCRT entries, as follows:

1. The first entry type is the VCRT open entry, which points to the user and restart AMBL/ACB set to be opened by restart. This entry is sixteen bytes in length and contains pointers to the user AMBL, the restart AMBL, the user ACB, and the restart ACB. The restart AMBL and ACB pointers will be filled in at restart time.
2. The second entry type is the VCRT Upgrade Entry, containing pointers to the user and restart upgrade AMBLs to be processed by restart. This eight-byte entry will exist only if the immediate-upgrade set for this data set was open at checkpoint time.
3. The third entry type is the VCRT Upgrade ACB Entry, which contains only a pointer to the user ACB to be updated. This four-byte entry exists if there are ACBs open at checkpoint time which need not be opened for restart processing but must be updated at restart time.
4. The fourth entry type is the VCRT Index Entry. Eight bytes in length, this entry exists only if the base data set is a KSDS open for load-mode processing. There will be one index entry for each index level that exists at checkpoint time. The index entry contains ICWA and buffer pointers for the index level it represents.

The VCRT is created by VSAM checkpoint and, except in error situations, is freed by VS checkpoint and VSAM restart. The following diagram shows the format of the VCRT.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 1 | VCRID | VCRT ID field |
| 1 (1) | 3 | VCRFLAG1 | VCRT flags |
| | 1... ..... | VCRUPGSW | Entry type indicator:<br>1=process VCRT upgrade entry 0=process VCRT open entry |
| | .1.. .... | VCRLSR | LSR specified |
| | ..1. .... | VCROUT | Output ACB is open |
| | ...x xxxx | | Reserved (Bytes two and three are also reserved.) |
| 4 (4) | 8 | VCRIDNM | VCRT identification name,'IDAVCRT' |
| 12 (C) | 4 | VCRSIZE | VCRT size in bytes |
| 16 (10) | 4 | VCRCHAIN | Pointer to next VCRT |
| 20 (14) | 2 | VCROPNCT | Number of VCRT open entries |
| 22 (16) | 2 | VCRUPGCT | Number of VCRT upgrade entries |
| 24 (18) | 2 | VCRUPDCT | Number of VCRT update ACB entries |
| 26 (1A) | 2 | VCRIDXCT | Number of VCRT index entries |
| 28 (1C) | 4 | VCRVCRWA | Pointer to VSAM restart work area (VCRWA) |
| 32 (20) | 4 | VCRRBUF | Pointer to Restart buffer |
| 36 (24) | 4 | VCROPN | Pointer to VCRT open entries |

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 40 (28) | 4 | VCRUPG | Pointer to VCRT upgrade entries |
| 44 (2C) | 4 | VCRUPD | Pointer to VCRT update ACB entries |
| 48 (30) | 4 | VCRIDX | Pointer to VCRT index entries |
| 52 (34) | 4 | VCRPASSW | Pointer to password |

Pointers to the first of each of the following entry types are at offsets 36 (24) through 48 (30) above.

| | | |
|---|---|---|
| VCRT Open Entry | | |
| | VCRUAMBL | Pointer to user AMBL |
| | VCRRAMBL | Pointer to restart AMBL |
| | VCRUACBP | Pointer to user ACB |
| | VCRRACBP | Pointer to restart ACB |
| VCRT Upgrade Entry (For addressability, this is the same as the AMBL portion of the Open Entry) | | |
| | VCRUAMBL | Pointer to user immediate-upgrade AMBL |
| | VCRRAMBL | Pointer to restart immediate-upgrade AMBL |
| VCRT Update ACB Entry | | |
| | VCRUPDPT | Pointer to update ACB |
| VCRT Index Entry | | |
| | VCRICWA | Pointer to ICWA |
| | VCRBUFPT | Pointer to associated buffer |

## *VCRWA—VSAM Checkpoint/Restart Work Area*

The VSAM Checkpoint/Restart Work Area (VCRWA) is created by VSAM checkpoint. If an error occurs during VSAM checkpoint processing, it is also freed by VSAM checkpoint. If no error occurs, it is freed by VS checkpoint during checkpoint processing, and by VSAM restart during restart processing. The VCRWA is shared by all loads of VSAM checkpoint restart and is therefore saved, in the checkpoint data set, at checkpoint time and restored at restart time.

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 0 (0) | 1 | VCRWID | VCRWA ID field |
| 1 (1) | 1 | VCRWFLG | Reserved |
| 2 (2) | 2 | VCRWSIZE | VCRW size in bytes |
| 4 (4) | 8 | VCRWIDNM | VCRWA identification name, IDAVCRWA |
| 12 (C) | 4 | VCRWVSWA | Pointer to VS work area |
| 16 (10) | 4 | VCRWVCRT | Pointer to current VCRT |
| 20 (14) | 4 | VCRWRWA | Pointer to VSAM reposition work area |
| 24 (18) | 4 | VCRWRWAS | Size in bytes of checkpoint/restart RWA, RPL, and PLH |
| 28 (1C) | 4 | VCRWRPL | Pointer to checkpoint/restart RPL |
| 32 (20) | 4 | VCRWPLH | Pointer to checkpoint/restart PLH |
| 36 (24) | 4 | VCRWSHR | JSCBSHR save area |
| 40 (28) | 4 | VCRWRET1 | Register 14 save area one |
| 44 (2C) | 4 | VCRWRET2 | Register 14 save area two |
| 48 (30) | 4 | VCRWRET3 | Register 14 save area three |
| 52 (34) | 4 | VCRWBASE | Save area for IDA0A05B base register |
| 56 (38) | 4 | VCRWRACB | Pointer to restart ACB core |
| 60 (3C) | 4 | VCRWACBS | Restart ACB core size |
| 64 (40) | 4 | VCRWB05B | Pointer to second level VSAM Restart |
| 68 (44) | 3 | VCRWECB | Restart page-fix ECB |
| 69 (47) | 1 | VCRWCC | Page-fix ECB flags |
| | xxxx x.xx | | unused |
| | .... .1.. | VCRWAD | Page-fix error indicator |
| 72 (48) | 4 | VCRWBUFC | Pointer to BUFC save area |
| VCRWMSG - Error message | | | |
| 76 (4C) | 16 | VCRWHEBS | Message body and HEB save |
| 92 (5C) | 5 | | Remainder of message body |
| 97 (61) | 8 | VCRWDDNM | DD name of data set in error |
| 105 (69) | 3 | | Remainder of message |
| VCRWGLST - GETMAIN list | | | |
| 108 (6C) | 4 | VCRWRETA | GETMAIN return address |
| 112 (70) | 4 | VCRWSZ | GETMAIN core size |
| 116 (74) | 12 | VCRWPARM | GETMAIN parameter list |
| VCRWRSAV - Register save area header | | | |
| 128 (80) | 4 | VCRWRS01 | Standard save area header |
| 132 (84) | 4 | VCRWRS02 | Standard save area header |
| 136 (88) | 4 | VCRWRS03 | Standard save area header |

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| VCRWRSAV - Register save area header | | | |
| 140 (8C) | 4 | VCRWRS04 | Standard save area header |
| 144 (90) | 4 | VCRWREGE | Register 14 save area |
| 148 (94) | 4 | VCRWREGF | Register 15 save area |
| 152 (98) | 4 | VCRWREG0 | Register 0 save area |
| 156 (9C) | 4 | VCRWREG1 | Register 1 save area |
| 160 (A0) | 4 | VCRWREG2 | Register 2 save area |
| 164 (A4) | 4 | VCRWREG3 | Register 3 save area |
| 168 (A8) | 4 | VCRWREG4 | Register 4 save area |
| 172 (AC) | 4 | VCRWREG5 | Register 5 save area |
| 176 (B0) | 4 | VCRWREG6 | Register 6 save area |
| 180 (B4) | 4 | VCRWREG7 | Register 7 save area |
| 184 (B8) | 4 | VCRWREG8 | Register 8 save area |
| 188 (BC) | 4 | VCRWREG9 | Register 9 save area |
| 192 (C0) | 4 | VCRWREGA | Register 10 save area |
| 196 (C4) | 4 | VCRWREGB | Register 11 save area |
| 200 (CB) | 4 | VCRWREGC | Register 12 save area |
| 204 (CC) | 4 | VCRWBAS2 | Save area for second base register |
| 208 (D0) | 4 | VCRWRKPT | Workarea pointers |
| 208 (D0) | 4 | RVATSAVE | VAT pointer save area |
| 208 (D0) | 4 | RWORKA | Work pointer A |
| 20C (D4) | 4 | RWORKB | Work pointer B |
| 210 (D8) | 4 | RWORKC | Work pointer C |
| 214 (DC) | 4 | IRBAD | IRB address |

## *VMT—Volume Mount Table*

The VMT identifies and describes volumes to be 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).

| 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 × *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).

| 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 |
| 12 (C) | 2 | VSRTFLGS | Flags: |
| | | | Byte 1: |
| | .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 |
| | x... 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) |

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 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 | | Reserved |
| 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 IOBs |
| | .... ...1 | VSRTCSBH | The storage contains a buffer |
| 37 (25) | 3 | VSRTCSSP | 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).

| 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 | WASBPLH | Address of the PLH for the base cluster |

| Offset | Bytes and Bit Pattern | Field Name | Description |
|---|---|---|---|
| 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).

| 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 |
| 2 4 (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 the cross-reference information published on microfiche cards.
- A list of messages issued by VSAM, cross-referenced to enable you to detect the module causing the message to be issued.
- A list of function codes that appear in messages to indicate the operation being performed when an error occurred.
- A list of VSAM macro instructions and their functions.
- A catalog debug aid that provides dumps that can be selected and activated upon termination of a catalog management request.
- A description of a system-provided service, GTF, used by VSAM, how VSAM requests this service, and what this service provides in the way of VSAM APAR information.
- A description of the Catalog Communications Area's register save area.
- A list of return codes and error codes.
- A description of the control blocks and control block interrelationships of the Virtual-Storage Manager.

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."
- A definition of terms and abbreviations used in this book, and in the VSAM listings, which is in the "Glossary."

## Microfiche Cross-Reference Aids

*OS/VS1 VSAM Cross Reference* contains valuable information that you should be aware of. Two types of cross-reference information are available:

- Symbolic-name usage table: lists each symbolic name that appears in the VSAM code listings, lists each module that refers to the symbolic name, and specifies how each module refers to the symbolic name.
- Macro-instruction usage table: lists each macro instruction that is issued in VSAM listings, specifies the total number of times the macro instruction is issued, lists each module that issues the macro instruction, and specifies the number of times the module issues the macro instruction.

### *How To Read the Symbolic-Name Usage Table*

*OS/VS1 VSAM Cross Reference* contains the symbolic-name usage table, or Symbol Where Used Report, for VSAM listings. Three kinds of information are available from the table, as shown in Figure 74:

- A list of symbolic names—this includes field names, symbolic address names, return code names, constant/value names, flag-bit names, etc.—in alphanumeric order from top to bottom on the page.

  **Note:** In the lower-right corner of each page, the lowest and highest name for the page is shown.
- A list of modules that refer to each symbolic name, in alphanumeric order from left to right across the page.
- 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

  then the module's use of 'A' is to modify it ('A' appears to the left of an equate sign in a statement that is not an 'IF' statement.)

  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, using it to modify 'A' ('B' appears to the right of an equate sign in any type of statement).

  C–COMPARE — The data field or value was compared against another value. If the module contains a statement:

  IF A = B, THEN ...

  then the module's use of 'A' is to compare it to 'B' ('A' appears to the left of an equate sign in an 'IF' statement). Note that 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 usage table.

```
DATE: 01/15/77      SYMBOL WHERE USED REPORT --- OS/VS2 RELEASE 1.7 VSAMPAGE 3
                                                                    (EXTERNAL SYMBOLS)

SYMBOL   MODULE    ACCESS   MODULE    ACCESS   MODULE    ACCESS   MODULE    ACCESS   MODULE    ACCESS   MODULE    ACCESS
-------------------------------------------------------------------------------------------------------------------------
ACBBUFSP IEAVNP1A  R
ACBCAT   IDACAT11  W        IDACAT12  W        IGG0CLAD  W        IGG0CLAE  W
ACBCBMWA IDA019C1  DRW
ACBCRNCK IGC0A05B  C
ACBCRNRE IGC0B05B  C        IGC0C06C  C
ACBDDNM  IDACAT11  W        IDA019C1  W        IEAVNP1B  R        IEAVNP11  W        IFG0191X  R        IGC0A05B  W
         IGG0CLAD  W        IGG0CLAE  W        IGG0CLBG  W
ACBDEB   IDACAT13  R        IEAVNP1A  W        IGC0B05B  C
ACBDIR   IEAVNP1A  C        IFG0191X  W
ACBDORGA IDACAT11  W        IDA019C1  W        IGC0C06C  C        IGG0CLAD  W        IGG0CLAE  W        IGG0CLBG  W
ACBDOSID IGG0CLAD  W        IGG0CLAE  W        IGG0CLBG  W
ACBDTFID IGG0CLAD  R        IGG0CLAE  R        IGG0CLBG  R
ACBERFL  IFG0191X  W
ACBERFLG IFG0191Y  W        IFG0200N  W        IGC0A05B  RWC      IGG0CLAE  C
ACBEXFG  IDACAT11  W        IDA019C1  W        IGG0CLAD  W        IGG0CLBG  W
ACBEXLST IGC0C06C  RW
ACBID    IDACAT11  W        IDA019C1  WC       IGC0C06C  C        IGG0CLAD  W        IGG0CLAE  W        IGG0CLBG  W
ACBIDVAL IGG0CLAD  R        IGG0CLAE  R        IGG0CLBG  R
ACBIN    IDA019C1  W        IFG0191X  W        IGG0CLBG  W
ACBINFL  IEAVNP1A  R        IFG0191X  W
ACBINRTN IGC0B05B  R
ACBKEY   IDA019C1  W        IFG0191X  W        IGG0CLBG  W
ACBLEN   IDACB2    M
ACBLENG  IDACAT11  W        IDA019C1  W        IGG0CLAD  W        IGG0CLAE  W        IGG0CLBG  W
-------------------------------------------------------------------------------------------------------------------------
ACCESS CODES: D=DEFINITION, R=READ, W=WRITE, C=COMPARE, E=EQUATE OPERAND, M=MACRO, A=ABSOLUTE, P=PARAMETER
```

Figure 74. Symbolic-Name Usage Table

## *How To Read the Macro-Instruction Usage Table*

*OS/VS1 VSAM Cross Reference* contains the macro-instruction usage table, or Macro Where Used Report for VSAM listings. Three kinds of information are available, as shown in Figure 75:

- A list of macro-instruction names in alphanumeric order from top to bottom.
- A list of the modules that issue each macro-instruction, in alphanumeric order from left to right across the page.
- The total number of times all VSAM modules issued the macro-instruction, and the number of times each module in the list issued the macro-instruction.

```
DATE: 01/15/77          MACRO WHERE USED REPORT  --- OS/VS2 RELEASE 1.7 VSAM                    PAGE1
MACRO    TOTAL #  MODULE      #   MODULE      #   MODULE      #   MODULE      #   MODULE      #   MODULE      #
-------- -------  ----------  --  ----------  --  ----------  --  ----------  --  ----------  --  ----------  --
ABEND          1  IEAVNP11     1
ABPCALL        1  IEAVNP12     1
ACB            1  IGC0A05B     1
ACBX           1  IEAVNP11     1
ADDRESS        1  IGG0CLAP     1
AMCBS             IDACAT11     1  IDACAT12     1  IDACAT13     1  IEAVNP1A     1  IEAVNP1B     1  IEAVNP11     1
               8  IEAVNP12     1  IGGMCDCL     1
AMOREGN        1  IGGMCDCL     1
ASM1           2  IEAVNP1A     1  IEAVNP11     1
BLDL           2  IEAVNP11     2
CALL           5  IDACB2       2  IGG0CLAQ     4
CARD           1  IGG0CLAP     1
CATGODSP       1  IGG0CLAP     1
CATG0GC1       1  IGG0CLAP     1
CATGOSEQ       1  IGG0CLAP     1
CCAASCIK       1  IGG0CLAP     1
CCACD1         2  IGG0CLAQ     2
CCACPE2        1  IGG0CLAP     1
CCACPE3        1  IGG0CLAP     1
CCACPE4        1  IGG0CLAP     1
CCACPE5        1  IGG0CLAP     1
CCACPE6        2  IGG0CLAP     2
CCACPE7        1  IGG0CLAP     1
CCARABB        1  IGG0CLAP     1
CCARABFL       1  IGG0CLAP     1

                                                                  MACRO ENTRIES: ABEND    - CCARABFL
```

Figure 75. Macro-Instruction Usage Table

# Messages

**Messages IDA001 through IDA025 are macro-instruction messages and refer to an incorrectly coded macro instruction.**

| 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 |

| Message Number | Message Text |
|---|---|
| IDA021 | *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 NONVSAM CONTROL BLOCK |
| IDA025 | *www,xxx,yyy* CONFLICTING SUBPARAMETERS IN *zzz* KEYWORD, *www* ASSUMED |

| Message Number | Message Text | 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 | *rrr[(sfi)]*-ccc,jjj,sss,ddn, *ddd,vol,cln,dsn,cat* | IDA0192C<br>IDA0192D<br>IDA0192S<br>IDA0192V<br>IDA0557A<br>IFG0551F | IDA0192P |
| IEC101A | M *ddd,ser,jjj,sss,dsn* | IGG0CLBL | IDA0192V |
| IEC111E | D, *ddd,ser* | IGG0CLBL | IDA0192V |
| IEC113A | ENTER PASSWORD FOR DATA SET | IGG0CLBG<br>IGG0CLBM<br>IGG0CLB6 | |
| IEC114E | D *ddd* [*ddn-n*] | IGG0CLBL | |
| IEC115I | INVALID PASSWORD | IGG0CLBG<br>IGG0CLBM<br>IGG0CLB6 | |
| IEC116I | REENTER | IGG0CLBG<br>IGG0CLBM<br>IGG0CLB6 | |
| IEC130I | *ddn* - DD STATEMENT MISSING | IFG0191X | |
| IEC161I | *rrr[(sfi)]*-ccc,jjj,sss,ddn, *ddd,vol,cln,dsn,cat* | IDA0192C<br>IDA0192D<br>IDA0192S<br>IDA0192V<br>IDA0192Z<br>IDA0192A<br>IFG0193A | IDA0192P |
| IEC251I | *rrr[(sfi)]*-ccc,jjj,sss,ddn, *ddd,vol,cln,dsn,cat* | IDA0192C<br>IDA0192D<br>IDA0192S<br>IDA0192V<br>IDA0200T<br>IFG0200V | IDA0192P |
| IEC252I | *rrr[(sfi)]*-ccc,jjj,sss,ddn, *ddd,vol,cln,dsn,cat* | IDA0192C<br>IDA0192D<br>IDA0192V<br>IDA0231T<br>IGC0002C | IDA0192P |
| IEC301A | S JOB *xxxxxxxx* DSNAME | IGG0CLBM | |
| IEC331I | *ccc-rrr,jjj,sss,ffff,mmm* | IGG0CLAG | IGG0CLAF |

| Message Number | Message Text | Detected by | Issued by |
|---|---|---|---|
| | | (IGGPIORA) | |
| (IGGPEMSG) | | | |
| IEC332I | *fff [ffff...]* | IGG0CLAG (IGGPIORA) | IGG0CLAF |
| (IGGPEMSG) | | | |
| IEC333I | *teee,xx,ddd,iii* | IGG0CLAG (IGGPIORA) | IGG0CLAF |
| (IGGPEMIO) | | | |
| IEC338I | IGG0CLC9, VALIDITY CHECK FAILED ON CATALOG PARAMETER LIST STORAGE | | |
| IEC339I | IGG0CLC9, INSUFFICIENT STORAGE FOR VSAM CATALOG COMMUNICATION AREA | | |
| IEF175I | AMP KEYWORD *xxxxxxxx* DUPLICATE OR CONFLICTING PARM STEP NOT EXECUTED | IEFNB902 | |
| IEF447I | AMP KEYWORD *nnnnnnnn* IS INVALID STEP WAS NOT EXECUTED | IEFNB902 | |
| IEF448I | AMP KEYWORD *nnnnnnnn* VALUE *xxxxxx* IS TOO LARGE STEP NOT EXECUTED | IEFNB902 | |
| IEF449I | AMP KEYWORD *nnnnnnnn* REQUIRES A DECIMAL VALUE STEP NOT EXECUTED | IEFNB902 | |
| IHJ000I | CHKPT *jjj* (*ddn*) NOT TAKEN (*xxx*) | | |
| IHJ007I | RESTART NOT SUCCESSFUL FOR *jjj* (*xxx* [,*cuu*]) | | |
| IHJ009I | ERROR ON *ddn* | IDA0A05B | |

## *Function Codes for VSAM Open, Close, and EOV Messages*

When an error occurs during open, close, or EOV processing for a VSAM data set, the message that is issued will contain 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 |
|---|---|---|
| 1 | IDA0192C | Initialize for catalog interface processing. |
| 2 | IDA0192C | Determine which data sets are associated with dsname or 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. |

| Function Code | Module that Detected Error | Operation Being Performed When Error Was Detected |
|---|---|---|
| 11 | IDA0192C | Retrieve 44-byte component name. |
| 20 | IDA0192V | Initialize for mounting and verify volume. |
| 21 | IDA0192V | Check volume timestamp. |
| 22 | IDA1092V | 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 | IDA1092W | Initialize for building channel program. |
| 71 | IDA0192W | Build channel program area. |
| 80 | IFG0193A | Read JFCB. |
| 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. |
| 88 | IDA1092F | Open base cluster. |
| 89 | IDA1092F | 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 VSAM ACB processed on a system not generated for VSAM. |
| 100 | IFG0200V | Read JFCB. |
| 101 | IDA0200T | Initialize for VSAM close processing. |
| 103 | IDA0200T | Complete deferred write requests. |
| 104 | IDA0200T | Close path. |

| Function Code | Module that Detected Error | Operation Being Performed When Error Was Detected |
|---|---|---|
| 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. |
| 150 | IGC0002C | Read JFCB. |
| 151 | IDA0231T | Initialize for VSAM CLOSE (TYPE=T) processing. |
| 153 | IDA0231T | Complete deferred write requests. |
| 154 | IDA023lT | Close (TYPE=T) path. |
| 155 | IDA023lT | Close (TYPE=T) base cluster. |
| 156 | IDA023lT | 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. |
| 200 | IFG0551F | Read JFCB. |
| 202 | IDA0557A | Initialize for VSAM end-of-volume processing. |
| 202 | IDA0557A | Locate and mount volume. |
| 203 | IDA0557A | Allocate space. |
| 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. |

# Macro Instructions

The following tables list VSAM and OS/VS macro instructions and explain what they do. Each module that issues the macro instruction is also listed. The macro instructions are divided into those that define control blocks and data area (mapping macro instructions) and those that issue executable code (action macro instructions).

## *Mapping Macro Instructions*

The following table lists macro instructions that define the format of control blocks and data areas used by VSAM modules.

**Macro Instructions That Define Data Areas**

| Macro Instruction | Description |
|---|---|
| AMCBS | Maps the VSAM catalog vectors table (AMCBS) |
| CIRB | Maps the CIRB control block |
| CLCL | Maps the CLCL control block |
| CVT | Maps the communications vector table (CVT) |
| F4DSCB | Maps the Format-4 Data Set Control Block (DSCB) |
| IDAAIR | Maps the alternate-index record |
| IDAAMB | Maps the Access Method Block (AMB) |
| IDAAMBL | Maps the Access Method Block List (AMBL) |
| IDAAMDSB | Maps the Access Method Data Set Statistics Control Block (AMDSB) |
| IDAARDB | Maps the Address Range Definition Block (ARDB) |
| IDABFK | Maps the buffer control set |
| IDABIB | Maps the Base Information Block (BIB) |
| IDABLPRM | Maps the Resource Pool Parameter List (BLPRM) |
| IDABSPH | Maps the Buffer Subpool Header for shared resources (BSPH) |
| 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 (CLW) |
| IDACMB | Maps the Cluster Management Block (CMB) |
| IDACPA | Maps the Channel Program Area (CPA) |
| IDACSL | Maps the Core Save List (CSL) |
| IDACTREC | Maps the work area built when the VSAM catalog management routines issue OPEN (SVC 19), CLOSE (SVC 20), or SVC 55 (calls End of Volume processing). |
| IDADIWA | Maps the Data Insert Work Area (DIWA) |
| 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/Close ACB Error Codes |

**Macro Instructions That Define Data Areas**

| Macro Instruction | Description |
|---|---|
| IDAESL | Maps the Enqueue Save List (ESL) |
| IDAFOREC | Maps the work area for VSAM Open/Close/EOV Modules |
| | 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 |
| IDAIOSCN | Maps the VSAM Open/Close/EOV 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 Open Work Area (OPW) |
| IDAPDPRM | Maps the VSAM Open/Close/EOV problem determination parameter list |
| IDAPLH | Maps the Placeholder (PLH) |
| 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) |
| IDASHOW | Maps the SHOWCB header argument control entry |
| 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) |
| IDAVMT | Maps the Volume Mount Table (VMT) |
| IDAVSRT | Maps the VSAM Shared Resource Table (VSRT) |
| IDAVUCBL | Maps the VSAM Open/EOV: Volume Mount and Verify UCB list |
| IDAVVOLL | Maps the VSAM Open/EOV: Volume Mount and Verify volume serial number list |
| IDAWAX | Maps the Work Area for Path Processing (WAX) |
| IDAWSHD | Maps the Working Storage Header (WSHD) |
| IECDIOSB | Maps the I/O supervisor control block |
| IECDSECS | Maps the DSECTS |
| IECSDSL1 | Maps the SDSL1 |
| IEESMCA | Maps the SMCA |

**Macro Instructions That Define Data Areas**

| Macro Instruction | Description |
|---|---|
| IEFJESCT | Maps the JESCT |
| IEFJFCBN | Maps the Job File Control Block (JFCB) |
| IEFJFCBX | Maps the Job File Control Block (JFCB) |
| IEFJMR | Maps the JMR |
| IEFPCCB | Maps the Private Catalog Control Block (PCCB) |
| IEFQMIOP | Maps the QMIOP |
| IEFTCT | Maps the TCT |
| IEFTIOT1 | Maps the Task Input/Output Table (TIOT) |
| IEFUCBOB | Maps the OS/VS Unit Control Block (UCB) |
| IEZCTGCV | Maps the VSAM Catalog Control Volume List (CTGCV) |
| IEZCTGFL | Maps the VSAM Catalog Field Parameter List (CTGFL) |
| IEZCTGFV | Maps the VSAM Catalog Field Vector Table (CTGFV) |
| IEZCTGPL | Maps the VSAM Catatalog Parameter List (CTGPL) |
| IEZCTGVL | Maps the VSAM Catalog Volume List (CTGVL) |
| IEZCTGWA | Maps the VSAM Catalog Scheduler Work Area (CTGWA) |
| IEZDEB | Maps the OS/VS Data Extent Block (DEB) |
| IEZIOB | Maps the OS/VS Input/Output Block (IOB) |
| IEZJSCB | Maps the OS/VS 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 VSAM Catalog Auxiliary Work Area (CAXWA) |
| IGGCCA | Maps the VSAM Catalog Communications Area (CCA) |
| IGGMCDCL | Contains the commonly used control block formats and constants for VSAM catalog management modules<br><br>IGGMCDCL issues AMCBS, AM0REGN, COMREGN, CVT, IEZCTGCV, IEZCTGFL, IEZCTGFV, IEZCTGPL, IEZCTGWA, IFGACB, IGGCAXWA, IGGCCA, IGGMCTRC, and IKJTCB. |
| IGGMCMDM | Maps the VSAM catalog management commonly-used record structures |
| IGGMCMWA | Maps the VSAM catalog management services work area |
| IGGMCTRC | Lists the catalog management return codes |
| IGGMDRWA | Maps the VSAM catalog DSCB read-in work area |
| IGGMF4WA | Maps the VSAM Catalog Management format-4 DSCB work area |
| IGGMGVO | Maps the volume information set of fields |
| IGGMSAWA | Maps the VSAM Catalog Management: Suballocate work area |
| IGGMUPDE | Defines the commonly-used declarations for VSAM Catalog Management: Update-Extend modules<br><br>IGGMUPDE issues IDAAMDSB, IGGMCDCL, IGGMCMDM, IGGMSAWA, and IGGMVEDC. |
| IGGMVEDC | Maps the Volume Catalog Record |
| IGGMZLOC | Defines the commonly-used declarations for VSAM Catalog Management: Suballocate modules |

**Macro Instructions That Define Data Areas**

| Macro Instruction | Description |
|---|---|
| IHADCB | Maps the OS/VS Data Set Control Block (DCB) |
| IHADCBDF | Maps the OS/VS Data Set Control Block (DCB) |
| IHADECB | Maps the OS/VS Data Extent Control Block (DECB) |
| IHARB | Maps the OS/VS Request Block (RB) |
| IKJPSCB | Maps the OS/VS PSCB |
| IKJTCB | Maps the Task Control Block (TCB) |
| SGIDA401 | Lists the VSAM SYSGEN global definitions |
| XCTLTABL | Maps the OS/VS Transfer Control (XCTL) table |

## *Action Macro Instructions*

This table lists the macro instructions that generate executable code.

**Macro Instructions That Generate Executable Code**

| Macro Instruction | Description |
|---|---|
| ABEND | Abnormal termination (OS/VS macro instruction) |
| ADDREC | Calls IGGPPAD to write a new record into the catalog |
| BLDVRP | Builds a VSAM resource pool (for shared resources) |
| CALLSF xxxx | Transfers control to procedure IGGPxxxx |
| CALL EXIT | Returns control to the caller of the procedure |
| CATLG | Loads the address of the catalog parameter list (CTGPL) into register 1 and issues SVC 26 |
| CATPROB | Problem determination |
| CLOSE | VSAM CLOSE: Disconnects a user from a VSAM data set |
| COMB | Generates combination name entries in the VSAM combination name index table |
| DEBCHK | Checks the validity of the DEB |
| DELETE | (Same as OS/VS DELETE macro instruction) |
| DELREC | Calls IGGPPDE to erase a catalog record |
| DEQ | (Same as OS/VS DEQ macro instruction) |
| DEVTYPE | Determines the direct-access device type |
| DICT | Generates entries in the VSAM catalog field name dictionary |
| DLVRP | Deletes a VSAM resource pool (for shared resources) |
| DOM | Deletes operator message (same as OS/VS DOM macro instruction) |
| ENDREQ | Terminates a VSAM record processing request (such as GET or PUT) |
| ENQ | (Same as OS/VS ENQ macro instruction) |
| ERASE | Deletes a VSAM record |
| EXCP | (Same as OS/VS EXCP macro instruction) |
| EXCPVR | (Same as OS/VS EXCPVR macro instruction) |
| FREEMAIN | Releases virtual storage obtained by a GETMAIN |
| GENCB | Generates a VSAM control block (ACB, EXLST, RPL) |
| GET | Retrieves a record from a data set on a direct-access device |

**Macro Instructions That Generate Executable Code**

| Macro Instruction | Description |
|---|---|
| GETIX | Retrieves a control interval from the index of a key-sequenced data set |
| GETMAIN | Obtains virtual storage for a temporary work area |
| GETREC | Calls IGGPGET to retrieve a catalog record |
| GTRACE | Calls the Generalized Trace Facility (GTF) to copy VSAM control blocks |
| IDABFR | Sets the RBA and/or status in the buffer control block (BUFC) |
| 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 macro instructions (GENCB, MODCB, SHOWCB, and TESTCB) |
| IDACB2 | Scans keywords and generates code for Control Block Manipulation macro instructions |
| IDAERMAC | Prints MNOTEs for Control Block Manipulation macro instruction 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<br>IDAEXITR issues DELETE, IDARST14, IDASVR14, and LOAD. |
| IDAGMAIN | Gets virtual storage for VSAM Open/Close/EOV |
| IDAPATCH | Generates maintenance space |
| IDARST14 | Puts the return address in register 14 |
| IDASVR14 | Saves register 14 in the placeholder (PLH) push-down list |
| IECRES | Transfers control to the OS/VS Resident routine |
| IGGMEND | Generates code at the end of VSAM Catalog Management modules |
| IGGMODUL | Generates header code for VSAM Catalog Management modules |
| IGGMPROC | Generates header code for VSAM Catalog Management procedures |
| LOAD | (Same as OS/VS LOAD macro instruction) |
| MODCB | Modifies a VSAM control block (ACB, EXLST, RPL) |
| MODESET | (Same as OS/VS MODESET macro instruction) |
| MRKBFR | Marks a buffer in a VSAM resource pool |
| OBTAIN | (Same as OS/VS OBTAIN macro instruction) |
| OPEN | Connects a user's program to a VSAM data set |
| PGFIX | "Fixes" a page of virtual storage so that it appears (to OS/VS) as real storage |
| PGFREE | "Frees" a "fixed" page of virtual storage |
| POINT | Identifies a starting point in a VSAM data set |
| POST | (Same as OS/VS POST macro instruction) |
| PUT | Writes a record into a VSAM data set |
| PUTIX | Writes a control interval in the index of a key-sequential data set |
| RESERVE | (Same as OS/VS RESERVE macro instruction) |
| RETURN | (Same as OS/VS RETURN macro instruction) |
| SCHBFR | Searches for a control interval in a VSAM resource pool |
| SHOWCAT | Displays information from a VSAM catalog |

**Macro Instructions That Generate Executable Code**

| Macro Instruction | Description |
|---|---|
| SHOWCB | Displays information from a VSAM control block |
| SMFWTM | Writes the SMF message into the SMF data set |
| SYNCH | (Same as ISAM SYNCH macro instruction) |
| TESTCB | Tests information in a VSAM control block (ACB, EXCST, RPL) |
| TIME | Obtains the correct time from the OS/VS 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 OS/VS WAIT macro instruction) |
| WRTBFR | Writes a buffer from a VSAM resource pool |
| WTO | Writes a message to the operator (no reply) |
| WTOR | Writes a message to the operator (a reply is expected) |
| XCTL | Transfers control (same as OS/VS XCTL macro instruction) |

**Note:** The following macros are VSAM user's macros and are described in detail in the *OS/VS Virtual Storage Access Method (VSAM) Programmer's Guide:*

CLOSE
ENDREQ
ERASE
GENCB
GET
MODCB
OPEN
POINT
PUT
SHOWCB
TESTCB

The following VSAM user's macros are described in detail in *OS/VS Virtual Storage Access Method (VSAM) Options for Advanced Applications:*

BLDVRP
DLVRP
GETIX
MRKBFR
PUTIX
SCHBFR
SHOWCAT
WRTBFR

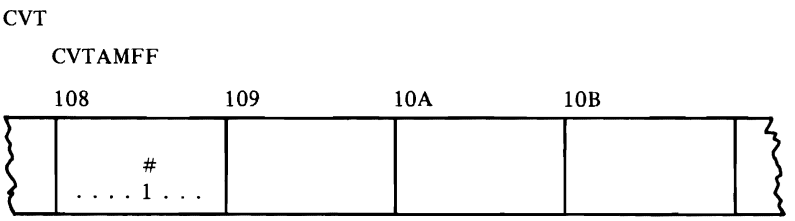
## Using the CVT's VSAM Debug Switches

The CVTAMFF field (displacement = X'108') in the CVT (Communications Vector Table) allows the PSR (Programming Systems Representative) to run a program using VSAM that contains an error and, when the error occurs, to save certain VSAM control blocks and work areas that would otherwise be destroyed.

### *Getting A Dump of Open, Close, and End-of-Volume Work Areas*

The messages that the problem determination routine (IDA0192P) issues for open, close, and end of volume (the function codes for these messages follow) may not be sufficient to find the problem.

In such a case, you can obtain an ABEND dump by turning on (setting to '1') a bit in the CVT and rerunning the job in error. Use the CPU manual procedure AM (alter real storage) to set bit 4 of the first byte of CVTAMFF to '1'. The CVT's location is stored in fixed-storage location X'10'. Add X'0108' to the CVT's address, or set this byte to X'08'.




When an error occurs with this bit turned on, the problem determination module (IDA0192P) issues its messages and also issues an ABEND with a user code of 888.

The contents of the general registers (0—15) of the module that called IDA0192A (the same module identified by the function code) can be found at the address calculated by adding X'0140' to the contents of register 4 at entry to ABEND.

The caller's register 13 points to its save area, which precedes its work area. By following the save area chaining address from save area to save area, you can locate the work area of each module in open, close, or end of volume that processed before the error occurred.

Each module in open, close, and end of volume pairs its save area with its work area. Work areas have no general format, but vary from module to module.

## Using the VSAM Catalog Debug Aid

The VSAM debug catalog aid allows the PSR to exercise certain options when VSAM catalog management requests terminate. The options are trapping and issuing a problem determination message. Either one or both can be selected, and you can specify that they be activated upon termination of (1) all requests, (2) only those requests that generate a nonzero return code in CCACD1, (3) only those requests that generate an abnormal return code in CCACD1, or (4) only those requests that generate a specific return code in CCACD1.

### *Defining Debug Aid Options*

Debug aid options are defined by storing values and setting bits within the CVTAMFF field in the CVT. The PSR can use the CPU manual procedure AM (alter main storage) to modify the CVTAMFF field. (Note, however, that bits 0-3 of CVTAMFF must not be changed.)

You accomplish debug activity by storing a nonzero value (X'01'—X'FF') into CVTAMFF+1 (the CVT's location + X'109') and X'07FE' (a BR 14 instruction) into bytes 3 and 4 of CVTAMFF (CVT's location + X'10A'). The nonzero value you store determines the scope of the debug activity, as follows:

| | |
|---|---|
| X'01' | Exercise options upon termination of all requests |
| X'02' | Exercise options only when the catalog return code is nonzero |
| X'03' | Exercise options only when the catalog return code is not a normal return code (0, 8, 36, 40, 44, 76, and 140, and reason codes 40, 188, and 240 are considered normal) |
| X'04'-X'FF' | Exercise options only when the catalog return code equals the value stored. |

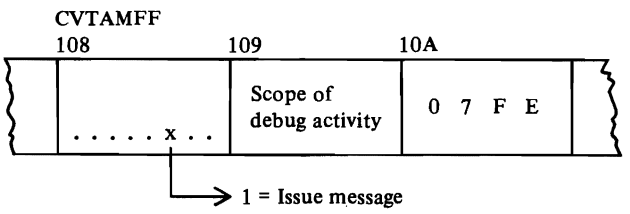
### *Selecting Debug Options*

Each option selected will be exercised only when the catalog termination routine determines that the catalog return code (in CCACD1) falls within the defined scope. The trap option is activated by setting a hardware address stop, a DSS 'AT', or a VM ADSTOP at the location of the BR 14 instruction contained in the low-order two bytes of CVTAMFF (the CVT's location + X'10A'). The catalog termination routine executes a BALR R14, R15 instruction to pass control to the BR 14 instruction.

Register contents at the time of the debug trap are:

| Register | Contents |
|---|---|
| 0 | CPL bytes 0, 1, 2, and 16 (the type of catalog management request can be derived from the information contained in these bytes) |
| 1 | Contents of CCAPROB (module ID, error code, and return code) |
| 11 | Pointer to CCA |
| 14 | Return address |
| 15 | Address of trap instruction |

To cause determination message IEC331I to be issued, set bit 5 (X'04') of CVTAMFF (the CVT's location + X'108') to 1.



## 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 OS/VS 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/EOV problems:

- The error trace routine traces VSAM control blocks when an error is detected. The optional work area trace traces the Open/Close/EOV work area WTG table prefix and the current entry in the WTG table at entry to and exit from the VSAM Open/Close/EOV modules.
- The work area trace is requested by specifying AMP="TRACE". This is the same trace that is obtained for nonVSAM Open/Close/EOV processing when DCB=DIAGNS=TRACE is specified in the JCL. (For details on the AMP and DCB JCL parameters and options, see *OS/VS JCL Reference.*

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/VS Service Aids.*

## Catalog Communication Area Register Save Area

A catalog communication area (CCA) is built for every call to VSAM catalog management. The CCA contains a register save area (CCAREGS) that allows the PSR (programming systems representative) to follow the flow of control from one catalog management external procedure to another, through each procedure called to process the request.

The contents of registers 12, 13, and 14 are put into CCAREGS whenever a catalog management procedure is entered. The current value of register 13 is the address of the latest entry in CCAREGS. If an external catalog management procedure is entered from another catalog management procedure, three words are saved as follows:

- the first word contains the contents of register 12—the calling procedure's base address,
- the second word contains the contents of register 13—a pointer to the previous 12-byte entry in the register save area (CCAREGS), and
- the third word contains the contents of register 14—the return address in the calling procedure.

Immediately after registers 12, 13, and 14 are saved (at register 13 + 12 (decimal)), register 12 is updated to contain the called procedure's base address. Register 13's value is increased by 12, so that it points to the latest

entry in CCAREGS. While a catalog management procedure is processing, register 11 contains a pointer to the beginning of the CCA.

Note that backward movement is not recorded in the trace table. For example, if procedure B returns to procedure A, the return is not shown in the register save area.

## Error Codes

VSAM sets error codes in the RPL, the ACB, and the CCA. Codes in the RPL and the ACB are paired with codes in register 15. Error codes set in the RPL are listed and explained under "Record Management Error Codes." Those set in the ACB are listed and explained under "Open, Close and End-of-Volume Error Codes." And those set in the CCA, are listed and explained under "Catalog Management Error Codes."

VSAM sets a pair of codes in registers 15 and 0 for control block manipulation macros. These are listed and explained under "Control Block Manipulation Error Codes."

### *Record Management Error Codes*

After a request macro is issued 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:

| FDBK 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) | For GET requests, indicates a duplicate key follows; for PUT requests, indicates a duplicate key was created in an alternate index with the nonunique attribute. |
| 12 (C) | (Shared resources only.) A buffer needs to be written. |

See the discussions below of the LERAD exit routine for the logical-error return codes and of the SYNAD exit routine 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 |

### LERAD Exit Routine: Logical Error Analysis

When a LERAD routine is provided, it gets control for logical errors, and register 15 doesn't contain 8, but contains the entry address of the LERAD routine.

The contents of the registers when VSAM exits to the LERAD routine are:

| 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. |

If a logical error occurs and no LERAD routine is 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.

The following list gives the logical-error return codes in the feedback field and explains what each one means.

| FDBK 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.<br><br>Detected by: IDA019RA, IDA019RD, IDA019RR, IDA019RY, IDA019R2, IDA019R4, IDA019R8 |
| 8 (8) | Attempt was made to store a record with a duplicate key.<br><br>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.<br><br>Detected by: IDA019RA, IDA019RR, IDA019RX, IDA019R4 |
| 16 (10) | Record not found.<br><br>Detected by:IDA019RA, IDA019RR, IDA019RY |
| 20 (14) | Record and its control interval already held in exclusive control by another requester.<br><br>Detected by: IDA019RF, IDA019RY, IDA019R2, IDA019R8 |
| 24 (18) | Record resides on a volume that can't be mounted.<br><br>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 is not enough space left in the data space to allocate 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.<br><br>Detected by: IDA019R5 |
| 32 (20) | An RBA was specified that doesn't give the address of any data record in the the data set.<br><br>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.<br><br>Detected by: IDA019RM |
| 40 (28) | Insufficient virtual storage in the address space to complete the request.<br><br>Detected by: IDA019RG, IDA019RU, IDA019RX |
| 44 (2C) | Work area not large enough for the data record (GET with OPTCD=MVE).<br><br>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.<br><br>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.<br><br>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 empty entry-sequenced or relative record data set.<br><br>Detected by: IDA019R1, IDA019R8 |

| FDBK 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.<br><br>Detected by: IDA019R1, IDA019R8 |
| 80 (50) | An ERASE request was issued for access to an entry-sequenced data set.<br><br>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.<br><br>Detected by: IDA019RQ, IDA019R1, IDA019R4, IDA019R8 |
| 88 (58) | A sequential GET or PUT request was issued without VSAM having been positioned for it, 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.<br><br>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.<br><br>Detected by: IDA019RQ, IDA019RX, IDA019R4, IDA019R8 |
| 96 (60) | Attempt was made to change a key during an update.<br><br>Detected by: IDA019RL, IDA019RX |
| 100 (64) | Attempt was made to change the length of a record during an addressed update.<br><br>Detected by: IDA019RL, IDA019RQ |
| 104 (68) | The RPL options are either invalid or conflicting in one of the following ways:<br>• SKP was specified and either KEY wasn't specified or BWD was specified<br>• BWD was specified for CNV processing<br>• FWD and LRD were specified<br>• Neither ADR, CNV, nor KEY was specified in the RPL<br>• WRTBFR, MRKBFR, or SCHBFR was issued, but either TRANSID was greater than 31 or a shared-resources option wasn't specified<br>• ICI processing was specified, but a request other than a GET or a PUT was issued<br><br>Detected by: IDA109RA, IDA019RR, IDA019RY, IDA019RX, IDA019R1, IDA019R4, IDA019R8 |
| 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.<br><br>Detected by: IDA019RL, IDA019RQ, IDA019RÚ, IDA019R4, IDA019R8 |
| 112 (70) | KEYLEN specified was too large or equal to 0.<br><br>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).<br><br>Detected by: IDA019RR, IDA019R4, IDA019R8 |
| 132 (84) | An attempt was made in locate mode to retrieve a spanned record.<br><br>Detected by: IDA019RT |
| 136 (88) | An addressed GET was issued for a spanned record in a key-sequenced data set.<br><br>Detected by: IDA019RT |

| FDBK Code | Condition |
|---|---|
| 140 (8C) | Inconsistent spanned-record segments.<br>Detected by: IDA019R4 |
| 144 (90) | Invalid pointer in an alternate index (no associated base record).<br>Detected by IDA019RX |
| 148 (94) | The maximum number of pointers in the alternate index has been exceeded.<br>Detected by: IDA019RU |
| 152 (98) | (Shared resources only.) Not enough buffers are available to process the request.<br>Detected by: IDA109RY |
| 192 (C0) | Invalid relative record number.<br>Detected by: IDA019RQ, IDA019RR |
| 196 (C4) | An addressed request was issued to a relative record data set.<br>Detected by: IDA019R1 |
| 200 (C8) | Addressed or control-interval access was attempted by way of a path.<br>Detected by: IDA019RX |
| 204 (CC) | PUT-insert requests are not allowed in backward mode.<br>Detected by: IDA019RQ, IDA019R4 |

## SYNAD Exit Routine: Physical Error Analysis

When a SYNAD routine is provided, it gets control for physical errors, and register 15 doesn't contain 12, but contains the entry address of the SYNAD routine.

The contents of the registers when VSAM exits to the SYNAD routine are:

| Reg | Contents |
|---|---|
| 0 | Unpredictable. |
| 1 | Address of the request parameter list that contains the feedback field the routine should examine and the address of the message area, if any. 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 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. |

If a physical error occurs and no SYNAD routine is 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; the message area contains more details about the error. Register 1 points to the request parameter list.

The physical-error return codes in the feedback field, and what each one indicates, are:

| FDBK 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 one sequence set of an index component. |

All physical errors are detected by IDA019R5.

Figure 76 gives the format of a physical-error message. The format and some of the contents of the message are purposely similar to the format and contents of the SYNADAF message, which is described in *OS/VS Data Management Macro Instructions.*

| Field | Bytes | Length | Discussion |
|---|---|---|---|
| Message Length | 0–1 | 2 | Binary value of 128 |
| | 2–3 | 2 | Unused (0) |
| Message Length - 4 | 4–5 | 2 | Binary value of 124 (provided for compatibility with SYNADAF message) |
| | 6–7 | 2 | Unused (0) |
| Address of I/O Buffer | 8–11 | 4 | The I/O buffer associated with the data in relation to which the error occurred |
| **The rest of the message is in printable format:** | | | |
| Date | 12–16 | 5 | YYDDD (year and day) |
| | 17 | 1 | Comma (,) |
| Time | 18–25 | 8 | HHMMSSTH (hour, minute, second, and tenths and hundredths of a second) |
| | 26 | 1 | Comma (,) |
| RBA | 27–34 | 8 | Relative byte address of the record in relation to which the error occurred. |
| | 35 | 1 | Comma (,) |
| 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 76 (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.<br><br>X'41'—<br>"INCORR LENGTH"<br>"UNIT EXCEPTION"<br>"PROGRAM CHECK"<br>"PROTECTION CHK"<br>"CHAN DATA CHK"<br>"CHAN CTRL CHK"<br>"INTFCE CTRL CHK"<br>"CHAINING CHK"<br>"UNIT CHECK"<br><br>*If the type of unit check can be determined, this message is replaced by one of the following:*<br>"CMD REJECT"<br>"INT REQ"<br>"BUS OUT CK"<br>"EQP CHECK"<br>"DATA CHECK"<br>"OVER RUN"<br>"TRACK COND CK"<br>"SEEK CHECK"<br>"COUNT DATA CHK"<br>"TRACK OVERRUN"<br>"CYLINDER END"<br>"INVALID SEQ"<br>"NO RECORD FOUND"<br>"FILE PROTECT"<br>"MISSING A.M."<br>"OVERFL INCP"<br><br>X'48'—"PURGED REQUEST"<br><br>X'4F'—"R.HA.RO. ERROR"<br><br>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 76 (Part 2 of 2). Format of Physical-Error Messages

## *Open, Close, and End-of-Volume Error Codes*

### (For EOV)

| Error Code Set in Register 15 | Meaning of Error Code |
|---|---|
| 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) | The IOB could not be locked |
| 16 (10) | The VSAM catalog could not be updated |

All End-of-Volume errors are detected by IDA0557A.

## (For OPEN/CLOSE/TCLOSE)

| Error Code Set in ACBERFLG Field of ACB | Meaning of Error Code |
|---|---|
| 0 (0) | When register 15 contains 0, all data sets were opened or closed successfully.<br><br>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 TCLOSE). |
| 96 (60) | Warning message: an unusable data set was opened for input.<br><br>Detected by: IDA0192B |
| 100 (64) | Warning message: Open encountered an empty alternate index that is part of an upgrade set.<br><br>Detected by: IDA0192B |
| 104 (68) | The timestamp for the volume does not match the timestamp in the volume catalog record. (This might mean the volume is not accurately described by its catalog record.)<br><br>Detected by: IDA0192A |
| 108 (6C) | 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.)<br><br>Detected by: IDA0192B |
| 116 (74) | The last request to close this data set was not completed successfully.<br><br>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.<br><br>Detected by: IDA0192F, IFG0200V |
| 136 (88) | Not enough storage was available for work areas, buffers, or control blocks.<br><br>Detected by: IDA0192A, IDA0192B, IDA0192C, IDA0192F, IDA0192W, IDA0192Y, IDA0192Z, IDA0200B, IDA0220T, IDA0231B, IDA0231T |
| 144 (90) | An I/O error occurred while reading or writing a catalog record. A return code was set by a VSAM catalog management routine.<br><br>Detected by: IDA0192C |
| 148 (94) | The catalog entry for the data set being opened or closed was not found.<br><br>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.<br><br>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 specified in the ACB's MACRF field and control-interval processing should be specified, but is not. Detected by: IDA0192A, IDA0192B, IDA0192C, IDA0192Z |

**(For OPEN/CLOSE/TCLOSE)**

| Error Code Set in ACBERFLG Field of ACB | Meaning of Error Code |
|---|---|
| 164 (A4) | The system detected an I/O error while reading the volume label and format-4 DSCB.<br><br>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.<br><br>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.<br><br>Detected by: IDA0192F |
| 180 (B4) | The requested VSAM master or user catalog does not exist or is not open.<br><br>Detected by: IDA0192C |
| 184 (B8) | An I/O error occurred during I/O processing.<br><br>Detected by: IDA0192B, IDA0200B, IDA0200T, IDA0231B, IDA0231T |
| 188 (BC) | The data set indicated by the access-method control block is not of the type that may be specified by an access-method control block.<br><br>Detected by: IDA0192Z,IDA0200B,IDA0231B |
| 192(C0) | An unusable data set was opened for output.<br><br>Detected by: IDA0192B |
| 196 (C4) | Access to data was requested by way of an empty path.<br><br>Detected by: IDA0192B |
| 200 (C8) | The volume is unusable.<br><br>Detected by: IDA0192F |
| 208 (D0) | The ACB MACRF specified GSR.<br><br>Detected by: IDA0192A |
| 212 (D4) | The ACB MACRF specified LSR, but the data set requires create processing.<br><br>Detected by: IDA0192B |
| 216 (D8) | The ACB MACRF specified LSR, but the key length of the data set exceeds the maximum key length specified in BLDVRP.<br><br>Detected by: IDA0192B |
| 220 (DC) | The ACB MACRF specified LSR, but the data set's control-interval size exceeds the size of the largest buffer specified in BLDVRP.<br><br>Detected by: IDA0192Z |
| 224 (E0) | The ACB MACRF specified ICI, but the data set requires create processing.<br><br>Detected by: IDA0192B |
| 228 (E4) | The ACB MACRF specified LSR, but the VSAM shared resource table doesn't exist.<br><br>Detected by: IDA0192A |
| 232 (E8) | Reset was specified for a nonreusable data set, but the data set is empty.<br><br>Detected by: IDA0192C |

**(For OPEN/CLOSE/TCLOSE)**

| Error Code Set in ACBERFLG Field of ACB | Meaning of Error Code |
|---|---|
| 236 (EC) | A stage or destage error occurred.<br>Detected by: IDA0192D |
| 240 (F0) | Format-4 DSCB and catalog time-stamp verification failed during volume mounting for output processing.<br>Detected by: IDA0192F |
| 244(F4) | The volume that contains the catalog recovery area wasn't mounted and verified for output processing.<br>Detected by: IDA0192F |

## *Catalog Management Error Codes*

Catalog mamagement sets error and reason codes in the CCAPROB field of the CCA (Catalog Communications Area). (For a description of the CCA, see "VSAM Control Block Descriptions" in the "Data Areas" section of this publication.) CCAPROB includes an identification of the catalog management module that set the code (CCAMODID), a reason code (CCAREASN), and a return code (CCACD1), which appears in register 15. Complete explanations of the error and return codes, together with the appropriate programmer responses, are given in the description of message IDC3009I in *OS/VS Message Library: VS2 System Messages.* Brief descriptions of the return codes are given below:

| Return Code Set in CCA's CCAPROB Field | Symbolic Name | Meaning of Return Code |
|---|---|---|
| 0 (0) | RCS | Operation completed successfully. |
| 4 (4) | RCCAT | An error occurred while performing open/close processing for a VSAM catalog or catalog recovery area. |
| 8 (8) | RCENT | Entry does not exist, if action is one that locates an entry; entry already exists, if action is one that adds an entry to a catalog. |
| 20 (14) | RCINSP | Not enough space is available in the catalog data set. Another extent cannot be obtained because there is no more space on the volume in which the catalog resides or the maximum number of extents has been reached. |
| 24 (18) | RCIOL | Permanent read error in VSAM catalog. |
| 28 (1C) | RCIONL | Permanent I/O error in VSAM catalog. |
| 32 (20) | RCINCPL | Error was detected in the catalog parameter list (CTGPL). |
| 36 (24) | RCDSNF | Data set was not found. |
| 40 (28) | RCVLSZ | Volume list or work area is too small—the required length value is returned in the feedback field. |
| 44 (2C) | RCVLSM | Work area is too small; system is unable to return required size. |
| 48 (30) | RCINFUNC | Operation is not a valid one. |
| 52 (34) | RCIOU | I/O error was detected on a user volume. An attempt to modify the VTOC of the volume on which a |

| Return Code Set in CCA's CCAPROB Field | Symbolic Name | Meaning of Return Code |
|---|---|---|
| | | user-specified data set is being defined or modified failed because of a read or write error. |
| 56 (38) | RCSEC | Password is incorrect. |
| 60 (3C) | RCINENT | Catalog record type is invalid. |
| 64 (40) | RCNAME | Data set or index catalog record associated with the cluster or alternate index catalog record was not found. |
| 68 (44) | RCNOSP | No space is available on a user volume. |
| 72 (48) | RCNUNIT | Unit is not available for mounting user volume or volume not mounted. |
| 76 (4C) | RCNUNIT | Unit is not available for mounting user volume or volume not mounted. |
| 80 (50) | RCRELOP | Invalid related object. The object specified in the RELATE parameter of the DEFINE command does not exist or is improper for the object being defined. |
| 84 (54) | RCDATE | Purge date has not expired. |
| 88 (58) | RCCRAOP | Error with a catalog recovery area define operation. |
| 92 (5C) | RCDSEXT | Data set has reached the maximum number of extents. |
| 94 (5E) | RCOBTAIN | An OS/VS DADSM Obtain request failed during a VSAM catalog delete request. |
| 96 (60) | RCSPANCK | Error in specifying key length, key position, or record size for an alternate index or a spanned cluster. |
| 98 (62) | RCRENAME | An unusual condition occurred during ALTER name of a unique or nonVSAM data set. |
| 102 (66) | RCSCRTCH | An OS/VS DADSM Scratch request failed during a VSAM delete request for a unique or nonVSAM data set. |
| 106 (6A) | RCNTFMT4 | A format-4 DSCB processing error was encountered. |
| 108 (6C) | RCINFNAM | Field name is invalid. |
| 112 (70) | RCINFPL | Field parameter list (CTGFL) contains invalid parameters. |
| 116 (74) | RCCATBAL | Catalog records are invalid. |
| 120 (78) | RCSYSFLD | User attempted to modify a system field or nonexistent field. |
| 124 (7C) | RCINCI | Control interval number is invalid. |
| 128 (80) | RCBLKVCK | User provided a work area outside his address space. |
| 132 (84) | RCINPTR | Pointer is not valid. |
| 136 (88) | RCMISPAR | Required parameter is missing. |
| 140 (8C) | RCINCNPM | Specified parameters are inconsistent or conflicting. |
| 144 (90) | RCINENTN | Entry name is invalid. |
| 148 (94) | RCVOLOWN | Volume is already owned by another VSAM catalog. |
| 152 (98) | RCDNECAT | User attempted to delete a catalog that is not empty. |
| 156 (9C) | RCNOSPSA | No space available to suballocate. |
| 160 (A0) | RCVNDSPD | Deletion of space object did not cause volume to be deleted. |
| 164 (A4) | RCINSSWA | Not enough storage is available for work area. |
| 168 (A8) | RCINVDTY | Specified device-type is not supported. |

| Return Code Set in CCA's CCAPROB Field | Symbolic Name | Meaning of Return Code |
|---|---|---|
| 172 (AC) | RCDUPNVL | Volume has duplicate data space name. |
| 176 (B0) | RCNSPVTC | No space available on VTOC for DSCB. |
| 180 (B4) | RCDSNFND | Data space was not found. |
| 184 (B8) | RCDSO | Data set is currently open, so the catalog record cannot be modified. |
| 188 (BC) | RCCATUNA | The catalog is unavailable. |
| 192 (C0) | RCMLRSZ | Maximum logical record length specified is greater than 32,761 for a nonspanned data set. |
| 196 (C4) | RCMCISZD | Data component control interval size specified is greater than 32,767. |
| 200 (C8) | RCMCISZI | Index component control interval size specified is greater than maximum block size of index device. |
| 204 (CC) | RCKEYINC | Key extends beyond end of record. |
| 208 (D0) | RCBUFSIZ | Buffer size is too small. |
| 212 (D4) | RCSIZCAL | Control interval size cannot be calculated. |
| 216 (D8) | RCVTCBAL | Volume's VTOC is invalid. |
| 220 (DC) | RCDOSVTC | DOS VTOC cannot be converted to OS/VS VTOC. |
| 224 (E0) | RCMXGRP | Catalog record has exceeded the maximum number of sets of fields allowed. |
| 226 (E2) | RCTSAUTH | Test authorization macro failed. |
| 228 (E4) | RCLOCKER | Error detected in time-of-day clock. |
| 230 (E6) | RCHIGH | VSAM catalog retrieve of a control interval failed to get a low-range record from the VSAM catalog. |
| 232 (E8) | RCSMFER | Error detected in SMF processing. |
| 234 (EA) | RCLEOD | End of data encountered while reading the low data key range of the VSAM catalog. |
| 236 (EC) | RCSMAPE | Error detected in scanning the space map. |
| 238 (EE) | RCNOUCEN | No user catalog entry in the master catalog for convert volume processing. |
| 240 (F0) | RCINDER | Required DD statement missing. |
| 242 (F2) | RCEFRMPH | A physical I/O error occurred during an erase of data set being deleted. |
| 244 (F4) | RCEF | Erase operation failed—DELETE operation was not performed. |
| 248 (F8) | RCVOLENT | The volume catalog record (identified with a caller-specified volume serial number) was not found. |
| 250 (FA) | RCEFRM | VSAM record management found a logical error during erase processing while deleting a VSAM data set. |
| 252 (FC) | RCEE | Error was detected, and the operation was not completed. |

### Alphabetic List of the Catalog Management Error Return Code Symbolic Names

| Name | Code | Name | Code |
|---|---|---|---|
| RCBLKVCK | 128(80) | RCKEYINC | 204(CC) |
| RCBUFSIZ | 208(D0) | RCLEOD | 234(EA) |
| RCCAT | 4(04) | RCLOCKER | 228(E4) |
| RCCATBAL | 116(74) | RCMCISZD | 196(C4) |
| RCCATUNA | 188(BC) | RCMCISZI | 200(C8) |
| RCCRAOP | 88(58) | RCMISPAR | 136(88) |
| RCDATE | 84(54) | RCMLRSZ | 192(C0) |
| RCDNECAT | 152(98) | RCMXGRP | 224(E0) |
| RCDOSVTC | 220(DC) | RCNAME | 64(40) |
| RCDSEXT | 92(5C) | RCNMNTD | 72(48) |
| RCDSNF | 36(24) | RCNOSP | 68(44) |
| RCDSNFND | 180(B4) | RCNOSPSA | 156(9C) |
| RCDSO | 184(B8) | RCNOUCEN | 238(EE) |
| RCDUPNVL | 172(AC) | RCNSPVTC | 176(B0) |
| RCEE | 252(FC) | RCNTFMT4 | 106(6A) |
| RCEF | 244(F4) | RCNUNIT | 76(4C) |
| RCEFRM | 250(FA) | RCOBTAIN | 94(5E) |
| RCEFRMPH | 242(F2) | RCRELOP | 80(50) |
| RCENT | 8(08) | RCRENAME | 98(62) |
| RCHIGH | 230(E6) | RCS | 0(0) |
| RCINCI | 124(7C) | RCSCRTCH | 102(66) |
| RCINCNPM | 140(8C) | RCSEC | 56(38) |
| RCINCPL | 32(20) | RCSIZCAL | 202(D4) |
| RCINDER | 240(F0) | RCSMFER | 232(E8) |
| RCINENT | 60(3C) | RCSPANCK | 96(60) |
| RCINENTN | 144(90) | RCSMAPE | 236(EC) |
| RCINFNAM | 108(6C) | RCSYSFLD | 120(78) |
| RCINFPL | 112(70) | RCTSAUTH | 226(E2) |
| RCINFUNC | 48(30) | RCVLSM | 44(2C) |
| RCINPTR | 132(84) | RCVLSZ | 40(28) |
| RCINSP | 20(14) | RCVNDSPD | 160(A0) |
| RCINSSWA | 164(A4) | RCVOLENT | 284(F8) |
| RCINVDTY | 168(A8) | RCVOLOWN | 148(94) |
| RCIOL | 24(18) | RCVTCBAL | 216(D8) |
| RCIONL | 28(1C) | | |
| RCIOU | 52(34) | | |

## *Control Block Manipulation Error 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:

| Code | Description |
|---|---|
| 0 | Successful completion. |
| 4 | An error has been detected. The error code in register 0 indicates the type of error. |
| 8 | Invalid use of the execute form of this macro instruction. Since the return code is set by the macro instruction 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 75 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 254, 241, 245, and 252 are protected with key 0; subpool 250 is unprotected—attributes of system subpools are described in *OS/VS2 Supervisor Logic*)
- The size of each storage block
- Whether each storage block is fixed in real storage by Open

| Storage Block | Contains | Pointed to by | Gotten in Subpool[1] | Size[6] | Fixed in Real Storage by Open? |
|---|---|---|---|---|---|
| *Blocks Related to the Job Step as a Whole* | | | | | |
| Sphere Block[2] | 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 | 1K or larger | No |
| Protected Sphere Block | AMBLs for the base cluster (path processing) and the alternate indexes in the upgrade set, HEBs | BIB | 252 | 1K or larger | No |
| *Blocks Related to a Particular Cluster* | | | | | |
| Buffer Block | I/O buffer | CMB | 250 | Length of buffer | No[5] |
| Upgrade Buffer Block[2] | I/O buffer | CMB | 250 | Length of buffer | No |
| DEB Block | DEB | CMB | 254[4] | Length of DEB | No |
| EDB Block | EDB | CMB | 252 | Length of EDB | No[5] |
| String Block[3] | BUFCs, CPAs, IOBs, PLHs, RPLs for path PLHs, WAXs for Path PLHs | CMB | 250 | 4K or larger | No[5] |
| Fixed String Block[3] | PFL | CMB | 254[4] | 1/2K or larger | No |
| Upgrade String Block[2] | BUFCs,CPAs,IOBs, PLHS | CMB | 250 | 4K or larger | No |
| Fixed Upgrade String Block | PFL | CMB | 254 | Length of PFL | No |
| User Block | AMBs, AMDSBs, ARDBs, BUFC headers, preformat BUFCs, preformat CPAs, IWAs | CMB | 250 | 1K or larger | No[5] |
| Protected User | LPMBs | CMB | 252 | 3 LPMBs, + 64 bytes for set sector table | No[5] |

[1] Subpool is 241 for a catalog or a catalog recovery area built in system storage.

[2] This block doesn't exist for a catalog or a catalog recovery area built in system storage.

[3] This block doesn't exist for processing with local shared resources.

[4] Subpool is 245 for a catalog or a catalog recovery area built in system storage.

[5] This block is fixed in real storage by Open if the user requests it for "fast path" (improved control-interval) processing.

[6] The sizes are given for time of allocation. After Open termination, excess storage will have been freed.

[7] This block doesn't exist with "fast path" (improved control interval) processing.

Figure 77. Storage Blocks used for Virtual-Storage Management.

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."

Figure 78 gives the interrelationship of these control blocks. It shows two storage blocks obtained for DEBs. Storage blocks are obrained 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.

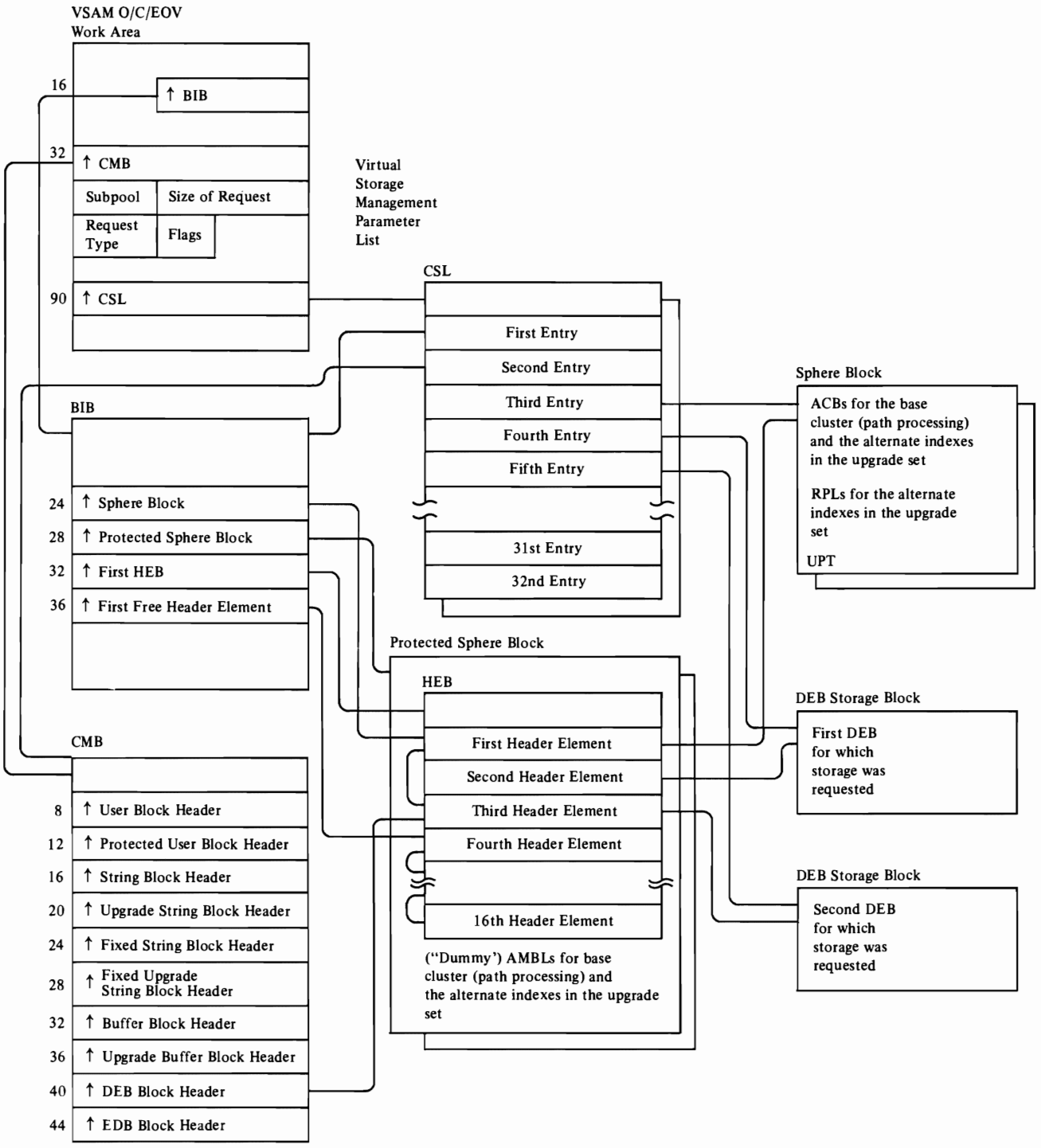



Figure 78. Virtual-Storage Management Control Block Structure

# GLOSSARY

## Abbreviations

Following is an alphabetized list of the abbreviations used in this book and in the VSAM code listings.

**ABEND:** abnormal end

**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

**AMCBS:** access method control block structure control block

**AMDSB:** access method data statistics block

**AMS:** Access Method Services

**AM/0:** Virtual Storage Access Method (VSAM)

**ARDB:** address range definition block

**ASPT:** available space pointer

**ATIOT:** alternate task input/output table

**BIB:** base information block

**BISAM:** basic indexed sequential access method

**BLK:** block, control interval

**BLPRM:** BDLVRP parameter list

**BSPH:** buffer subpool header

**BUFC:** buffer control block

**BWD:** backward (processing)

**CA:** control area

**CAXWA:** catalog auxiliary work area

**CCA:** catalog communication area

**CCB:** command control block

**CCR:** catalog control record

**CHKPT:** checkpoint

**CI:** control interval

**CICB:** ISAM interface control block

**CIDF:** control interval definition field

**CLW:** close work area

**CM:** catalog management

**CMB:** cluster management block

**CMS:** VSAM catalog management services

**CNV:** control interval or control-interval processing

**core:** virtual storage

**CPA:** channel program area

**CPL:** catalog parameter list (CTGPL)

**CRA:** catalog recovery area

**CSL:** core save list

**CTGCV:** catalog control volume

**CTGFL:** field parameter list (FPL)

**CTGFV:** field vector table (FVT)

**CTGPL:** catalog parameter list (CPL)

**CTGVL:** catalog volume list

**CTGWA:** catalog work area

**CVT:** communications 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

**ERFLG:** error flags

**ESDS:** entry-sequenced data set

**ESL:** enqueue save list

**EXCD:** exceptional conditions

**EXCP:** execute channel program

**EXLST:** exit list

**Ext Proc:** external procedure

**FKS:** full key search

**Fn:** format n

**FPL:** field parameter list (CTGFL)

**FS:** free space

**FVT:** field vector table (CTGFV)

**FWD:** forward (processing)

**GC:** type code (group code)

**GEN:** generic key search

**GO:** Set of fields (group occurrence)

**GOP:** Set-of-fields pointer (group occurrence pointer)

**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

**ISAM:** indexed sequential access method

**IXSPL:** index search parameter list

**JFCB:** job file control block

**JSCB:** job step control block

**KEQ:** search on key equal

**KEY:** keyed accessing

**KGE:** search on key greater or equal

**KSDS:** key-sequenced data set

**L:** link

**LLOR:** least length of record (that contains all key fields)

**LOC:** locate

**LPMB:** logical-to-physical mapping block

**LSR:** local shared resources

**MACR:** macro instruction 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/C/EOV:** Open/Close/End-of-Volume

**OFLG:** open flags

**OPTCD:** option code

**OPW:** Open work area

**OPWA:** (same as **OPW**)

**OPWRK:** (same as **OPW**)

**OS/VS:** operating system/virtual storage

**PCCB:** private catalog control block

**PFL:** page fix list

**PFPL:** PGFIX parameter list (same as **PFL**)

**PL/I:** programming language/one

**PL/S:** programming language/systems

**PLH:** placeholder

**PLHDR:** placeholder header

**PROC:** procedure

**PSR:** Programming Systems Representative

**PSW:** program status word

**PUT:** write-a-record command

**QISAM:** queued indexed sequential access method

**RAB:** record area block

**RB:** request block

**RBA:** relative byte address

**RDF:** record definition field

**REP:** replication

**RM:** record management

**Rn:** general-purpose register n

**RPL:** request parameter list

**RPLE:** RPL extension for ISAM interface processing

**RPS:** rotational position sensing

**RRDS:** relative record data set

**RTN:** routine

**SCIB:** search compressed index block

**SCRA:** catalog recovery area in system storage

**SEQ:** sequential or sequential processing

**SKP:** skip sequential or skip sequential processing

**SMF:** system management facilities

**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

**USVR:** user security verification routine

**VAT:** valid-AMBL table

**VL:** variable length

**VMT:** volume mount table

**VPL:** virtual page list

**VRP:** VSAM resource pool

**VSAM:** virtual storage access method

**VSL:** virtual subarea liast (same as PFL)

**VSRT:** VSAM shared resource table

**VTOC:** volume table of contents

**VVIC:** (replaced by **MSVI**)

**WAX:** work area for path processing

**WSHD:**

**WTG:** where-to-go table working storage header

**XCTL:** transfer control (macro instruction)

**XPT:** checkpoint

**XREF:** cross reference

# Definitions of Terms Used In This Book

The following terms are defined as they are used in this book. If you do not find the term you are looking for, refer to the index or to the *IBM Data Processing Glossary,* GC20-1699.

**Access Method Services:** A multifunction service program that 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 catalog record:** The first catalog record (control interval) that describes the VSAM object. This record contains the object's data set name, cluster name, or volume serial number in the ENTNAME field. This record also contains the header fields required for the object. The base catalog record can contain set-of-fields pointers that point to sets of fields in the base catalog record, or that point to sets of fields in extension records (vertical extension). The base catalog record's extension pointer can point to a control interval that continues the information (set-of-fields pointers) contained in the base catalog record (horizontal extension).

**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.)

**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:** A named, cataloged collection of stored records. The lowest member in the data structure hierarchy. A data set contains at least one component, and the component can contain no subsets.

**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.

**extension record:** The continuation of a catalog record that contains set-of-fields pointers and their sets of fields. Set-of-fields pointers in an extension record always point to sets of fields within the extension record. The extension record's extension pointer can point to a control interval that contains part of a set of fields too large to fit in the extension record (horizontal extension).

**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.

**group code:** (*See* type code.)

**group occurrence:** (*See* set of fields.)

**group occurrence pointer:** (*See* set-of-fields pointer.)

**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.)

**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.

**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.

**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.)

**set of fields:** A group of catalog record fields that contain related information. Sets of fields are referred to in the code as "group occurrences" or "GOs."

**set-of-fields pointer:** A field used to identify and locate a set of fields by its displacement from the beginning of the record's sets of fields (the set of fields is in the same control interval as the set-of-fields pointer) or by a control interval number (the set of fields pointer is in the base catalog record or its extension and the set of fields is in an extension record). Set-of-fields pointers are grouped by type code and are in ascending sequence by sequence number. Set-of-fields pointers are referred to in the code as "group occurrence pointers" or "GOPs."

**shared resources:** A set of functions that permit the sharing of a pool of I/O-related control blocks, channel programs, and buffers among several VSAM data sets open at the same time.

**skip sequential access:** Keyed sequential retrieval or storage of records here and there throughout a data set, skipping automatically to the desired record or collating position for insertion: VSAM scans the sequence set to find a record or a collating position.

**spanned record:** A record whose length exceeds control-interval lengthened and, as a result crosses, or spans, one or more control interval boundaries within a single control area.

**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.

**type code:** A code that identifies the set-of-fields type. Type codes are referred to in the code as "group codes" or "GCs." (*See* "Field Name Dictionary" for a list of type codes.)

**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/VS1 Master Index of Logic,* GY24-5164.

## A

# D

## E

## F



## G



## H

# I

## J



## K



## L

# M



# N



# O

# P

# Q



# R



# S

# T



# U



# V

## W



## X



## 123

OS/VS2 SVS IC: Virtual Storage Access Method Logic (File No. S370-30) Printed in U.S.A. SY26-3857-0

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 SVS IC: Virtual Storage
Access Method (VSAM) Logic
SY26-3857-0

**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 and 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 6090
San Jose, California

**Business Reply Mail**

No postage necessary if mailed in the U.S.A.

Postage will be paid by:

**IBM Corporation**
**P.O. Box 50020**
**Programming Publishing**
**San Jose, California 95150**

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 SVS IC: Virtual Storage Access Method Logic (File No. S370-30) Printed in U.S.A. SY26-3857-0