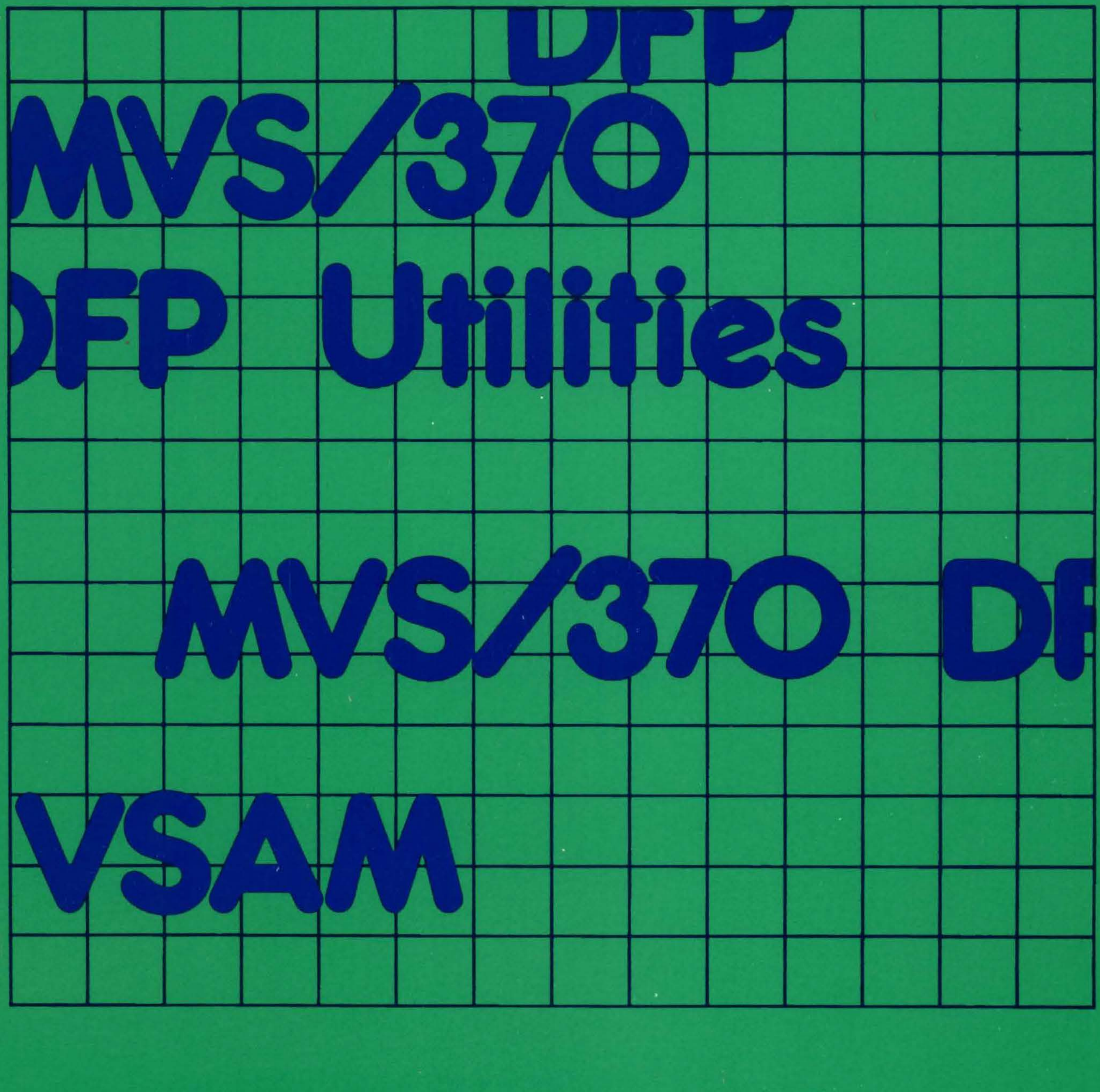
GC26-4074-2

MVS/370 VSAM Administration: Macro Instruction Reference

Release 1.2



IBM

GC26-4074-2

MVS/370
VSAM Administration:
Macro Instruction Reference

Release 1.2

Third Edition (May 1990)

This is a major revision of, and makes obsolete, GC26-4074-1.

This edition applies to Release 1.2 and Release 1.3 (available only in Brazil) of MVS/370 Data Facility Product, Licensed Program 5665-295, and to any subsequent releases until otherwise indicated in new editions or technical newsletters.

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

Changes are made periodically to this publication; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370, 30xx, 4300, and 9370 Processors Bibliography*, GC20-0001, for the editions that are applicable and current.

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

Requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality. If you request publications from the address given below, your order will be delayed because publications are not stocked there.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, P.O. Box 49023, Programming Publishing, San Jose, California, U.S.A. 95161-9023. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Preface

This book is intended to help you use VSAM macro instructions to process data.

Prerequisite Knowledge

Readers of this publication are assumed to have a programming background that includes:

- VSAM data management
- Catalog administration
- Job control language

You should be familiar with the information presented in the following publications:

- *MVS/370 VSAM Administration Guide*, GC26-4066, describes how to use VSAM. You should understand the information in the *VSAM Administration Guide* before you use this manual.
- *MVS/370 Catalog Administration Guide*, GC26-4053, describes the administration of tasks for catalogs and how to use the access method services commands to manipulate catalogs, and the objects cataloged in them.
- *MVS/370 JCL User's Guide*, GC28-1349, and *MVS/370 JCL Reference*, GC28-1350, describe the JCL parameters referred to in this publication and describes dynamic allocation.
- *MVS/370 Message Library: System Messages*, GC28-1374 and GC28-1375, provides a complete listing of the messages issued by VSAM.

Referenced Publications

Within the text, references are made to the publications listed in the following table:

Short Title	Publication Title	Order Number
Access Method Services Reference	*MVS/370 Integrated Catalog Administration: Access Method Services Reference*	GC26-4051
	MVS/370 VSAM Catalog Administration: Access Method Services Reference	GC26-4059
Catalog Administration Guide	*MVS/370 Catalog Administration Guide*	GC26-4053
Checkpoint/ Restart	*MVS/370 Checkpoint/Restart Users Guide*	GC26-4054
Data Areas	*OS/VS2 Data Areas*	SYB8-0606
Data Facility Product: Master Index	*MVS/370 Data Facility Product: Master Index*	GC26-4062

Short Title	Publication Title	Order Number
Data Facility Product: Planning Guide	*MVS/370 Data Facility Product: Planning Guide*	GC26-4052
Debugging Handbook	*MVS/370 System Programming Library: Debugging Handbook* Volumes 1 through 5	LC28-1385 through LC28-1389
Introduction to the IBM 3850 Mass Storage System	*Introduction to the IBM 3850 Mass Storage System (MSS)*	GA32-0038
JCL User's Guide	*MVS/370 JCL User's Guide*	GC28-1349
JCL Reference	*MVS/370 JCL Reference*	GC28-1350
Job Management	*OS/VS2 MVS System Programming Library: Job Management*	GC28-0627
OS/VS Mass Storage System Services: Reference Information	*OS/VS Mass Storage System (MSS) Services: Reference Information*	GC35-0017
RACF General Information	*Resource Access Control Facility (RACF): General Information*	GC28-0722
TSO Command Language Reference	*OS/VS2 TSO Command Language Reference*	GC28-0646
TSO Terminal User's Guide	*OS/VS2 TSO Terminal User's Guide*	GC28-0645
Supervisor Services and Macro Instructions	*OS/VS2 MVS System Programming Library: Supervisor Services and Macro Instructions*	GC28-0683
System Messages	*MVS/370 Message Library: System Messages* Volumes 1 and 2	GC28-1374 and GC28-1375
VSAM Administration Guide	*MVS/370 VSAM Administration Guide*	GC26-4066
VSAM Logic	*MVS/370 VSAM Logic*	LY26-3928

Notational Conventions

A uniform system of notation describes the syntax of VSAM macro instructions. This notation is not part of the language; it merely provides a basis for describing the structure of the macros.

The macro syntax illustrations in this book use the following conventions:

- Brackets [ ] indicate optional parameters.
- Braces { } indicate a choice of entry; unless a default is indicated, you must choose one of the entries.
- Items separated by a vertical bar (|) represent alternative items. No more than one of the items may be selected.
- An ellipsis (...) indicates that multiple entries of the type immediately preceding the ellipsis are allowed.
- Other punctuation (parentheses, commas, etc.) must be entered as shown.

- **BOLDFACE** type indicates the exact characters to be entered. Such items must be entered exactly as illustrated (in uppercase, except in TSO).
- *Italics* type specifies fields to be supplied by the user.
- ***BOLDFACE UNDERSCORED*** type indicates a default option. If the parameter is omitted, the underscored boldface value is assumed.
- A ‘ ’ in the macro syntax indicates that a blank (an empty space) must be present before the next parameter.

Summary of Changes

Library Refresh, May 1990

The list of GENCB, MODCB, SHOWCB, and TESTCB reason codes returned in register 0 has been updated. See Figure 5 on page 8.

The list of logical error reason codes in the feedback field of the request parameter list has been updated. See Figure 8 on page 13.

Enhancements have been added which enable you to process a linear data set (LDS) on MVS/370 DFP. The recognition of an LDS on the MVS/370 DFP system provides compatibility of LDS usage with MVS/XA™ DFP Version 2 Release 3.0. The enhancements added to MVS/370 DFP enable you to do the following:

- Process an LDS in control interval mode.
- Receive return and reason codes for logical errors that may occur while processing an LDS.

Information has been added to reflect service changes.

Release 1.1 Library Update, December 1985

Service Changes

The title of this publication has been changed from *MVS/370 VSAM Reference* to *MVS/370 VSAM Administration: Macro Instruction Reference*.

Many MVS/370 titles referred to in this publication have been changed.

Information has been added to reflect technical service changes.

MVS/XA is a trademark of the International Business Machines Corporation.

Contents

Figures

Chapter 1. Macro Instruction Return Codes and Reason Codes

This chapter describes the return codes you may get from the macro instructions that are used to open and close a data set, manage VSAM control blocks, and issue data processing requests.

VSAM sets reason codes in the ACB and the RPL. These reason codes are paired with return codes in register 15. (Register usage conventions are doumented in the *Data Administration: Macro Instruction Reference*.) Codes set in the ACB indicate open or close errors. Codes set in the RPL indicate record management errors.

The return codes and reason codes in this manual are listed in decimal and hexadecimal values. The decimal value is shown first, followed by the hexadecimal value in parentheses. Format descriptions and examples of each macro are in Chapter 2, "VSAM Macro Formats and Examples" on page 23.

Return Codes and Reason Codes from OPEN

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

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

If register 15 contains 4, 8, or 12, you can find out whether a VSAM data set had a warning message, or wasn't opened successfully and why, by issuing SHOWCB to display the ERROR field in each access method control block specified in OPEN. (See "SHOWCB Macro (Display Fields of an Access Method Control Block)" on page 99.) Figure 1 shows the possible reason codes that you may get from OPEN in the ERROR field in the access method control block. In addition to these reason codes, VSAM writes a message to the operator console and the programmer's listing to further explain the error. For a listing of VSAM messages, see *System Messages*.

Reason Code	Condition
0(0)	One of the following conditions exists: • VSAM is processing the access method control block for some other request. • The access method control block address is invalid.
76(4C)	Warning message: The interrupt recognition flag (IRF) was detected for a data set opened for input processing.
92(5C)	Warning message: Inconsistent use of CBUF processing. Sharing options differ between index and data components.
96(60)	Warning message: An unusable data set was opened for input.
100(64)	Warning message: OPEN encountered an empty alternate index that is part of an upgrade set.
104(68)	Warning message: The time stamp of the volume on which a data set is stored doesn't match the system time stamp in the data set's catalog record; this indicates that extent information in the catalog record may not agree with the extents indicated in the volume's VTOC.
108(6C)	Warning message: The time stamps of a data component and an index component do not match; this indicates that either the data or the index has been updated separately from the other.
116(74)	Warning message: The data set was not properly closed and either OPEN's implicit verify was unsuccessful or the user specified that OPEN's implicit verify should not be executed. A previous VSAM program may have abnormally terminated. Data may be lost if processing continues; the access method services VERIFY command may be used to cause the data set to be properly closed. For a description of the VERIFY command, see *Access Method Services Reference*. In a cross-system shared DASD environment, a return code of 116 can have two meanings: (1) the data set was not properly closed, or (2) the data set is opened for output on another processor.
118(76)	Warning message: The data set was not properly closed but OPEN's implicit verify was successfully executed.
128(80)	DD statement for this access method control block is missing or invalid.
132(84)	One of the following errors occurred: • Not enough storage was available for work areas. • The required volume could not be mounted. • An uncorrectable I/O error occurred while VSAM was reading the job file control block (JFCB). • The format-1 DSCB or the catalog cluster record is invalid. • The user-supplied catalog name does not match the name on the entry. • The user is not authorized to open the catalog as a catalog.
136(88)	Not enough virtual storage space is available in your program's address space for work areas, control blocks, or buffers.
140(8C)	The catalog indicates this data set has an invalid physical record size.
144(90)	An uncorrectable I/O error occurred while VSAM was reading or writing a catalog record.

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

145(91)	An uncorrectable error occurred in the VSAM volume data set (VVDS).
148(94)	No record for the data set to be opened was found in the available catalog(s), or an unidentified error occurred while VSAM was searching the catalog. For the catalog return code, see system message IDC3009I in *System Messages*.
152(98)	Authorization checking has failed for the following reasons: • The password specified in the access method control block for a specified level of access doesn't match the password in the catalog of that level of access. • RACF failure. For the catalog return code, see system message IDC3009I in *System Messages*.
160(A0)	The operands specified in the ACB or GENCB macro are inconsistent with each other or with the information in the catalog record. With shared resources, this code can mean: MACRF options are inconsistent: LSR or GSR is specified with ICI, CBIC, or UBF (see "Using Control Interval Access with Shared Resources" in *VSAM Administration Guide*), or DFR is specified without LSR or GSR (see "Deferring Write Requests" in *VSAM Administration Guide*.) MACRF DFR is specified for a data set that was defined with SHAREOPTIONS 4 (see "Deferring Write Requests" in *VSAM Administration Guide*.)
164(A4)	An uncorrectable I/O error occurred while VSAM was reading the volume label.
168(A8)	The data set was not available for the type of processing you specified, or an attempt was made to open a reusable data set with the reset option while another user had the data set open. The data set may have the INHIBIT attribute specified. The data set cannot be opened for CBUF processing because it was already opened for non-CBUF processing. Or the data set has conflicting CBUF attributes for the data and index components of the ACB.
176(B0)	An error occurred while VSAM was attempting to fix a page of virtual storage in real storage.
180(B4)	A VSAM catalog specified in JCL either does not exist or is not open, and no record for the data set to be opened was found in any other catalog.
184(B8)	An uncorrectable I/O error occurred while VSAM was completing an I/O request.
188(BC)	The data set indicated by the access method control block is not of the type that may be specified by an access method control block.
192(C0)	An unusable data set was opened for output.
193(C1)	The interrupt recognition flag (IRF) was detected for a data set opened for output processing.
196(C4)	Access to data was requested via an empty path.
200(C8)	The Format-4 DSCB indicates that the volume is unusable. There was an error in CONVERTV to convert the volume from either real to virtual or virtual to real.

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

204(CC)	The ACB MACRF specification is GSR and caller is not operating in supervisor protect key 0 to 7, or ACB MACRF specification is CBIC (Control Blocks in Common) and caller is not operating in supervisor state with protect key 0 to 7.
205(CD)	The ACBCATX option or VSAM volume data set OPEN was specified and the calling program was not authorized.
208(D0)	The ACB MACRF specification is GSR and caller is using an OS/VS1 system.
212(D4)	The ACB MACRF specification is GSR or LSR and the data set requires load mode processing.
216(D8)	The ACB MACRF specification is GSR or LSR and the key length of the data set exceeds the maximum key length specified in BLDVRP.
220(DC)	The ACB MACRF specification is GSR or LSR and the data set's control interval size exceeds the size of the largest buffer specified in BLDVRP.
224(E0)	Improved control interval processing is specified and the data set requires load mode processing.
228(E4)	The ACB MACRF specification is GSR or LSR and the VSAM shared resource table (VSRT) does not exist (no buffer pool is available).
232(E8)	Reset was specified for a nonreusable data set and the data set is not empty.
236(EC)	A permanent staging error occurred in MSS (ACQUIRE).
240(F0)	Format-4 DSCB and volume timestamp verification failed during volume mount processing for output processing.
244(F4)	The volume containing the catalog recovery area was not mounted and not verified for output processing.

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

Return Codes and Reason Codes from CLOSE

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

Return Code	**Condition**
0(0)	All data sets were closed successfully.
4(4)	At least one data set (VSAM or non-VSAM) was not closed successfully.

If register 15 contains 4, you can use SHOWCB to display the ERROR field in each access method control block to find out whether a VSAM data set wasn't closed successfully and why not. (See "SHOWCB Macro (Display Fields of an Access Method Control Block)" on page 99.) Figure 2 on page 5 gives the reason codes that the ERROR field may contain following CLOSE. In addition to these reason codes, VSAM writes a message to the operator's console and the programmer's listing to further explain the error. For a listing of these messages, see *System Messages*.

Return Code	Condition
0(0)	No error (set when register 15 contains 0).
4(4)	The data set indicated by the access method control block is already closed.
129(81)	TCLOSE was issued against a media manager's structure.
132(84)	An uncorrectable I/O error occurred while VSAM was reading the job file control block (JFCB).
136(88)	Not enough virtual storage was available in your program's address space for a work area for CLOSE.
144(90)	An uncorrectable I/O error occurred while VSAM was reading or writing a catalog record.
145(91)	An uncorrectable error occurred in the VSAM volume data set (VVDS).
148(94)	An unidentified error occurred while VSAM was searching the catalog.
184(B8)	An uncorrectable I/O error occurred while VSAM was completing outstanding I/O requests.
236(EC)	A permanent destaging error occurred in MSS (RELINQUISH). With temporary CLOSE, a destaging error or a staging error (ACQUIRE) occurred.

Figure 2. CLOSE Reason Codes in the ERROR Field of the Access Method Control Block

OPEN/CLOSE Message Area for Multiple Reason or Warning Messages

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

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

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

- The message area header
- The message list

Message Area Header

The message area header contains statistical, pointer, and general information. Its contents are unrelated to the individual messages. The format of the message area header is shown in Figure 3.

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

Figure 3. Format of the Message Area Header

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

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

Message List

The message list contains individual messages corresponding to detected warning or error conditions. Bytes 16 through 19 of the message area header point to the location of the message list within the message area. If the message area header is not stored completely (bit 0 of byte 0 is 0), the location of the message list is not provided. Within the message list, individual messages are stored as a contiguous string of variable-length records. Bytes 14 and 15 of the message area header contain the number of messages stored. Check for a nonzero stored message count before investigating the message list. However, messages may not be stored even if the ACB error flag contains a nonzero value and the message area header bit 0 of byte 0 is 1. For example, no messages will be stored if MLEN is not large enough to allow at least one message to be stored.

The format of the individual messages is given in Figure 4.

Bytes 0-1 — Length of message including these two bytes.

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

Byte 3 — Function type code:

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

- X'00' No dsname stored. Bytes 4-47 of the message contain binary zeros. The error warning condition is not clearly related to a component, or VSAM was unable to identify or obtain the cluster name of the component in error. This code is used only if, in addition, the ddname of the ACB does not identify a valid DD statement or VSAM was unable to obtain the dsname contained in the DD statement.
- X'01' dsname contained in DD statement is stored. The error or warning condition is not clearly related to a component, or VSAM was unable to identify or obtain the cluster name of the component in error.
- X'02' dsname (cluster name) of base cluster stored. Error occurred during OPEN/CLOSE/TCLOSE for base cluster.
- X'03' dsname (cluster name) of alternate index component stored. Error occurred during OPEN/CLOSE/TCLOSE for alternate index component.
- X'04' dsname (cluster name) of member of upgrade set stored. Error occurred during OPEN/CLOSE/TCLOSE for this member of the upgrade set.

Bytes 4-47 — Binary zeros (function type code=X'00') or a dsname as described by byte 3.

Figure 4. Format of Individual Messages in Message List

Bytes 0 and 1 of each message specify the actual length of the individual message. You must inspect the length so that you can take the variable-length nature of the message into account in your processing.

Byte 2 of the message contains the ACB error flag code; it does not indicate that a dsname has been stored. Depending on the condition that raised the ACB error flag code, either no dsname or different types of dsnames (DD, base cluster, alternate index, or upgrade set member) may be stored. (The same condition may be detected both when opening the base cluster and when opening a member of the upgrade set. For example, an I/O error may occur when trying to obtain the dsname for the component in error.) Bytes 4 through 47 of the message can contain a dsname, but do not specify its type. Only byte 3 of the message specifies whether a dsname has been stored, and if so, its type.

Control Block Manipulation Macro Return Codes and Reason Codes

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

Return Code	Condition
0(0)	Task completed.
4(4)	Task not completed.
8(8)	An attempt was made to use the execute form of a macro to modify a keyword that isn't in the parameter list. (See Appendix A, "List, Execute, and Generate Forms of Macros" on page 121.)

An error can occur because you specified the operands incorrectly or, if you constructed a parameter list yourself, because the parameter list was coded incorrectly. See Appendix C, "Building Parameter Lists" on page 131, for an explanation of how to construct parameter lists for GENCB, MODCB, SHOWCB, and TESTCB.

When register 15 contains 4, register 0 contains a reason code indicating the reason VSAM couldn't perform the task. If you construct the parameter list yourself, you can get in register 0 reason codes 1, 2, 3, 10, 14, 20, and 21. Figure 5 describes each reason code that can be returned in register 0.

Figure 5 (Page 1 of 2). GENCB, MODCB, SHOWCB, and TESTCB Reason Codes Returned in Register 0

Reason Code	Applicable Macros[1]	Reason VSAM Couldn't Perform the Task
1(1)	G,M,S,T	The request type (generate, modify, show, or test) is invalid.
2(2)	G,M,S,T	The block type (access method control block, exit list, or request parameter list) is invalid.
3(3)	G,M,S,T	One of the keyword codes in the parameter list is invalid.
4(4)	M,S,T	The block at the address indicated is not of the type you indicated (access method control block, exit list, or request parameter list).
5(5)	S,T	Access method control block fields were to be shown or tested, but the data set is not open or it is not a VSAM data set.
6(6)	S,T	Access method control block information about an index was to be shown or tested, but no index was opened with the data set.
7(7)	M,S	An exit list was to be modified, but the list was not large enough to contain the new entry; or an exit was to be modified or shown but the specified exit wasn't in the exit list. (With TESTCB, if the specified exit address isn't present, you get an unequal condition when you test for it.)
8(8)	G	There isn't enough virtual storage in your program's address space to generate the access method control block(s), exit list(s), or request parameter list(s) and no work area outside your address space was specified.

Figure 5 (Page 2 of 2). GENCB, MODCB, SHOWCB, and TESTCB Reason Codes Returned in Register 0

Reason Code	Applicable Macros[1]	Reason VSAM Couldn't Perform the Task
9(9)	G,S	The work area specified was too small for generation or display of the indicated control block or fields.
10(A)	G,M	With GENCB, exit list control block type was specified and you specified an exit without without giving an address. With MODCB, exit list control block type was specified and you specified an exit without giving an address; in this case, either active or inactive must be specified, but load cannot be specified.
11(B)	M	Either (1) a request parameter list was to be modified, but the request parameter list defines an asynchronous request that is active (that is, no CHECK or ENDREQ has been issued on the request) and thus cannot be modified; or (2) MODCB is already issued for the control block, but hasn't yet completed.
12(C)	M	An access method control block was to be modified, but the data set identified by the access method control block is open and thus cannot be modified.
13(D)	M	An exit list was to be modified, and you attempted to activate an exit without providing a new exit address. Because the exit list indicated does not contain an address for that exit, your request cannot be honored.
14(E)	G,M,T	One of the option codes (for MACRF, ATRB, or OPTCD) has an invalid combination of option codes specified (for example, OPTCD=(ADR, SKP)).
15(F)	G,S	The work area specified did not begin on a fullword boundary.
16(10)	G,M,S,T	A VTAM keyword or subparameter was specified but the AM=VTAM parameter was not specified. AM = VTAM must be specified in order to process a VTAM version of the control block. AM = VTAM was specified but the control blocksubtype was not VTAM.
19(13)	M,S,T	A keyword was specified which refers to a field beyond the length of the control block located at the address indicated. (For example, a VTAM keyword was specified, but the control block pointed to was a shorter, non-VTAM block.)
20(14)	S	Keywords were specified which apply only if MACRF=LSR or GSR.
21(15)	S,T	The block to be displayed or tested does not exist because the data set is a dummy data set.
22(16)	S	AM=VTAM was specified and the RPL FIELDS parameter conflicts with the RPLNIB bit status. Either RPL FIELDS=NIB was specified and the RPLNIB bit was off, or RPL FIELDS=ARG was specified and the RPLNIB bit was on.
23(17)	G	The value specified in the length parameter exceeds the 65,535 byte limit.

Note:

[1] G=GENCB, M=MODCB, S=SHOWCB, T=TESTCB

Record Management Return Codes and Reason Codes

The following record management macros give return codes and reason codes in the feedback field of the RPL: GET, PUT, POINT, ERASE, CHECK, ENDREQ, GETIX, PUTIX, ACQRANGE, CNVTAD, MNTACQ, MRKBFR, SCHBFR, and WRTBFR.

The feedback field in the RPL consists of four bytes.

☒	Return Code (Register 15)	Function Code (FTNCD Code)	FDBK Code

For more information on the RPL feedback word, see *VSAM Logic*.

Return Codes

The meaning of the return code depends on whether processing is asynchronous or synchronous.

Asynchronous Request

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

Return Code	**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.

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

Synchronous Request

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

Return Code	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.

Component Codes

When a logical or physical error occurs, VSAM uses the component code field of the RPL to identify the component being processed when the error occurred and indicates whether the alternate index upgrade set is correct following the request that failed. The component code can be displayed and tested by using the SHOWCB and TESTCB macros. The codes and their meanings are given in Figure 6.

Figure 6. Component Codes Provided in the RPL

Component Code	What Was Being Processed	Upgrade Set Status
0(0)	Base cluster	Correct
1(1)	Base cluster	May be incorrect
2(2)	Alternate index	Correct
3(3)	Alternate index	May be incorrect
4(4)	Upgrade set	Correct
5(5)	Upgrade set	May be incorrect

Reason Codes

Paired with the 0, 8, and 12 return codes in register 15 are reason codes in the feedback field of the request parameter list.

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

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

Reason Code (Successful Request)

Successful completion of a VSAM request is defined as register 15=0 when the request is completed. The reason code field in the feedback word of the RPL may not be zero for a variety of reasons. Figure 7 lists these codes and the reasons they are set.

Reason Code When Register 15=0(0)	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 alternate key exists (applies only when accessing a data set using an alternate index that allows nonunique keys); for PUT requests, indicates that a duplicate key was created in an alternate index with the nonunique attribute.
12(C)	Write-buffer suggested (shared resources only).
16(10)	The sequence-set record does not have enough space to allow it to address all of the control intervals in the control area that should contain the record. The record was written into a new control area.
20(14)	Data set is not on virtual DASD for CNVTAD/MNTACQ/ACQRANGE request.
24(18)	Buffer found but not modified; no buffer writes performed.
28(1C)	Control interval split indicator was detected during an addressed GET NUP request.
32(20)	Request deferred for a resource held by the terminated RPL is asynchronous and cannot be restarted by TERMRPL.
36(24)	Possible data set error condition was detected by TERMRPL: • The request was abnormally terminated in the middle of its I/O operation. • One of the data/index BUFCs of the string contains data that needs to be written (BUFCMW=ON) but it was invalidated by TERMRPL.
40(28)	Error in PLH data BUFC pointer was detected by TERMRPL.

Figure 7. Successful Completion Reason Codes in the Feedback Field of the Request Parameter List

Reason Code (Logical Errors)

If a logical error occurs and you have no LERAD routine (or the LERAD exit is inactive), VSAM returns control to your program following the last executed instruction. ("User-Written-Exit Routines" in *VSAM Administration Guide* describes the LERAD routine.) The return code in register 15 indicates a logical error (8), and the feedback field in the request parameter list contains a reason code identifying the error. Register 1 points to the request parameter list. Figure 8 on page 13 gives the reason codes in the feedback field and explains the meaning of each.

Reason Code When Register 15=8(8) — **Condition**

4(4) — End of data set encountered (during sequential or skip sequential retrieval), or the search argument is greater than the high key of the data set. Either no EODAD routine is provided, or one is provided, returned to VSAM, and the processing program issued another GET. ("User-Written-Exit Routines" in *VSAM Administration Guide* describes the EODAD routine.)

8(8) — You attempted to store a record with a duplicate key, or there is a duplicate record for an alternate index with the unique key option.

12(C) — A key sequence check was performed and an error was detected in one of the following processing conditions:

- For a key-sequenced data set
  - PUT sequential or skip-sequential processing
  - GET sequential, single string input only
  - GET skip-sequential processing and the previous request is not a POINT
- For a relative record data set
  - GET skip-sequential processing
  - PUT skip-sequential processing

16(10) — Record not found, or the RBA is not found in the buffer pool.

20(14) — The RBA is found, but the buffer is under the exclusive control of another request. With this condition, it is possible to also have buffers invalidated. Or, the control interval is for a record already held in exclusive control by another requester.

Note: If the RPL message area is correctly specified, the following information is returned:

Offset	Length	Discussion
0	4	Address of RPL in exclusive control
4	1	Flag Byte: X'00' neither RPL is doing a control area split X'01' current RPL is attempting a control area split X'02' other RPL is doing a control area split

Figure 8 (Part 1 of 5). Logical Error Reason Codes in the Feedback Field of the Request Parameter List

Reason Code When Register 15=8(8)	**Condition**
24(18)	Record resides on a volume that can't be mounted.
28(1C)	Data set cannot be extended because VSAM can't allocate additional direct access storage space. Either there is not enough space left to make the secondary allocation request or you attempted to increase the size of a data set while processing with SHAREOPTIONS=4 and DISP=SHR.
32(20)	You specified an RBA that doesn't give the address of any data record in the data set.
36(24)	Key ranges were specified for the data set when it was defined, but no range was specified that includes the record to be inserted.
40(28)	Insufficient virtual storage in your address space to complete the request.
44(2C)	Work area not large enough for the data record or for the buffer (GET with OPTCD=MVE).
48(30)	Invalid options, data set attributes, or processing conditions specified for TERMRPL request: • CNV processing • The specified RPL is asynchronous • Chained RPLs • Path processing • Shared resources (LSR/GSR) • Load mode • Relative record data set • Data set contains spanned records • User not in key 0 and supervisor state • End-of-volume in process (secondary allocation)
52(34)	The previous request was TERMRPL.
64(40)	There is insufficient storage available to dynamically add another string. Or, the maximum number of placeholders that may be allocated to the request has been allocated, and a placeholder is not available.
68(44)	You attempted to use a type of processing (output or control interval processing) that was not specified when the data set was opened.
72(48)	You made a keyed request for access to an entry-sequenced data set, or you issued a GETIX or PUTIX to an entry-sequenced or relative record data set.
76(4C)	You issued an addressed or control interval PUT to add to a key-sequenced data set, or you issued a control interval PUT to a relative record data set.
80(50)	You issued an ERASE request in one of the following situations: • For access to an entry-sequenced data set. • For access to an entry-sequenced data set via a path. • With control interval access.

Figure 8 (Part 2 of 5). Logical Error Reason Codes in the Feedback Field of the Request Parameter List

Reason Code When Register 15=8(8)	Condition
84(54)	You specified OPTCD=LOC in one of the following situations: • For a PUT request. • In the previous request parameter list in a chain of parameter lists. • For UBF processing.
88(58)	You issued a sequential GET request without having caused VSAM to be positioned for it, or you changed from addressed access to keyed access without causing VSAM to be positioned for keyed-sequential retrieval; there was no positioning established for sequential PUT insert for a relative record data set, or you attempted an illegal switch between forward and backward processing.
92(5C)	You issued a PUT for update or an ERASE without a previous GET for update, or a PUTIX without a previous GETIX.
96(60)	You attempted to change the prime key or key of reference while making an update.
100(64)	You attempted to change the length of a record while making an addressed update.
104(68)	The RPL options are either invalid or conflicting in one of the following ways: • SKP was specified and either KEY was not specified or BWD was specified. • BWD was specified for CNV processing. • FWD and LRD were specified. • Neither ADR, CNV, nor KEY was specified in the RPL. • BFRNO is invalid (less than 1 or greater than the number of buffers in the pool). • WRTBFR, MRKBFR, or SCHBFR was issued, but either TRANSID was greater than 31 or the shared resource option was not specified. • ICI processing was specified, but a request other than a GET or a PUT was issued. • MRKBFR MARK=OUT or MARK=RLS was issued but the RPL did not have a data buffer associated with it. • The RPL specified WAITX, but the ACB did not specify LSR or GSR.
108(6C)	RECLEN specified was larger than the maximum allowed, equal to 0, or smaller than the sum of the length and the displacement of the key field; RECLEN was not equal to record (slot) size specified for a relative record data set. The automatic increase in the record size of an upgrade index for the base cluster may cause an incorrect RECLEN specification.

Figure 8 (Part 3 of 5). Logical Error Reason Codes in the Feedback Field of the Request Parameter List

Reason Code When Register 15=8(8)	Condition
112(70)	KEYLEN specified was too large or equal to 0.
116(74)	During initial data set loading (that is, when records are being stored in the data set the first time it's opened), GET, POINT, ERASE, direct PUT, skip-sequential PUT, or PUT with OPTCD=UPD is not allowed. For initial loading of a relative record data set, the request was other than a PUT insert.
120(78)	The request was operating under an incorrect TCB. For example, an end-of-volume call or a GETMAIN would have been necessary to complete the request, but the request was issued from a job step other than the one that opened the data set. The request can be resubmitted from the correct task, if the new request reestablishes positioning.
124(7C)	A request was cancelled for a user JRNAD exit.
128(80)	A loop exists in the index horizontal pointer chain during index search processing.
132(84)	An attempt was made in locate mode to retrieve a spanned record.
136(88)	You attempted an addressed GET of a spanned record in a key-sequenced data set.
140(8C)	The spanned record segment update number is inconsistent.
144(90)	Invalid pointer (no associated base record) in an alternate index.
148(94)	The maximum number of pointers in the alternate index has been exceeded.
152(98)	Not enough buffers are available to process your request (shared resources only).
156(9C)	An invalid control interval or invalid record definition field was detected during keyed processing, or an addressed GET UPD request failed because the control interval flag was on. The RPL contains the invalid control interval's RBA.
160(A0)	One or more candidates were found that have a modified buffer marked to be written. The buffer was left in write status with valid contents. With this condition, it is possible to have other buffers invalidated or found under exclusive control.
164(A4)	One of the following invalid options was specified for a CNVTAD/MNTACQ/ACQRANGE request: • Generic key (GEN) • Load mode • Path processing • User buffers (UBF) with LSR/GSR • Key-sequenced data set, but not key processing (KEY) • Entry-sequenced data set, but not address processing (ADR) • Relative record data set, but not key processing (KEY) • RPL is chained • Key-sequenced data set has single-level imbedded index

Figure 8 (Part 4 of 5). Logical Error Reason Codes in the Feedback Field of the Request Parameter List

Reason Code When Register 15=8(8)	Condition
168(A8)	One of the following user parameter list errors was detected for CNVTAD/MNTACQ/ACQRANGE request: • No user parameter list is specified (RPLARG=0) • Argument count is zero for CNVTAD/MNTACQ request • Ending argument is less than starting argument for ACQRANGE request • Parameter list not on word boundary
172(AC)	ACQUIRE error returned by SVC 126 for MNTACQ/ACQRANGE request.
176(B0)	Staging failure for MNTACQ/ACQRANGE request.
180(B4)	RBA/volume error for MNTACQ/ACQRANGE request. (Required volume not mounted or specified RBA(s) not on mounted volume.)
184(B8)	Catalog errors returned from SVC 126 for CNVTAD request.
188(BC)	Storage for ACQUIRE ECBs (subpool 241) is not available.
192(C0)	Invalid relative record number.
196(C4)	You issued an addressed request to a relative record data set.
200(C8)	You attempted addressed or control interval access through a path.
204(CC)	PUT insert requests are not allowed in backward mode.
208(D0)	The user has issued an ENDREQ macro instruction against an RPL that has an outstanding WAIT against the ECB associated with the RPL. This can occur when an ENDREQ is issued from a STAE or ESTAE routine routine against an RPL that was started before the abend. No ENDREQ processing has been done.
212(D4)	During control area split processing, a condition exists that prevents the split of the index record. Index and/or Data control interval size may need to be increased.
224(E0)	MRKBFR OUT was issued for a buffer with invalid contents.
228(E4)	Caller in cross-memory mode is not in supervisor state or RPL of caller in SRB or cross-memory mode does not specify SYN processing.
232(E8)	UPAD error; ECB was not posted by user in cross-memory mode.
236(EC)	Validity check error for SHAREOPTIONS 3 or 4.
240(F0)	For shared resources, one of the following is being performed: (a) an attempt is being made to obtain a buffer in exclusive control, (b) a buffer is being invalidated, or (c) the buffer use chain is changing. For more detailed feedback, reissue the request.
252(FC)	Record mode access not valid for an LDS.
253(FD)	VERIFY function not valid for an LDS.

Figure 8 (Part 5 of 5). Logical Error Reason Codes in the Feedback Field of the Request Parameter List

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

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

Positioning Following Logical Errors

VSAM is unable to maintain positioning after every logical error. Whenever positioning is not maintained following an error request, you must reestablish it before processing resumes.

Positioning may be in one of four states following a POINT or a direct request that encountered a logical error:

Yes VSAM is positioned at the position in effect before the request in error was issued.

No VSAM is not positioned, because no positioning was established at the time the request in error was issued.

New VSAM is positioned at a new position.

U VSAM is positioned at an unpredictable position.

The following table shows which positioning state applies to each reason code listed for sequential, direct, and skip-sequential processing. "N/A" indicates that the reason code is not applicable to the type of processing indicated.

Reason Code When Register 15=8(8)	Sequential	Direct	Skip-Sequential
4(4)	Yes	N/A	Yes
8(8)[1]	Yes	No	New
12(C)	Yes	N/A	Yes
16(10)	No	No	No
20(14)	U	No[2]	No[2]
24(18)	Yes	No	No
28(1C)	Yes	No	Yes
32(20)	No	No	N/A
36(24)	Yes	No	New
40(28)	Yes	No	No
44(2C)	Yes	New	Yes
64(40)	No	No	No
68(44)	Yes	Yes	Yes
72(48)	Yes	Yes	Yes
76(4C)	Yes	Yes	Yes
80(50)	Yes	Yes	Yes
84(54)	Yes	Yes	Yes
88(58)	Yes	Yes	Yes
92(5C)	Yes	Yes	Yes

96(60)	Yes	Yes	Yes
100(64)	Yes	Yes	Yes
104(68)	Yes	New	Yes
108(6C)	Yes	New	Yes
112(70)	Yes	Yes	Yes
116(74)	Yes	Yes	Yes
120(78)	Yes	No	No
124(7C)	No	No	No
132(84)	Yes	New	Yes
136(88)	No	No	N/A
140(8C)	Yes	New	Yes
144(90)	Yes	Yes	Yes
148(94)	Yes	Yes	Yes
152(98)	Yes	No	No
156(9C)	Yes	No	No
160(A0)	N/A	No	N/A
192(C0)	Yes	Yes	Yes
196(C4)	Yes	Yes	Yes
200(C8)	Yes	Yes	Yes
204(CC)	Yes	Yes	Yes
208(D0)	Yes	Yes	Yes
224(E0)	N/A	No	N/A
228(E4)	No	No	No
232(E8)	No	No	No
236(EC)	No	No	No
240(F0)	Yes	Yes	Yes

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

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

Reason Code (Physical Errors)

If a physical error occurs and you have no SYNAD routine (or the SYNAD exit is inactive), VSAM returns control to your program following the last executed instruction. The return code in register 15 indicates a physical error (12), and the feedback field in the request parameter list contains an reason code identifying the error; the RPL message area contains more details about the error. Register 1 points to the request parameter list. The RBA field in the request parameter list gives the relative byte address of the control interval in which the physical error occurred. Figure 9 gives the reason codes in the feedback field and explains what each indicates.

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

Figure 9. Physical Error Reason Codes in the Feedback Field of the Request Parameter List

Figure 10 on page 20 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 *Data Administration: Macro Instruction Reference*.

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 where 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 where the error occurred
	35	1	Comma (,)
Component Type	36-41	6	"DATA" or "INDEX"
	42	1	Comma (,)
Volume Serial Number	43-48	6	Volume serial number of the volume where the error occurred
	49	1	Comma (,)
Job Name	50-57	8	Name of the job where 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), where the error occurred
	71	1	Comma (,)
Device Type	72-73	2	The type of device where 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 where the error occurred
	83	1	Comma (,)
Channel	84-89	6	The channel command that caused the error in the first two bytes, followed by "_OP"
	90	1	Comma (,)
Message	91-105	15	Messages are divided according to ECB condition codes: X'41' "INCORR LENGTH" "UNIT EXCEPTION" "PROGRAM CHECK" "PROTECTION CHK" "CHAN DATA CHK"

Figure 10 (Part 1 of 2). Physical Error Message Format

Field	Bytes	Length	Discussion
			"CHAN CTRL CHK"
			"INTFCE CTRL CHK"
			"CHAINING CHK"
			"UNIT CHECK"
			If the type of unit check can be determined, the 'UNIT CHECK' message is replaced by one of the following:
			"CMD REJECT"
			"INT REQ"
			"BUS OUT CK"
			"EQP CHECK"
			"DATA CHECK"
			"OVER RUN"
			"TRACK COND CK"
			"SEEK CHECK"
			"COUNT DATA CHK"
			"TRACK OVERRUN"
			"CYLINDER END"
			"NO RECORD FOUND"
			"FILE PROTECT"
			"MISSING A.M."
			"OVERFL INCP"
			X'48' "PURGED REQUEST"
			X'4F' "R.HA.RO. ERROR"
			For any other ECB condition 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 10 (Part 2 of 2). Physical Error Message Format

Return Codes from Macros Used to Share Resources among Data Sets

VSAM has a set of macros that enables you to share I/O buffers, I/O related control blocks, and channel programs among VSAM data sets.

Return Codes from BLDVRP

VSAM returns a code in register 15 that indicates whether the BLDVRP request was successful:

Return Code	Condition
0(0)	VSAM completed the request.
4(4)	A resource pool already exists in the partition or address space (LSR) or in the system (GSR).
8(8)	There is not enough virtual storage space to satisfy the request. GETMAIN or ESTAE failed.
12(C)	Buffers cannot be fixed in real storage. PAGEFIX failed.
16(10)	TYPE=GSR is specified but the program that issued BLDVRP is not in supervisor state with protection key 0 to 7.
20(14)	STRNO is less than 1 or greater than 255.
24(18)	BUFFERS is specified incorrectly. A size or number is invalid.

Return Codes from DLVRP

VSAM returns a code in register 15 that indicates whether the DLVRP request was successful:

Return Code	Condition
0(0)	VSAM completed the request.
4(4)	There is no resource pool to delete.
8(8)	There is not enough virtual storage space to satisfy the request. GETMAIN or ESTAE failed.
12(C)	There is at least one open data set using the resource pool.
16(10)	TYPE=GSR is specified, but the program that issued DLVRP is not in supervisor state with protection key 0 to 7.

Return Codes from End-of-Volume

End-of-volume returns the following codes in register 15:

Return Code	Condition
0(0)	Successful.
4(4)	The requested volume could not be mounted.
8(8)	The requested amount of space could not be allocated.
12(C)	I/O operations were in progress when end-of-volume was requested.
16(10)	The catalog could not be updated.

Chapter 2. VSAM Macro Formats and Examples

This chapter contains the macro instruction formats and examples for the macro instructions.

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

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

For most programming applications, you can conveniently use register notation or absolute numeric expressions for numbers, character strings for names, and register notation or expressions that generate valid A-type address constants for addresses. Appendix B, "Operand Notation" on page 129, gives all the ways of coding each parameter for the macros that work at execution time.

You can write a reentrant program only with execution-time macros. Appendix A, "List, Execute, and Generate Forms of Macros" on page 121, describes alternative ways of coding these macros for reentrant programs. The standard form of these macros is described in this chapter.

ACB Macro (Generate an Access Method Control Block)

The syntax of the ACB macro is:

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

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

label
is 1 to 8 characters that provide a symbolic address for the access method control block that is assembled and also, if you omit the DDNAME parameter, serves as the ddname.

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

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

BUFND=*number*
specifies the number of I/O buffers VSAM is to use for transmitting data between virtual and auxiliary storage. A buffer is the size of a control

interval in the data component. The minimum number you may specify is 1 plus the number specified for STRNO (if you omit STRNO, BUFND must be at least 2, because the default for STRNO is 1). The number can be supplied by way of the JCL DD AMP parameter as well as by way of the macro. The default is the minimum number required. Note, however, that minimum buffer specification does not provide optimum sequential processing performance. Generally, the more data buffers specified, the better the performance. Note also that additional data buffers will benefit direct inserts or updates during control area splits and will benefit spanned record accessing. For more information, see "Optimizing Performance" in *VSAM Administration Guide*.

BUFNI=*number*

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

Additional index buffers will improve performance by providing for the residency of some or all of the high-level index, thereby minimizing the number of high-level index records to be retrieved from DASD for key-direct processing. For more information, see "Optimizing Performance" in *VSAM Administration Guide*.

BUFSP=*number*

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

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

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

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

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

Option	Meaning
SKP	Skip-sequential access to a key-sequenced or a relative record data set; used only with keyed access in a forward direction.
ICI	Processing is limited to improved control interval processing; access is faster because fewer processor instructions are executed.
NCI	Processing other than improved control interval processing.
IN	Retrieval of records of a key-sequenced, entry-sequenced, or a relative record data set; (not allowed for an empty data set). If the data set is password protected, you must supply the address of the read or higher-level password in the ACB PASSWD parameter.
OUT	Storage of new records in a key-sequenced, entry-sequenced, or relative record data set (not allowed with addressed access to a key-sequenced data set); update of records in a key-sequenced, entry-sequenced, or relative record data set; deletion of records from a key-sequenced data set or relative record data set. If the data set is password protected, you must supply the address of the update or higher-level password in the ACB PASSWD parameter.
NIS	Normal insert strategy.
SIS	Sequential insert strategy (split control intervals and control areas at the insert point rather than at the midpoint when doing direct PUTs); although positioning is lost and writes are done after each direct PUT request, SIS allows more efficient space usage when direct inserts are clustered around certain keys.
NRM	The object to be processed is the one named in the specified ddname.
AIX	The object to be processed is the alternate index of the path specified by ddname, rather than the base cluster via the alternate index.
NRS	Data set is not reusable.
RST	Data set is reusable (high-used RBA is reset to 0 during OPEN). If the data set is password protected, you must supply the address of the update or higher-level password in the ACB PASSWORD parameter.
NSR	Nonshared resources.
LSR	Local shared resources; each partition or address space may have one resource pool independently of other partitions or address spaces.
GSR	Global shared resources; all address spaces may have local and global resources pools, where tasks in an address space with a local resource pool may use either the local resource pool or the global resource pool.
NUB	Management of I/O buffers is left up to VSAM.
UBF	Management of I/O buffers is left up to the user; the work area specified by the RPL (or GENCB) AREA parameter is, in effect, the I/O buffer—VSAM transmits the contents of a control interval directly between the work area and direct access storage; valid when OPTCD=MVE and MACRF=CNV are specified; when ICI is specified, UBF is assumed.

Figure 11 (Part 2 of 2). MACRF Options

MAREA=*address*

specifies the address of an optional OPEN/CLOSE or TYPE=T option (CLOSE macro) message area. See "OPEN/CLOSE Message Area for Multiple Reason or Warning Messages" on page 5 for more information.

MLEN=*number*

specifies the length of an optional OPEN/CLOSE or TYPE=T option (CLOSE macro) message area. Default=0; maximum=32K. See "OPEN/CLOSE Message Area for Multiple Reason or Warning Messages" on page 5 for more information.

PASSWD=*address*

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

STRNO=*number*

specifies the number of requests requiring concurrent data set positioning VSAM is to be prepared to handle. The default is 1. A request is defined by a given request parameter list or chain of request parameter lists. See "RPL Macro (Generate a Request Parameter List)" on page 93 and "GENCB Macro (Generate a Request Parameter List)" on page 57 for information on request parameter lists. When records are loaded into an empty data set, the STRNO value in the access method control block must be 1.

VSAM dynamically extends the number of strings as needed by concurrent requests for this ACB, and this automatic extension can influence performance. The VSAM control blocks for the set of strings specified by STRNO are allocated on contiguous virtual storage, but this is not guaranteed for the strings allocated by dynamic extension. Dynamic string addition cannot be done when using the following options:

- Load mode
- ICI
- LSR or GSR

For STRNO, you could specify the total number of request parameter lists or chains of request parameter lists that you are using to define requests. (VSAM needs to remember only one position for a chain of request parameter lists.) However, each position beyond the minimum number that VSAM needs to be able to remember requires additional virtual storage space for:

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

- Internal control blocks and other areas

Example 1: ACB Macro

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

```
BLOCK     ACB     AM=VSAM,BUFND=4,  BLOCK gives symbolic
                  BUFNI=3,          address of the access
                  BUFSP=19456,      method control block.
                  DDNAME=DATASETS,
                  EXLST=EXITS,
                  MACRF=(KEY,DIR,SEQ,OUT),
                  PASSWD=FIELD,
                  STRNO=2
FIELD     DC      FL1'6',C'CHANGE'  The update password:
                                    CHANGE has 6 characters.
```

The ACB macro's parameters are:

- BUFND specifies four I/O buffers for data; BUFNI specifies three I/O buffers for index entries; and BUFSP specifies 19456 bytes of buffer space, enough space to accommodate control intervals of data that are 4096 bytes and control intervals of index entries that are 1024 bytes.
- DDNAME specifies that this access method control block is associated with a DD statement named DATASETS.
- EXLST specifies that the exit list associated with this access method control block is named EXITS.
- MACRF specifies keyed-direct and keyed-sequential processing for both insertion and update.
- PASSWD specifies the location, FIELD, of the password provided. FIELD contains the length of the password as well as the password itself.
- STRNO specifies that two requests will require concurrent positioning.

ACQRANGE Macro (Stage Data)

The syntax of the ACQRANGE macro is:

[*label*]	**ACQRANGE**	**RPL=***address*

RPL=*address*
specifies the address of the RPL that identifies your open data set and your argument range. RPL parameters that have meaning for ACQRANGE are as follows:

- **ACB=***address*
  identifies your VSAM data set.

- **ARG=***address*
  identifies your starting and ending arguments. Address points to a parameter list, aligned on a fullword boundary as follows:

  Key-sequenced data set:

Offset	Length	Contents
0	4	Feedback area: Address of an ECB WAIT list
4	K	Starting full argument (K = key length)
4+K	K	Ending full argument (K = key length)

  Entry-sequenced data set or relative record data set:

Offset	Length	Contents
0	4	Feedback area: Address of an ECB WAIT list
4	4	Starting RBA/RRN
8	4	Ending RBA/RRN

  The maximum number of argument pairs you may specify is one.

- **OPTCD=({ADR|KEY}**
  ,{ASY|SYN}
  ,{KEQ|KGE}
  ,FKS)

  ADR is valid for an entry-sequenced data set, error for key-sequenced data set or relative record data set.

  KEY is valid for key-sequenced data set and relative record data set, error for entry-sequenced data set.

  If ASY is specified, you cannot WAIT on the RPLECB field for MNTACQ or ACQRANGE. You use the address placed in the parameter list feedback area. This address points to a list of ECBs (in standard WAIT list format) which you may use in place of the RPLECB field.

  GEN is not supported; if specified, it will give an error indication.

All other OPTCD subparameters are not applicable, and, if specified, are ignored with no error indication.

Because your request may result in the staging of numerous cylinders, a single ECB is not sufficient for an asynchronous ACQRANGE request. The RPLECB field is inoperative for the ACQRANGE interface. Upon return from an asynchronous ACQRANGE, the feedback area of the ACQRANGE parameter contains the address of a standard ECB WAIT list. You must then use this list in conjunction with the WAIT macro or you may use the list in conjunction with the EVENTS macro of MVS. An asynchronous request must conclude with either CHECK, ENDREQ, or CLOSE. The parameter list cannot be reused until the CHECK, ENDREQ, or CLOSE is completed.

At the conclusion of this macro, the RPL is disconnected. Any positioning in effect prior to execution of ACQRANGE will be lost. You may have to reposition. Chained RPLs are not supported by this macro.

BLDVRP Macro (Build VSAM Resource Pool)

The syntax of the BLDVRP macro is:

BLDVRP	**BUFFERS**=(*size*(*number*),*size*(*number*),...) [,**FIX**={**BFR**|**IOB**|(**BFR**,**IOB**)}] [,**KEYLEN**=*length*] ,**STRNO**=*number* [,**TYPE**={**LSR**|**GSR**}]

The BLDVRP macro has a standard form and list and execute forms. The standard form builds a parameter list and passes control to VSAM to build the resource pool. The list and execute forms are described in Appendix A, "List, Execute, and Generate Forms of Macros" on page 121.

BUFFERS=(*size*(*number*),*size*(*number*),...)
specifies the size and number of buffers in each buffer pool in the resource pool. The number of buffer pools in the resource pool is implied by the number of size(number) pairs you specify.

When you process a key-sequenced data set, the index component, as well as the data component, shares the buffers of a buffer pool. When you use an alternate index to process a base cluster, the components of the alternate index and the base cluster share buffers. The components of alternate indexes in an upgrade set share buffers. Buffers of the appropriate size and number must be provided for all these components, each of which uses the buffer pool whose buffers are exactly the right size or the next larger size.

Note: LSR/GSR users can ensure buffer pool selection by explicitly defining data and index control interval size(s).

size
is 512, 1024, 2048, 4096, and then in increments of 4096 to a maximum of 32K bytes.

number
is at least 3.

The size of the buffers multiplied by the number of buffers (*size* x *number*) must be less than 16 megabytes.

FIX={**BFR**|**IOB**|(**BFR**,**IOB**)}
specifies that I/O buffers (BFR), or I/O-related control blocks (IOB), or both, are to be fixed in real storage. With GSR, IOB includes channel programs. If the program that issues BLDVRP with FIX specified is not authorized to fix areas in real storage, FIX is ignored. A program is authorized if it is in supervisor state with protection key 0 to 7, or has been link-edited with authorization (the authorized program facility is described in *Supervisor Services and Macro Instructions*).

Note: If FIX is specified, DLVRP must be issued by the same task that issues BLDVRP.

KEYLEN=*length*
specifies the maximum key length of the data sets that are to share the resource pool. The default is 255. The keys whose lengths must be provided for are the prime key of each key-sequenced data set and the alternate key

of each alternate index that is used for processing or is being upgraded. If none of the data sets is keyed, specify 0.

STRNO=*number*

specifies the total number of placeholders required for all the data sets that are to share the resource pool. 1 is minimum; 255 is maximum.

The number should equal the potential number of requests that may be issued concurrently for all the data sets that will share the resource pool. If a request fails because the number of placeholders is insufficient (logical return code 64 (X'40')), you may retry the request; it will be assigned a placeholder if one has been released.

TYPE={LSR|GSR}

specifies whether a local (LSR) or a global (GSR) resource pool is to be built. Only one BLDVRP TYPE=LSR may be issued for each partition or address space. Only one BLDVRP TYPE=GSR may be issued for the system for each of the protection keys 0 through 7. The program that issues BLDVRP TYPE=GSR must be in supervisor state with protection key 0 to 7.

CHECK Macro (Wait for Completion of Request)

The syntax of the CHECK macro is:

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

where:

label
is 1 to 8 characters that provide a symbolic address for the CHECK macro.

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

Example 1: Check Return Codes after an Asynchronous Request

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

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

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

Example 2: Check Return Codes after a Synchronous Request

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

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

Example 3: Overlap Processing

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

```
BLOCK    ACB
LIST     RPL    ACB=BLOCK,  Asynchronous access.
                AREA=WORK,
                AREALEN=50,
                OPTCD=ASY
         .
         .
LOOP     GET    RPL=LIST
         LTR    15,15
         BNZ    NOTACCEP
```

Do other processing.

```
         CHECK  RPL=LIST    Suspends your processing to
                            wait for completion of GET
                            if necessary and to cause VSAM
                            to indicate return codes.
         LTR    15,15
         BNZ    ERROR
```

Process the record.

```
         B      LOOP
NOTACCEP ...                Request was not accepted.
ERROR    ...                Request failed.
         .
         .
WORK     DS     CL50        Work area.
```

After issuing the request, make sure that VSAM accepted it before you go on to other processing. When you have done as much other processing as you can, issue the CHECK macro. VSAM will not give you back control now until the request is complete. If you don't want to issue CHECK until you know the request is complete, use the ECB parameter of the RPL macro or the IO=COMPLETE parameter of the TESTCB macro. After you issue the CHECK, VSAM immediately returns a code and takes an exit, if necessary. See "RPL Macro (Generate a Request Parameter List)" on page 93 and "GENCB Macro (Generate a Request Parameter List)" on page 57 for information on the ECB parameter.

Example 4: Suspend a Request for Many Records

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

```
FIRST    RPL    ACB=BLOCK,
                AREA=AREA1,
                AREALEN=50,
                NXTRPL=SECOND,
                OPTCD=ASY
SECOND   RPL    ACB=BLOCK,
                AREA=AREA2,
                AREALEN=50,
                NXTRPL=THIRD,
                OPTCD=ASY
THIRD    RPL    ACB=BLOCK,     Last list does not indicate a
                AREA=AREA3,    next list.
                AREALEN=50,
                OPTCD=ASY
         .
         .
LOOP     GET    RPL=FIRST      Request gives the address of the
                               first request parameter list.
         LTR    15,15
         BNZ    NOTACCEP
```

Do other processing.

```
         CHECK  RPL=FIRST
         LTR    15,15
         BNZ    ERROR
```

Process the three records retrieved by the GET.

```
         B      LOOP
NOTACCEP ...                   Request wasn't accepted.
ERROR    ...                   Display the feedback field
                               (FIELDS=FDBK) of each request
                               parameter list to find
                               out which one had an error.
AREA1    DS     CL50           A single GET request causes VSAM
                               to put a record in each of AREA1,
                               AREA1, and AREA3.
AREA2    DS     CL50
AREA3    DS     CL50
```

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

CLOSE Macro (Disconnect Program and Data)

The syntax of the CLOSE macro is:

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

where:

label

is 1 to 8 characters that provide a symbolic address for the CLOSE macro.

address

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

options

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

Note: Because the CLOSE parameters are positional, include a comma for options (even if you don't specify options) before a subsequent parameter.

TYPE=T

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

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

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

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

CNVTAD Macro (Convert Address)

The syntax of the CNVTAD macro is:

[*label*]	**CNVTAD**	**RPL**=*address*

RPL=*address*
specifies the address of the request parameter list (RPL). The RPL identifies your opened VSAM data set and your arguments. The following RPL parameters and subparameters have meaning for the CNVTAD macro:

- **ACB**=*address*
  identifies your VSAM data set.

- **ARG**=*address*
  identifies your arguments. The address points to a parameter list, aligned on a fullword boundary as follows:

  Key-sequenced data set:

Offset	Length	Contents
0	3	Reserved; unused
3	1	Number of arguments (N) (N = 1 to 255)
4+(N−1)(10+K)	4	Feedback RBA (K = key length)
8+(N−1)(10+K)	4	Feedback volume serial number (K = key length)
14+(N−1)(10+K)	K	Full key argument (K = key length)

  Entry-sequenced data set or relative record data set:

Offset	Length	Contents
0	3	Reserved; unused
3	1	Number of arguments (N)
4+(N−1)(14)	4	Feedback RBA
8+(N−1)(14)	6	Feedback volume serial number
18+(N−1)(14)	6	RBA/RRN argument

  The value for K is always 4 in an entry-sequenced or relative record data set. Therefore, 10+K is always 14 for these two types of data sets. The maximum number of arguments allowed is 255.

- **ECB**=*address*
  specifies the address of an event control block (ECB) which you may specify. VSAM indicates in the ECB whether or not a request is complete. This parameter is optional.

- **OPTCD=({ADR|KEY}**
  ,{ASY|SYN}
  ,{KEQ|KGE}
  ,FKS)

  ADR is only valid for entry-sequenced data sets.

  KEY is only valid for key-sequenced data sets and relative record dat a sets.

  If ASY is specified, you cannot WAIT on the RPLECB field for MNTACQ or ACQRANGE. You use the address placed in the parameter list feedback area. This address points to a list of ECBs (in standard WAIT list format) which you may use in place of the RPLECB field.

  GEN is not supported; if specified, it will give an error indication.

  All other OPTCD subparameters are not applicable, and, if specified, are ignored with no error indications.

For a given list of discrete arguments, CNVTAD returns the volume serial number (volser) and the RBA corresponding to each argument in the parameter list feedback area. The data portion of your VSAM data set is not referenced and need not be mounted even if the sequence set is embedded.

For an entry-sequenced data set, the volser is returned, and the same RBA specified in the argument field is also returned.

Note: The RBA returned by CNVTAD in the case of a key-sequenced data set is not the exact RBA of the record. It is, in fact, an approximate value. (For data sets with the IMBED option, it is the RBA of the beginning of the sequence set for the record's control area; for data sets with NOIMBED, it is the RBA of the record's control interval.) When passed to MNTACQ, these RBA values cause MNTACQ to stage the appropriate cylinders corresponding to the requested arguments originally passed to CNVTAD. You should therefore use caution if you are planning to use the RBAs obtained from CNVTAD for any purpose other than as input to MNTACQ.

At the conclusion of this macro, the RPL is disconnected. Any positioning in effect prior to execution of this macro will be lost. You may have to reposition. Chained RPLs are not supported by CNVTAD.

DLVRP Macro (Delete VSAM Resource Pool)

The DLVRP macro has a standard form and an execute form. The standard form builds a parameter list and passes control to VSAM to delete the resource pool. The execute form is described in Appendix A, "List, Execute, and Generate Forms of Macros" on page 121.

The syntax of the DLVRP macro is:

DLVRP	**TYPE={LSR|GSR}**

TYPE={LSR|GSR}
specifies the type of resource pool to be deleted: local (LSR) or global (GSR). The local resource pool is the one in the partition or address space in which DLVRP is issued. The program that issues DLVRP TYPE=GSR must be in supervisor state with protection key 0 to 7.

ENDREQ Macro (Terminate a Request)

The syntax of the ENDREQ macro is:

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

where:

label
is 1 to 8 characters that provide a symbolic address for the ENDREQ macro.

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

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

Example: Release Positioning for Another Request

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

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

Decide whether to update the record.

```
          B      FORGET           No; do not update the record  Yes; update
          PUT    RPL=DIRUPD       the record, causing VSAM to forget its
                                  position for DIRUP.
          LTR    15,15
          BNZ    ERROR
          B      LOOP
FORGET    ENDREQ RPL=DIRUPD       Cause VSAM to forget its position for
                                  DIRUPD.  Release exclusive control.
          LTR    15,15
          BNZ    ERROR
          B      LOOP
ERROR     xxx                     Request wasn't accepted or failed.
```

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

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

ENDREQ can also be used to cancel an asynchronous request, rather than suspending processing with CHECK.

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

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

ERASE Macro (Delete a Record)

The syntax of the ERASE macro is:

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

where:

label
is 1 to 8 characters that provide a symbolic address for the ERASE macro.

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

With ERASE processing of key-sequenced data sets, VSAM attempts to make the control interval available to the control area when the last record in the control interval is erased. Thus, key-sequenced data set control intervals can be reused for new records whose keys fall anywhere within the control area's range of keys. You may suppress the process of reclaiming the control interval by setting the RPLNOCIR bit directly in the RPL used for ERASE. The format of an RPL is discussed in *VSAM Logic*. The high key control interval of a control area is never reclaimed.

Example 1: Keyed-Direct Deletion

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

```
DELETE   ACB    MACRF=(KEY,DIR,
                OUT)
LIST     RPL    ACB=DELETE,
                AREA=WORK,
                AREALEN=50,
                ARG=KEYFIELD,
                OPTCD=(KEY,DIR,
                SYN,UPD,          UPD indicates deletion.
                MVE,FKS,
                KEQ)
          .
          .
          .
LOOP     MVC   KEYFIELD,source    Search argument for retrieval, from a
                                  table or transaction record.
         GET    RPL=LIST
         LTR    15,15
         BNZ    ERROR
```

Decide whether to delete the record.

```
         BE     LOOP            No; retrieve the next record.
         ERASE  RPL=LIST        Yes; delete the record.
         LTR    15,15
         BNZ    ERROR
         B      LOOP
ERROR    ...                    Request was not accepted, or failed
WORK     DS     CL50            Examine the data record here.
KEYFIELD DS     CL5             Search argument.
```

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

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

Example 2: Addressed-Sequential Deletion

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

```
DELETE   ACB     MACRF=(ADR,SEQ,OUT)
REQUEST  RPL     ACB=DELETE,
                 AREA=WORK,
                 AREALEN=100,
                 ARG=ADDR,
                 OPTCD=(ADR,SEQ,ASY,
                 UPD,MVE)         UPD indicates deletion.

   ⋮
LOOP             ...              Decide whether you need to skip to another
                                  position (forward or backward)
         B       RETRIEVE         No; bypass the POINT.
         MVC     ADDR,source      Yes; move search argument for
                                  POINT into search-argument field.
         POINT   RPL=REQUEST      Position VSAM to the record to
                                  be retrieved next.
         LTR     15,15
         BNZ     ERROR
         CHECK   RPL=REQUEST
         LTR     15,15
         BNZ     ERROR
RETRIEVE GET     RPL=REQUEST
         LTR     15,15
         BNZ     ERROR
         CHECK   RPL=REQUEST
         LTR     15,15
         BNZ     ERROR
```

Decide whether to delete the record.

```
         BE      LOOP            No; skip ERASE and CHECK.
         ERASE   RPL=REQUEST     Yes; delete the record.
         LTR     15,15
         BNZ     ERROR
         CHECK   RPL=REQUEST
         LTR     15,15
         BNZ     ERROR
         B       LOOP
ERROR    ...                      Request was not accepted, or
         .                        failed.
         .
         .
ADDR     DS      F                RBA search argument for POINT.
WORK     DS      CL100            Work area.
```

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

EXLST Macro (Generate an Exit List)

The syntax of the EXLST macro is:

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

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

label
: is 1 to 8 characters that provide a symbolic address for the exit list that is established.

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

EODAD=(*address*[**,A|N**][**,L**])
JRNAD=(*address*[**,A|N**][**,L**])
LERAD=(*address*[**,A|N**][**,L**])
SYNAD=(*address*[**,A|N**][**,L**])
UPAD=(*address*[**,A|N**][**,A**])
: specify that you are supplying a routine for the exit specified. The exits and values that can be specified for them are:

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

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

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

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

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

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

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

L specifies that the address is that of an 8-byte field that contains the name of an exit routine in a partitioned data set that is identified by a JOBLIB or STEPLIB DD statement or in SYS1.LINKLIB. VSAM is to load the exit routine for exit processing. If **L** is omitted, the address gives the entry point of the exit routine in virtual storage.

Example: EXLST Macro

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

```
EXITS    EXLST  EODAD=(ENDUP,N),   EXITS gives symbolic
                LERAD=LOGICAL,     address of the exit
                SYNAD=(ROUTNAME,L) list.
ENDUP                              EODAD routine.
LOGICAL                            LERAD routine.
ROUTNAME DC     C'PHYSICAL'        Pad shorter names with blanks:
                                    C'SYN' or CL8'SYN'.
```

The EXLST macro's parameters are:

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

GENCB Macro (Generate an Access Method Control Block)

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

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

The subparameters of the GENCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. Appendix B, "Operand Notation" on page 129, gives all the ways of coding each subparameter for the macros that work at execution.

label
is 1 to 8 characters that provide a symbolic address for the GENCB macro.

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

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

BSTRNO=*number*
specifies the number of strings initially allocated for access to the base cluster of a path. The default is STRNO. BSTRNO is ignored if the object being opened is not a path. If the number specified for BSTRNO is insuffi-

cient, VSAM will dynamically extend the number of strings as needed for the access to the base cluster. BSTRNO can also influence performance. The VSAM control blocks for the set of strings specified by BSTRNO are allocated on contiguous virtual storage, whereas this is not guaranteed for the strings allocated by dynamic extension.

BUFND=*number*

specifies the number of I/O buffers VSAM is to use for transmitting data between virtual and auxiliary storage. A buffer is the size of a control interval in the data component. The minimum number you may specify is 1 plus the number specified for STRNO (if you omit STRNO, BUFND must be at least 2, because the default for STRNO is 1). The number can be supplied by way of the JCL DD AMP parameter as well as by way of the macro. The default is the minimum number required. A larger number for BUFND can improve the performance of sequential access.

BUFNI=*number*

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

BUFSP=*number*

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

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

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

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

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

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

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

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

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

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

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

DDNAME=*ddname*
is 1 to 8 characters that identify the data set that you want to process by specifying the JCL DD statement for the data set. You may omit DDNAME and provide it by way of the MODCB macro before opening the data set. MODCB is described later in this chapter.

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

LENGTH=*number*
specifies the length, in bytes, of the area, if any, that you are supplying for VSAM to generate the access method control block(s). (See the WAREA parameter.) The LENGTH value cannot exceed 65535 (X'FFFF').

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

specifies the kind(s) of processing you will do with the data set. The subparameters must be meaningful for the data set. For example, if you specify keyed access for an entry-sequenced data set, you cannot open the data set. You must specify all the types of access you're going to use, whether you use them concurrently or by switching from one to the other. The subparameters are shown in Figure 11 on page 27. They are arranged in groups, and each group has a default value (indicated by underlining). You may specify subparameters in any order. You may specify both ADR and KEY to process a key-sequenced data set. You may specify both DIR and SEQ; with keyed access, you may specify SKP as well. If you specify OUT and want merely to retrieve some records as well as update, delete, or insert others, you need not also specify IN.

MAREA=*address*
specifies the address of an optional OPEN/CLOSE or TYPE=T option (CLOSE macro) message area.

MLEN=*number*
specifies the length of an optional OPEN/CLOSE or TYPE=T option (CLOSE macro) message area.

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

STRNO=*number*
specifies the number of requests requiring concurrent data set positioning VSAM is to be prepared to handle. A request is defined by a given request parameter list or chain of request parameter lists. See "RPL Macro (Generate a Request Parameter List)" on page 93 and "GENCB Macro (Generate a Request Parameter List)" on page 57 for information on request parameter lists.

WAREA=*address*

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

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

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

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

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

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

The GENCB macro's parameters are:

- BUFND specifies four I/O buffers for data; BUFNI specifies three I/O buffers for index entries; and BUFSP specifies 19456 bytes of buffer space, enough space to accommodate control intervals of data that are 4096 bytes and of index entries that are 1024 bytes.
- DDNAME specifies that this access method control block is associated with a DD statement named DATASETS.
- EXLST specifies that the exit list associated with this access method control block is named EXITS.
- MACRF specifies keyed direct and keyed sequential processing for both insertion and update.
- PASSWD specifies the location, FIELD, of the password provided.
- STRNO specifies that two requests will require concurrent positioning.

GENCB Macro (Generate an Exit List)

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

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

The parameters of the GENCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. Appendix B, "Operand Notation" on page 129, gives all the ways of coding each subparameter for the macros that work at execution.

label
is 1 to 8 characters that provide a symbolic address for the GENCB macro.

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

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

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

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

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

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

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

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

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

L specifies that the address is that of an 8-byte field that contains the name of an exit routine in a partitioned data set that is identified by a JOBLIB or STEPLIB DD statement or in SYS1.LINKLIB. VSAM is to load the exit routine for exit processing. If **L** is omitted, the address gives the entry point of the exit routine in virtual storage. **L** may precede or follow the **A** or **N** specification.

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

LENGTH=*number*
specifies the length, in bytes, of the area, if any, that you are supplying for VSAM to generate the exit list(s). (See the WAREA parameter.) The LENGTH value cannot exceed 65535 (X'FFFF').

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

If you do not specify an area in which the exit list is to be generated, VSAM returns to your program the address of the area in which the exit list(s) is generated in register 1, and the length of the area in register 0. You can find the length of each exit list by dividing the length of the area by the number of copies. The address of each exit list can then be calculated by this offset from the address in register 1. You can find the length of an exit list with the SHOWCB macro, described under "SHOWCB Macro (Display Fields of an Exit List)" on page 104.

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

Example: GENCB Macro (Generate an Exit List)

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

```
EXITS    GENCB BLK=EXLST,
               EODAD=(EOD,N),
               LERAD=LOGICAL
               SYNAD=(ERROR,
               A,L)
         LTR   15,15
         BNZ   ERROR1          If error, go to the SYNAD routine.
         ST    1,EXLSTADR      Address of the exit list is
                               saved.
EOD      EQU   *               EODAD routine.
LOGICAL  EQU   *               LERAD routine.
ERROR    DC    C'PHYSICAL'     Name of the SYNAD module.
EXLSTADR DS    F               Save area for exit-list
                               address.
```

The GENCB macro's parameters are:

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

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

GENCB Macro (Generate a Request Parameter List)

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

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

The parameters of the GENCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. Appendix B, "Operand Notation" on page 129 gives all the ways of coding each subparameter for the macros that work at execution.

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

label
: is 1 to 8 characters that provide a symbolic address for the GENCB macro. For addressing lists generated by GENCB, see the discussion of the COPIES parameter.

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

ACB=*address*
: specifies the address of the access method control block that identifies the data set to which access will be requested. If you omit this parameter, you

must issue MODCB to specify the address of the access method control block before you issue a request. (MODCB is described later in this chapter.)

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

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

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

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

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

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

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

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

ECB=*address*
specifies the address of an event control block (ECB) that you may supply. VSAM indicates in the ECB whether a request is complete or not (using standard completion codes, which are described in ***Data Areas***). You can use the ECB to determine that an asynchronous request is complete before issuing a CHECK macro. This parameter is always optional.

KEYLEN=*number*
specifies the length, in bytes, of the generic key (OPTCD=GEN) you are using for a search argument (given in the field addressed by the ARG parameter). This parameter is required with a search argument that is a

generic key. The number can be 1 through 255. For full-key searches, VSAM knows the key length, which is taken from the catalog definition of the data set when you open the data set.

LENGTH=*number*

specifies the length, in bytes, of the area, if any, that you are supplying for VSAM to generate the request parameter list(s). (See the WAREA parameter.) The LENGTH value cannot exceed 65535 (X'FFFF'). You can find out how long a request parameter list is with the SHOWCB macro, described later in this chapter.

MSGAREA=*address*

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

MSGLEN=*number*

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

NXTRPL=*address*

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

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

specifies the subparameters that govern the request defined by the request parameter list. Each group of subparameters has a default; subparameters are shown in Figure 12 on page 95 with defaults underlined. Only one subparameter from each group is effective for a request. Some requests do not require an subparameter from all of the groups to be specified. The groups that are not required are ignored; thus, you can use the same request parameter list for a combination of requests (GET, PUT, POINT, for example) without zeroing out the inapplicable subparameters each time you go from one request to another.

RECLEN=*number*

specifies the length, in bytes, of a data record being stored. If the records you are storing are all the same length, you will not need to change RECLEN after you set it. This parameter is required for PUT requests. For GET requests, VSAM puts the length of the record retrieved in this field in the request parameter list. It will be there if you update and store the record.

TRANSID=*number*

specifies a number that relates modified buffers in a buffer pool. Use in shared resource applications and a description are in "Sharing Resources" in *VSAM Administration Guide*.

WAREA=*address*

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

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

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

Building a Chain of Request Parameter Lists

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

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

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

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

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

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

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

Example: GENCB Macro (Generate a Request Parameter List)

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

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

The GENCB macro's parameters are:

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

GET Macro (Retrieve a Record)

The syntax of the GET macro is:

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

where:

label
is 1 to 8 characters that provide a symbolic address for the GET macro.

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

Example 1: Keyed-Sequential Retrieval (Forward)

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

```
INPUT   ACB  MACRF=(KEY,       All MACRF and OPTCD subparameters specified
             SEQ,IN)           are defaults and could have been omitted.
RETRVE  RPL  ACB=INPUT,
             AREA=IN,
             AREALEN=100,
             OPTCD=(KEY,SEQ,
             SYN,NUP,MVE)

  :
LOOP    GET  RPL=RETRVE        This GET or identical GETs can be issued,
                               with no change in the request parameter
                               list, to retrieve subsequent records in
                               key sequence.
        LTR  15,15
        BNZ  ERROR

  :
        B    LOOP
ERROR   ...                    Request was not accepted, or failed.

  :
IN      DS   CL100             IN contains a data record after
                               GET is completed.
```

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

Example 2: Keyed-Sequential Retrieval (Backward)

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

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

  ⋮
        B      LOOP
EOD     EQU    *                 Come here for end of data.
ERROR   ...                      Request failed.

  ⋮
IN      DS     CL100             Area for retrieved record.
```

Example 3: Skip-Sequential Retrieval

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

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

```
        GENCB  BLK=ACB,         VSAM gets an area in virtual
               DDNAME=INPUT,    storage to generate the access
               MACRF=(KEY,      method control block and
               SKP,IN)          returns the address in register 1.
        LTR    15,15
        BNZ    CHECK0
        LR     2,1
        GENCB  BLK=RPL,
               ACB=(2),
               AREA=RCDADDR,
               AREALEN=4,
               ARG=SRCHKEY,
               OPTCD=(KEY,SKP,
               SYN,NUP,KGE,
               FKS,LOC)
        LTR    15,15
        BNZ    CHECK0
        LR     3,1               Address of the request parameter
                                 list.

  ⋮
LOOP    MVC    SRCHKEY,source    Search argument for retrieval, moved in
                                 in from a table or a transaction record.
        GET    RPL=(3)
        LTR    15,15
        BNZ    ERROR
        SHOWCB AREA=RCDLEN,     Display the length of the record.
               FIELDS=RECLEN,
               LENGTH=4,
               RPL=(3)
        LTR    15,15
        BNZ    CHECK0

  ⋮
        B       LOOP
ERROR   ...                     Request was not accepted, or failed.
CHECK0  ...                     Generation or display failed.

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

The macros and instructions are as follows:

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

Example 4: Addressed-Sequential Retrieval

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

```
BLOCK  ACB    DDNAME=INPUT,
              MACRF=(ADR,SEQ,IN)

  ⋮
       GENCB  BLK=RPL,
              COPIES=10,
              ACB=BLOCK,
              OPTCD=(ADR,SEQ,
              SYN,NUP,MVE)
       LTR    15,15
       BNZ    CHECK0
       LA     3,10            Number of lists(10).
       LR     2,1             Address of the first list.
       LR     1,0             Length of all the lists.
                              Registers 0 and 1 contain length and
                              address of the generated control blocks
                              when VSAM returns control after GENCB.
       SR     0,0             Prepare for following division.
       DR     0,3             Divide number of lists into length
                              of all the lists.
       LR     3,1             Save the resulting length of a
                              single list for an offset.
       LR     4,2             Save address of the first list.
       LA     5,RECAREA       Address of the first work area.
       .                      Do the following 6 instructions 10 times
       .                      to set up all the request parameter lists.
       .                      The 10th time, register 4 must be set
                              to 0 to indicate the last request
                              parameter list in the chain.
       AR      4,3            Address the next list.
       MODCB   RPL=(2),       In each request parameter list,
               NXTRPL=(4),    indicate the address of the next
               AREA=(5),      list and the address and length
               AREALEN=20     of the work area.
        LTR    15,15
        BNZ    CHECK0
        AR     2,3            Address the next list.
        LA     5,20(5)        Address the next work area.
        .                     Restore register 2 to address the
        .                     first list before continuing to
        .                     process.
LOOP    GET    RPL=(2)
        LTR    15,15
        BNZ    ERROR
        .                     Process the 10 records that have
        .                     been retrieved by the GET.
        .
        B      LOOP

CHECK0  ...
ERROR   ...                   Display the feedback field (FIELDS=FDBK)
                              of each request parameter list to find out
                              which one had an error.
RECAREA  DS    CL200          Space for a work area for each of
                              the 10 request parameter lists.
```

The GENCB macro generates 10 request parameter lists; the lists are subsequently chained together by using the MODCB macro to modify the NXTRPL parameter in each copy. Because SEQ is specified in each request parameter list and no previous request has been issued against the access method control block since it was opened, retrieval begins at the beginning of the data set.

Each time the GET macro is executed, VSAM is positioned at the next record in RBA sequence. VSAM moves each record into the work area provided for the request parameter list that identifies the record.

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

Example 5: Sequential Retrieval for a Relative Record Data Set

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

```
INPUT    ACB  MACRF=(KEY,SEQ,  All MACRF and OPTCD subparameters specified
              IN)              are defaults and could have been omitted.
RETRVE   RPL  ACB=INPUT,
              AREA=IN,
              AREALEN=100,
              ARG=RCDNO,
              OPTCD=(KEY,SEQ,
              SNY,NUP,MVE)

  ⋮
LOOP     GET  RPL=RETRVE       This GET or identical GETs can be issued, with
                               no change in the RPL, to retrieve subsequent
                               records in relative record number sequence.
         LTR  15,15
         BNZ  ERROR

  ⋮
         B    LOOP
ERROR    ...                   Request was not accepted or it failed.

  ⋮
IN       DS   CL100            IN contains a data record after GET
                               is completed.
RCDNO    DS   CL4              VSAM returns relative record number
                               of retrieved record in this field.
```

The records are retrieved in relative record number sequence. Empty records are bypassed for sequential retrieval. A 4-byte search argument must be specified. The relative record number of each record retrieved is stored in the search argument. VSAM is positioned at the first relative record when the data set is opened, and the next nonempty record is retrieved automatically as each GET is issued. The branch to ERROR would also be taken if the end of the data set is reached.

Example 6: Keyed-Direct Retrieval

In this example, a GET macro is used to retrieve fixed-length, 100-byte records directly by key. The key length is 15 bytes; the search argument is a 5-byte generic key, compared equal. The control blocks are generated at assembly.

```
INPUT    ACB  MACRF=(KEY,
              DIR,IN)
RETRVE   RPL  ACB=INPUT,         You specify all parameters for the request
              AREA=IN,           in the RPL macro.
              AREALEN=4,
              OPTCD=(KEY,
              DIR,SYN,NUP,
              KEQ,GEN,LOC),
              ARG=KEYAREA,
              KEYLEN=5

  ⋮
LOOP     MVC  KEYAREA,SOURCE     Search argument for retrieval, moved in
                                 from a table or a transaction record.
         GET  RPL=RETRVE         This GET or identical GETs can be issued
                                 with no change in the RPL:  Specify each
                                 new search argument in the field KEYAREA.
         LTR  15,15
         BNZ  ERROR
         .                       Process the record.

  ⋮
         B    LOOP
ERROR    ...                     Request was not accepted, or failed.

  ⋮
IN       DS   CL4                VSAM puts here the address of the record
                                 within the I/O buffer.
KEYAREA  DS   CL5                You specify the search argument here.
```

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

Example 7: Addressed-Direct Retrieval

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

```
BLOCK     ACB     DDNAME=INPUT,      Access method control block
                  MACRF=(ADR, DIR,   generated at assembly.
                  IN)

  ⋮
          GENCB   BLK=RPL,           ARG=SRCHADR, AREA=IN, AREALEN=20
                  COPIES=1,          Request parameter list generated
                  ACB=BLOCK,         at execution.
                  OPTCD=(ADR, DIR,
                  SYN, NUP, MVE)
          LTR     15,15
          BNZ     CHECK0
          LR      2, 1               Address of the list.

  ⋮
LOOP      MVC     SRCHADR,           Search argument for retrieval;
                                     calculated or moved in from a table
                                     or a transaction record.
          GET     RPL=(2)
          LTR     15, 15
          BNZ     ERROR
                                     Process the record.

  ⋮
          B      LOOP
CHECK0    ...                        Generation failed.
ERROR     ...                        Request was not accepted, or failed.

  ⋮
IN        DS     CL20                VSAM puts a record here for each GET
                                     request.
SRCHADR   DS     CL4                 You specify the RBA search argument
                                     here for each request.
```

The RBA provided for a search argument must match the RBA of a record. Keyed insertion and deletion of records in a key-sequenced data set will probably cause the RBAs of some records to change. Therefore, if you process a key-sequenced data set by addressed-direct access (or by addressed-sequential access using POINT), you need to keep track of changes. You can use the JRNAD exit for this purpose. See "EXLST Macro (Generate an Exit List)" on page 47.

Example 8: Switch from Direct to Sequential Retrieval

In this example, GET macros are used to retrieve fixed-length, 100-byte records. The retrieval is via an alternate index path defined with the nonunique key option. Every time a nonunique key is retrieved, the program switches to sequential processing to retrieve the other records with the same key. The control blocks were generated at assembly, but the MODCB macro is used to modify the request parameter list to permit switching from keyed-direct to keyed-sequential retrieval. For the direct request preceding sequential requests, the search argument is an 8-byte, generic key, compared equal. Positioning is requested for direct requests.

```
INPUT    ACB     MACRF=(KEY,DIR,    Both direct and sequential
                 SEQ,IN)            access specified.
RETRVE   RPL     ACB=INPUT,         NSP specifies that VSAM is
                 AREA=IN,           to remember its position.
                 AREALEN=100,
                 OPTCD=(KEY,DIR,
                 SYN,NSP,KEQ,
                 GEN,MVE),
                 ARG=KEYAREA,
                 KEYLEN=8

   ⋮
LOOP     MVC     KEYAREA,source     Search argument for direct retrieval;
                                    moved in from a table or a transaction
                                    record.
LOOP1    GET     RPL=RETRVE
         LTR     15,15
         BNZ     ERROR

   ⋮
         SHOWCB  RPL=RETRVE,        Extract feedback information.
                 AREA=FDBAREA,
                 FIELDS=FDBK
         LTR     15,15
         BNZ     ERROR
         CLI     ERRCD,8            Does a duplicate key follow?
         BE      SEQ                Yes; retrieve duplicates sequentially
         B       LOOP               No; retrieve next record in direct mode.
SEQ      MODCB   RPL=RETRVE,        Alter request parameter list for
                 OPTCD=SEQ          sequential access.
         LTR     15,15
         BNZ     CHECKO
SEQGET   GET     RPL=RETRVE         Do sequential retrieval.
         LTR     15,15              Test for error.
         BNZ     ERROR

   ⋮
         SHOWCB  RPL=RETRVE,        Extract feedback information.
                 AREA=FDBAREA,
                 FIELDS=FDBK
         LTR     15,15
         BNZ     ERROR
         CLI     ERRCD,8            Does a duplicate key follow?
         BE      SEQGET             Yes; retrieve sequentially.
DIR      MODCB   RPL=RETRVE,        Alter request parameter list
                 OPTCD=DIR          for direct access.
         LTR     15,15
         BNZ     CHECKO
         B       LOOP               Prepare new search argument.
ERROR    ...                        Request was not accepted, or
                                    failed.
CHECKO   ...                        Modification failed.

   ⋮
IN       DS      CL100              VSAM puts retrieved records
                                    here.
KEYAREA  DS      CL8                Specify the generic key for
                                    a direct request here.
FDBAREA  DS      OF                 Feedback area for SHOWCB.
         DS      1C                 Reserved.
TYPECD   DS      1C                 Error type code.
CMPCD    DS      1C                 Component code.
ERRCD    DS      1C                 Reason code.
```

MNTACQ Macro (Mount Acquire)

The syntax of the MNTACQ macro is:

[*label*]	**MNTACQ**	**RPL=***address*

RPL=*address*

specifies the address of the RPL that identifies your opened VSAM data set and your arguments. The following RPL parameters have meaning for MNTACQ:

- **ACB=***address*
  identifies your VSAM data set.

- **ARG=***address*
  identifies your arguments. address points to a parameter list, aligned on a fullword boundary as follows:

Offset	Length	Contents
0	4	Feedback area: address of an ECB WAIT list
4	6	VOLSER, target volume
10	1	Reserved
11	1	Argument entry count (N) (N = 1 to 255)
12	4N	Argument entries
12+4(N−1)	4	RBA for which an ACQUIRE is requested.

The maximum number of arguments is 255.

For the specified list, MNTACQ will acquire (stage) the data cylinders corresponding to each RBA for the one given volume. The volume will be mounted if necessary.

- **OPTCD=({ADR|KEY}**
  ,{ASY|SYN}
  ,{KEQ|KGE}
  ,FKS)

  ADR is valid for entry-sequenced data set, error for key-sequenced data set or relative record data set.

  KEY is valid for key-sequenced data set and relative record data set, error for entry-sequenced data set.

  If ASY is specified, you cannot WAIT on the RPLECB field for MNTACQ or ACQRANGE. You use the address placed in the parameter list feedback area. This address points to a list of ECBs (in standard WAIT list format) which you may use in place of the RPLECB field.

  GEN is not supported; if specified, it will give an error indication.

  All other OPTCD parameters are not applicable, and, if specified, are ignored with no error indication.

Because your request may result in the staging of numerous cylinders, a single ECB is not sufficient for an asynchronous MNTACQ request. The RPLECB field is inoperative for the MNTACQ interface. Upon return from an asynchronous MNTACQ, the feedback area of the MNTACQ parameter list will contain the address of a standard ECB WAIT list. You must then use this list in conjunction with the WAIT macro or you may use the list in conjunction with the EVENTS macro of MVS. An asynchronous request must conclude with either CHECK, ENDREQ, or CLOSE.

At the conclusion of this macro, the RPL is disconnected in a manner similar to that of a direct VSAM request. Any positioning in effect prior to execution of this macro will be lost. You may have to reposition. Chained RPLs are not supported by MNTACQ.

MODCB Macro (Modify an Access Method Control Block)

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

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

The parameters of the MODCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. Appendix B, "Operand Notation" on page 129, gives all the ways of coding each parameter for the macros that work at execution.

label

is 1 to 8 characters that provide a symbolic address for the MODCB macro.

ACB=*address*

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

Note: The remaining parameters represent parameters of the ACB macro that can be modified. The value specified replaces the value, if any, presently in the access method control block. *There are no defaults*. For an explanation of these parameters, see "ACB Macro (Generate an Access Method Control Block)" on page 24.

If MODCB is used to modify a MACRF subparameter, other subparameters are unaffected, except when they are mutually exclusive. For example, if you specify

MACRF=ADR in the MODCB and MACRF=KEY is already indicated in the control block, both ADR and KEY will now be indicated. But, if you specify MACRF=UBF in the MODCB and NUB is indicated, only UBF will now be indicated.

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

Note: If a user issues a MODCB for a non-VSAM and non-VTAM ACB, unpredictable results will occur.

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

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

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

MODCB Macro (Modify an Exit List)

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

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

The subparameters of the MODCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. Appendix B, "Operand Notation" on page 129, gives all the ways of coding each parameter for the macros that work at execution.

label
is 1 to 8 characters that provide a symbolic address for the MODCB macro.

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

The remaining parameters represent parameters of the EXLST macro that can be modified or added to an exit list. For an explanation of these parameters, see "EXLST Macro (Generate an Exit List)" on page 47.

Note: If the JRNAD exit is changed for an OPEN ACB, then the ACB must be closed and reopened in order to use the modified JRNAD exit.

Example: MODCB Macro (Modify an Exit List)

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

```
           MODCB  EXLST=(*,         Indirect notation is used to
                  EXLSTADR),        specify the address of the
                  EODAD=(EOD,L,A)   exit list, which was generated
                                    at execution.

   ⋮
EOD        DC     C'ENDUP'
EXLSTADR   DS     F                 When the exit list was generated,
                                    its address was saved here.
```

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

MODCB Macro (Modify a Request Parameter List)

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

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

The parameters of the MODCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. Appendix B, "Operand Notation" on page 129, gives all the ways of coding each parameter for the macros that work at execution.

label
: is 1 to 8 characters that provide a symbolic address for the MODCB macro.

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

Note: The remaining parameters represent parameters of the RPL macro that can be modified. The value specified replaces the value, if any, presently in the request parameter list. *There are no defaults.* For an explanation of these parameters, see "GENCB Macro (Generate a Request Parameter List)" on page 57.

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

Example: MODCB Macro (Modify a Request Parameter List)

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

Note: This example also shows the one exception to GENCB, MODCB, SHOWCB, and TESTCB building a parameter list and passing it to the control block manipulation module in register 1. In this example, the RPL address (in register 2) would be loaded into register 1 and the RECLEN value (in register 3) would be loaded into register 0. These registers would be passed to the control block manipulation macro. This will occur if the LIST, EXECUTE, or GENERATE form of the MODCB macro is not used and the only parameter specified, besides RPL, is RECLEN.

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

The MODCB macro's parameters are:

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

MRKBFR Macro (Mark Buffer)

The syntax of the MRKBFR macro is:

MRKBFR	**MARK={DINVALID|XINVALID|OUT|RLS}** **,RPL=***address*

MARK={DINVALID|XINVALID|OUT|RLS}
specifies whether to mark for output or to release from exclusive control or shared status the buffer identified in the RPL. To do both, issue MRKBFR twice, once with MARK=OUT, again with MARK=RLS.

DINVALID|XINVALID
specifies whether to mark the data component or index component buffers invalid. The buffers to be invalidated are identified as those which contain records, whose RBA values are within the RBA range pointed to by the RPL ARG address. DINVALID specifies that the data component buffers are to be marked invalid; XINVALID specifies that the index component buffers are to be marked invalid.

OUT
indicates that the buffer is to be marked for output. The buffer is kept under exclusive control or in shared status.

RLS
indicates that the buffer is to be released from exclusive control or shared status.

RPL=*address*
specifies the address of the request parameter list that defines the MRKBFR request. Use the RPL used by SCHBFR or GET to locate the buffer being marked or released. These RPL parameters have meaning for MRKBFR:

- **ACB=***address*
- **ARG=***address*
  The address of the 8-byte field that contains the beginning and ending RBAs of the range to be searched on.
- **ECB=***address*
- **TRANSID=***number*

All other RPL parameters are ignored. RPLs are assumed not to be chained. OPTCD=LOC is assumed.

If the ACB to which the RPL is related has MACRF=GSR, the program that issues MRKBFR must be in supervisor state with protection key 0 to 7.

OPEN Macro (Connect Program and Data)

The syntax of the OPEN macro is:

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

label

is 1 to 8 characters that provide a symbolic address for the OPEN macro.

address

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

options

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

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

Example: OPEN Macro

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

```
        GENCB  BLK=ACB,        An access method control block.
               DDNAME=DATA
        LTR    15,15
        BNZ    ERROR
        LR     2,1             Address of the control block.
        OPEN   (BLOCK,,(2))    A label is used for the access
                               method control block generated
                               by ACB; register notation is
                               used for the one generated by
                               GENCB.  The two commas indicate
                               the omission of options.
BLOCK   ACB                    Another access method control
                               block.
```

POINT Macro (Position for Access)

The syntax of the POINT macro is:

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

label
is 1 to 8 characters that provide a symbolic address for the POINT macro.

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

Example: Position with POINT

In this example, the POINT macro is used to position at a record identified by a full key (5-byte) search argument, compared equal.

```
BLOCK    ACB    DDNAME=IO        Default MACRF subparameters sufficient.
POSITION RPL    ACB=BLOCK,       ARG parameter and KEQ and FKS
                AREA=WORK,       OPTCD subparameters define the
                AREALEN=50,      POINT request.
                ARG=SRCHKEY,
                OPTCD=(KEY,SEQ,
         .      SYN,KEQ,FKS)
         .
LOOP     MVC    SRCHKEY,source   Search argument for positioning, moved
                                 in from a table or a transaction record.
         POINT  RPL=POSITION
         LTR    15,15
         BNZ    ERROR
LOOP1    GET    RPL=POSITION
         LTR    15,15
         BNZ    ERROR
```

Process the record. Decide whether to skip to another position (forward or backward).

```
         BE     LOOP       Yes; skip.
         B      LOOP1      No; continue in consecutive sequence.
ERROR   ...                Request was not accepted, or failed.
SRCHKEY DS     CL5         Search argument for positioning.
WORK    DS     CL50        VSAM puts a record here for each GET request.
```

PUT Macro (Store a Record)

The syntax of the PUT macro is:

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

label
is 1 to 8 characters that provide a symbolic address for the PUT macro.

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

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

Example 1: Keyed-Sequential Insertion

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

```
BLOCK    ACB     DDNAME=OUTPUT,
                 MACRF=(KEY,SEQ,OUT)
LIST     RPL     ACB=BLOCK,AREA=BUILDRCD,
                 AREALEN=250,OPTCD=(KEY,SEQ,
                 SYN,NUP,MVE)

   ⋮
LOOP     L       2,source            Put length of record to be
                                     inserted into register.
         MODCB   RPL=LIST,           Indicate record length in
                 RECLEN=(2)          request parameter list.
         LTR     15,15
         BNZ     CHECKO
         PUT     RPL=LIST
         LTR     15,15
         BNZ     ERROR
         B       LOOP
CHECKO   ...                         Modification failed.
ERROR    ...                         Request was not accepted, or failed.

   ⋮
BUILDRCD DS      CL250               Work area for building records.
```

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

Example 2: Recording RBAs When Loading

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

```
         LA     3,RBATABLE       Address of the beginning of the table.

   ⋮
LOOP     L      2,source         Put length of record to be
                                 inserted into register 2.
         MODCB  RPL=LIST,        Indicate record length in
                RECLEN=(2)       request parameter list.
         LTR    15,15
         BNZ    CHECKO
         PUT    RPL=LIST
         LTR    15,15
         BNZ    ERROR
         SHOWCB AREA=(3),        Each SHOWCB puts a record's
                FIELDS=RBA,      RBA into the table.
                LENGTH=4,
                RPL=LIST
         LTR    15,15
         BNZ    CHECKO
         MVC    4(15,3),         Put the record's key field in
                keyfield         the table.
         LA     3,20(3)          Point to the next entry.
         B      LOOP
ERROR    ...                     Request was not accepted, or failed.
CHECKO   ...                     Modification or display failed.

   ⋮
         DSECT                   Get enough virtual storage for
                                 as many table entries as there
                                 are records in the data set.
RBATABLE DS     OF
RBA      DS     CL4
KEY      DS     CL15
         DS     CL1              Padding to keep each RBA entry
                                 on a fullword boundary: SHOWCB's
                                 display area must be on a
                                 fullword boundary.
```

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

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

Example 3: Loading a Relative Record Data Set (Skip-Sequential and Direct Processing)

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

```
OUTPUT    ACB    MACRF=(KEY,SKP,
                 OUT)

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

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

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

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

```
   ⋮
LOOP      ...                      Restore address of first list
                                   in register 2.
                                   Build 5 records in WORK.
          PUT    RPL=(2)           Register 2 points to the first
                                   request parameter list in the chain.
                                   The five records in WORK are
                                   stored with this one PUT request.
          LTR    15,15
          BNZ    NOTACCEP

   ⋮
          CHECK  RPL=(2)
          LTR    15,15
          BNZ    ERRO
          B      LOOP
CHECKO    ...                      Generation or modification failed.
NOTACCEP  ...
ERROR     ...                      Display the feedback field in
                                   each request parameter list to
                                   find out which one had an error.
WORK      DS     CL500             Contains five 100-byte work areas.
```

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

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

Example 6: Keyed-Direct Insertion

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

```
OUTPUT   ACB    MACRF=(KEY,DIR,OUT)
DIRECT   RPL    ACB=OUTPUT,AREA=WORK,
                AREALEN=100,OPTCD=(KEY,
                DIR,ASY,NUP,MVE),
                RECLEN=100

   ⋮
LOOP     PUT    RPL=DIRECT
         LTR    15,15
         BNZ    NOTACCEP

   ⋮
         CHECK  RPL=DIRECT
         LTR    15,15
         BNZ    ERROR
         B      LOOP
NOTACCEP ...                      Request was not accepted.
ERROR    ...                      Request failed.

   ⋮
WORK     DS     CL100             Work area.
```

The macros are as follows:

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

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

Example 7: Addressed-Sequential Addition

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

```
BLOCK   ACB    MACRF=(ADR,SEQ,OUT)
LIST    RPL    ACB=BLOCK,AREA=NEWRCD,
               AREALEN=100,OPTCD=(ADR,
               SEQ,SYN,MVE)

   ⋮
LOOP    ...                       Build the record.
        L      3,source           Put the length of the record
                                  into register 3.
        MODCB  RPL=LIST,          Indicate length of new
               RECLEN=(3)         record.
        LTR    15,15
        BNZ    CHECKO
        PUT    RPL=LIST
        LTR    15,15
        BNZ    ERROR
        B      LOOP
CHECKO  ...                       Modification failed.
```

```
ERROR   ...                       Request was not accepted, or failed.
NEWRCD  DS     CL100              Build record in this work area.
```

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

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

Example 8: Keyed-Sequential Update

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

```
UPDATA  ACB    MACRF=(KEY,SEQ,OUT)
LIST    RPL    ACB=UPDATA,      UPD indicates the record may be
               AREA=WORK,       stored back (or deleted).
               AREALEN=50,
               OPTCD=(KEY,SEQ,
               SYN,UPD,MVE)

   :
LOOP    GET    RPL=LIST
        LTR    15,15
        BNZ    ERROR
```

Decide whether to update the record.

```
        BE     LOOP             Do not update it; retrieve another.
```

Do update the record.

```
        PUT    RPL=LIST         Store the record back.
        LTR    15,15
        BNZ    ERROR
        B      LOOP
ERROR   ...                     Request was not accepted, or failed.

   :
WORK    DS    CL50              VSAM puts the retrieved record here.
```

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

Example 9: Keyed-Direct Update

In this example, GET and PUT macros are used to retrieve and update records. The MODCB macro is used to modify record length (RECLEN) in the request parameter list when an update causes the record length to change. The maximum record length is 120 bytes. The search argument is a full key (5 bytes), compared equal.

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

  ⋮
```

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

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

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

```
         BE     STORE             No; length not changed.
         L      5,length          Yes; load new length into register 5.
         MODCB  RPL=UPDTE,        Modify length indication in
                RECLEN=(5)        the request parameter list.
         LTR    15,15
         BNZ    CHECKO
STORE    PUT    RPL=UPDTE
         LTR    15,15
         BNZ    ERROR
         B      LOOP
ERROR    ...                      Request was not accepted, or failed.
CHECKO   ...                      Display or modification failed.

  ⋮
IN       DS      CL120            Work area for retrieving, updating,
                                  and storing a record.
KEYAREA  DS      CL5              Search argument for
                                  retrieving a record.
RLNGTH   DS      F                Area for displaying the
                                  length of a retrieved record.
```

You cannot update records in the I/O buffer. A direct GET for update positions VSAM at the record retrieved, in anticipation of storing back (or deleting) the record. This positioning also allows you to switch to sequential access to retrieve the next record. When PUT is issued after a DIRUPD GET request, PUT causes VSAM to release exclusive control.

You do not have to store back a record that you retrieve for update, but, if you do not store it back before another retrieval, the current updates are lost.

Example 10: Addressed-Sequential Update

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

```
ENTRY   ACB    MACRF=(ADR,SEQ,OUT)
ADRUPD  RPL    ACB=ENTRY,         UPDTE indicates update (or deletion).
               AREA=WORK,
               AREALEN=200,
               OPTCD=(ADR,SEQ,
               SYN,UPD,MVE)

  ⋮

LOOP    GET    RPL=ADRUPD
        LTR    15,15
        BNZ    ERROR
        SHOWCB RPL=ADRUPD,        Find out how long the record is.
               AREA=RECLEN,
               FIELDS=RECLEN,
               LENGTH=4
        LTR    15,15
        BNZ    CHECKO

  ⋮
        PUT    RPL=ADRUPD
        LTR    15,15
        BNZ    ERROR
        B      LOOP
ERROR   ...                       Request was not accepted, or failed.
CHECKO  ...                       Display failed.
WORK    DS     CL200              Record-processing work area.
RLNGTH  DS     F                  Display area for length of records.
```

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

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

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

Example 11: Marking Records Inactive

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

```
ENTRYSEQ ACB    MACRF=(ADR,DIR,
                OUT)
LIST     RPL    ACB=ENTRYSEQ,      UPD indicates update;
                AREA=RECORD,       storing the record back
                AREALEN=100,       marked inactive.
                OPTCD=(ADR,DIR,
                SYN,UPD,MVE),
                ARG=RBAAREA

  ⋮
LOOP     GET    RPL=LIST
         LTR    15,15
         BNZ    ERROR
```

Decide whether you still want the data in the record.

```
         BE     LOOP               Yes; retrieve the next record.
         MVI    RECORD,X'FF'       No; flag the record inactive.
         PUT    RPL=LIST           Storing the record with an inactive
                                   indicator is equivalent to deletion
                                   for an entry-sequenced data set.
         LTR    15,15
         BNZ    ERROR
         B      LOOP
ERROR    ...                       Request was not accepted, or failed.
RECORD   DS     CL100              Work area for marking records.
RBAAREA  DS     F                  Search argument for retrieving
                                   the record.
```

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

PUTIX Macro (Store an Index Record)

The syntax of the PUTIX macro is:

[*label*]	**PUTIX**	**RPL=***address*

where:

label
is 1 to 8 characters that provide a symbolic address for the PUTIX macro.

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

The following RPL parameters and subparameters are required for PUTIX:

- **OPTCD=(CNV**
  ,DIR
  ,UPD
  ,MVE)
  The contents of a control interval must previously have been retrieved for update by way of GETIX.

  OPTCD=LOC is not allowed.

- **AREALEN**
  must be at least index control interval size.

To process the index of a key-sequenced data set with GETIX, you must open the cluster with:

- **ACB MACRF=(CNV,...)**

RPL Macro (Generate a Request Parameter List)

The syntax of the RPL macro is:

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

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

label
: is 1 to 8 characters that provide a symbolic address for the request parameter list that is generated. You can use it in the request macros to give the address of the list. You can use it in the NXTRPL parameter of the RPL macro, when you are chaining request parameter lists, to indicate the next list.

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

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

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

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

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

ECB=*address*
specifies the address of an event control block (ECB) that you may supply. VSAM indicates in the ECB whether a request is complete or not (using standard completion codes, which are described in *Data Areas*). You can use the ECB to determine that an asynchronous request is complete before issuing a CHECK macro. (If you issue a CHECK before a request is complete, you give up control and must wait for completion.) The ECB parameter is always optional.

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

MSGAREA=*address*
specifies the address of an area that you may, optionally, supply for VSAM to send you a message in case of a physical error. The format of a physical error message is given in "Reason Code (Physical Errors)" on page 19.

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

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

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

specifies the subparameters that govern the request defined by the request parameter list. Each group of subparameters has a default; subparameters are shown in Figure 12 with defaults underlined. Only one subparameter from each group can be specified. Some requests do not require an subparameter from all of the groups to be specified. The groups that aren't required are ignored; thus, you can use the same request parameter list for a combination of requests (GET, PUT, POINT, for example) without zeroing out the inapplicable subparameters each time you go from one request to another.

Option	Meaning
ADR	Addressed access to a key-sequenced or an entry-sequenced data set: RBAs are used as search arguments and sequential access is done by entry sequence.
CNV	Control interval access (this type of access is described in *VSAM Administration Guide*).
KEY	Keyed access to a key-sequenced or relative record data set: keys or relative record numbers are used as search arguments and sequential access is done by key or relative record number sequence.
DIR	Direct access to a key-sequenced, entry-sequenced, or relative record data set.
SEQ	Sequential access to a key-sequenced, entry-sequenced, or relative record data set.
SKP	Skip sequential access to a key-sequenced or a relative record data set: used with keyed access only.
ARD	User's argument determines the record to be located, retrieved, or stored.
LRD	Last record in the data set is to be located (POINT) or retrieved (GET direct); requires OPTCD=BWD.
FWD	Processing to proceed in a forward direction.
BWD	Processing to proceed in a backward direction; for keyed (KEY) or addressed (ADR) sequential (SEQ) or direct (DIR) requests; valid for POINT, GET, PUT, and ERASE operations; establish positioning by a POINT with OPTCD=BWD or by a GET direct with OPTCD=(NSP,BWD). When OPTCD=BWD is specified, subparameters KGE and GEN are ignored; subparameters KEQ and FKS are assumed.
ASY	Asynchronous access; VSAM returns to the processing program after scheduling a request so the program can do other processing while the request is being carried out.

Figure 12 (Part 1 of 2). OPTCD Options

Option	Meaning
SYN	Synchronous access; VSAM returns to the processing program after completing a request.
NSP	With OPTCD=DIR only, VSAM is to remember its position (for subsequent sequential access); that is, the position is not to be forgotten unless an ENDREQ macro is issued.
NUP	A data record that is being retrieved will not be updated or deleted; a record that is being stored is a new record; VSAM doesn't remember its position for direct requests into a work area.
UPD	A data record that is being retrieved may be updated or deleted; a record that is being stored or deleted was previously retrieved with OPTCD=UPD; VSAM remembers its position for sequential and direct GET requests. When PUT is issued after a DIRUPD GET request, PUT causes VSAM to release exclusive control.
KEQ	For GET with OPTCD=(KEY,DIR) or (KEY,SKP) and for POINT with OPTCD=KEY, the key (full or generic) that you provide for a search argument must equal the key or relative record number of a record. For a relative record data set, KEQ is assumed except for POINT.
KGE	For the same cases as KEQ, if the key (full or generic) that you provide for a search argument doesn't equal that of a record, the request applies to the record that has the next higher key. For a relative record data set and POINT, KGE positions to the specified relative record number whether the slot is empty or not. If the relative record number is greater than the highest existing record, EOD is returned. A subsequent PUT will insert the record at this position.
FKS	A full key is provided as a search argument.
GEN	A generic key is provided as a search argument; give the length in the KEYLEN parameter.
NWAITX	Never take the user's UPAD exit.
WAITX	If OPTCD=SYN and the ACB's MACRF=LSR GSR and UPAD exit routing is specified, VSAM takes the UPAD exit at points when VSAM would normally issue a WAIT.
LOC	For retrieval, VSAM leaves the data record in the I/O buffer for processing; not valid for PUT or ERASE; valid for GET with OPTCD=UPD. However, to update the record, you must build a new version of the record in a work area and modify the request parameter list OPTCD from LOC to MVE before issuing a PUT. For keyed-sequential retrieval, modifying key fields in the I/O buffer may cause incorrect results for subsequent GET requests until the I/O record is reread.
MVE	For retrieval, VSAM moves the data record to a work area for processing, and for storage, VSAM moves it from the work area to the I/O buffer.

Figure 12 (Part 2 of 2). OPTCD Options

RECLEN=*number*

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

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

TRANSID=*number*
specifies a number that relates modified buffers in a buffer pool. Used in shared resource applications and described in the chapter "Sharing Resources" in *VSAM Administration Guide*.

Example: RPL Macro

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

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

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

The RPL macro's parameters are:

- ACB associates the request parameter list with the access method control block generated by ACCESS.
- AREA and AREALEN specify a work area, WORK, that is 125 bytes long.
- ARG specifies that the search argument is defined at SEARCH. The search argument is 8 bytes long.
- MSGAREA and MSGLEN specify a message area, MESSAGE, that is 128 bytes long. The message area is provided for physical error messages.
- OPTCD specifies skip-sequential processing and specifies that a retrieved record may be updated or deleted.

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

OBJECT=DATA|INDEX

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

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

specifies the fields whose contents are to be displayed. Some of the fields can be displayed at any time; others only after a data set is opened. The ones that can be displayed only after a data set is opened can, in the case of a key-sequenced data set that has been opened for keyed access, pertain either to the data or to the index. See the OBJECT parameter. Figure 13 explains the keywords you can code in the FIELDS parameter for an access method control block.

Figure 13 (Page 1 of 3). FIELDS Operand Keywords for an Access Method Control Block

Keyword	Fullwords	Description of the Field
		The following fields can be displayed at any time:
ACBLEN	1	Length of an access method control block (displaying the length of an access method control block gives your program independence from changes in the length that may occur from release to release of VSAM)
BSTRNO	1	Number of strings initially allocated for access to the base cluster by a path
BUFND	1	Number of I/O buffers to be used for data, as specified in the ACB (or GENCB)
BUFNI	1	Number of I/O buffers to be used for index entries, as specified in the ACB (or GENCB)
BUFSP	1	Amount of space specified in the ACB (or GENCB) for I/O buffers
DDNAME	2	Name of the DD statement that identifies the data set
ERROR	1	The code returned by VSAM after the opening or closing of the data set (see "OPEN Macro (Connect Program and Data)" on page 80 and "CLOSE Macro (Disconnect Program and Data)" on page 38).
EXLST	1	Address of the exit list, if any; 0 if none
MAREA	1	Address of the message area, if any; 0 if none
MLEN	1	Length of the message area, if any; 0 if none
PASSWD	1	Address of the field containing the password; the first byte of the field contains the length of the password (in binary)
STRMAX	1	Maximum number of strings concurrently active

Figure 13 (Page 2 of 3). FIELDS Operand Keywords for an Access Method Control Block

Keyword	Fullwords	Description of the Field
STRNO	1	Number of requests for which VSAM is prepared to remember its position in the data set
		The following fields can be displayed only after the data set is opened:
AVSPAC	1	Amount of available space in the data component or index component, in bytes
BFRFND	1	Number of successful look-asides
BUFNO	1	Number of I/O buffers actually in use for the data component or index component
BUFRDS	1	Number of buffer reads
CINV	1	Control interval size for the data component or index component
ENDRBA	1	Ending RBA of the space used by the data component or index component; not the RBA of any record in the data set, but of the last used byte in the data set
FS	1	Number of free control intervals per control area in the data component (0 for OBJECT=INDEX)
HALCRBA	1	High-allocated RBA; the relative byte address of the end of the data component (OBJECT=DATA) or the index component (OBJECT=INDEX)
KEYLEN	1	Length of the key of reference of the key field of data records in the data component (whether OBJECT=DATA or INDEX)
LRECL	1	Length of data records in the data component (maximum length for variable-length data records) or of index records in the index component (control interval length minus 7)
NCIS	1	Number of control intervals that have been split in the data component (0 for OBJECT=INDEX)
NDELR	1	Number of records that have been deleted from the data component (0 for OBJECT=INDEX)
NEXCP	1	Number of EXCP macros that VSAM has issued for access to the data component or index component.
NEXT	1	Number of extents now allocated to the data component or index component (the maximum that can be allocated in 123)
NINSR	1	Number of records that have been inserted into (or added to) the data component (0 for OBJECT=INDEX)
NIXL	1	Number of levels in the index component (0 for OBJECT=DATA)
NLOGR	1	Number of records in the data component or index component
NRETR	1	Number of records that have ever been retrieved from the data component (0 for OBJECT=INDEX)
NSSS	1	Number of control areas that have been split in the data component (0 for OBJECT=INDEX)
NUIW	1	Number of writes not initiated by the user

Figure 13 (Page 3 of 3). FIELDS Operand Keywords for an Access Method Control Block

Keyword	Fullwords	Description of the Field
NUPDR	1	Number of records in the data component or index component that have ever been updated
RKP	1	Displacement of the key of reference of the key field from the beginning of a data record (whether OBJECT=DATA or INDEX)
STMST	2	System time stamp, which gives the time and day of the last time the data component or index component was closed, with bit 51 (counting from 0 at the left) equivalent to one microsecond and bits 52 through 63 unused
UIW	1	Number of user-initiated writes

Note: If a user issues a SHOWCB for a non-VSAM and non-VTAM ACB, unpredictable results will occur.

Example 1: SHOWCB Macro (Display an Access Method Control Block)

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

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

The SHOWCB macro's parameters are:

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

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

Example 2: SHOWCB Macro (Display an Exit List Address)

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

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

The SHOWCB macro's parameters are:

- ACB specifies the address of an access method control block from which the address of an exit list is to be displayed.
- AREA and LENGTH specify an area and length, 4 bytes, to be used to display the address of the exit list.
- FIELDS specifies that the EXLST field in an access method control block is to be displayed.

SHOWCB Macro (Display Fields of an Exit List)

The syntax of the SHOWCB macro used to display fields in an exit list is:

[*label*]	**SHOWCB**	**EXLST=***address* **,AREA=***address* **,LENGTH=***number* **,FIELDS=([EODAD] [,EXLLEN] [,JRNAD]** **[,LERAD][,SYNAD])**

The subparameters of the SHOWCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. Appendix B, "Operand Notation" on page 129, gives all the ways of coding each subparameter for the macros that work at execution.

label
is 1 to 8 characters that provide a symbolic address for the SHOWCB macro.

EXLST=*address*
specifies the address of the exit list whose fields are to be displayed. If you used the EXLST macro with a label, you can specify the label here. The EXLST parameter is optional only when you want to display the length that an exit list can have (see FIELDS=EXLLEN below). The SHOWCB macro does not support the UPAD user exit.

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

LENGTH=*number*
specifies the length, in bytes, of the work area that you are providing for VSAM to display the indicated fields in. Each exit-list field requires a fullword. If the area is not large enough for all the fields, VSAM doesn't display any of their contents and returns an error code (see "Control Block Manipulation Macro Return Codes and Reason Codes" on page 8).

FIELDS=([EODAD][,EXLLEN][,JRNAD]
[,LERAD][,SYNAD])
specifies the values to be displayed, as follows:

EODAD
specifies that the address of the end-of-data-set routine is to be displayed.

EXLLEN
specifies that the length of the exit list indicated in the EXLST parameter or if EXLST is omitted, the maximum length an exit length can have, is to be displayed.

JRNAD
specifies that the address of the journalizing routine is to be displayed.

LERAD
specifies that the address of the logical error analysis routine is to be displayed.

SYNAD
specifies that the address of the physical error analysis routine is to be displayed.

You can use SHOWCB to display the address of an exit routine only if the exit routine is indicated in the exit list. If it isn't, the SHOWCB request will fail. Use TESTCB to test whether an entry for a given exit type is present in the exit list and to find out whether the exit is active and whether the routine is to be loaded.

Example: SHOWCB Macro (Display the Length of an Exit List)

In this example, a SHOWCB macro is used to display the maximum length of an exit list. The maximum length of an exit list is subsequently used in a GENCB macro to get virtual storage for an exit list.

```
        SHOWCB  AREA=LENGTH,
                FIELDS=EXLLEN,
                LENGTH=4
        L       0,LENGTH          Amount of storage for GETMAIN.
        GETMAIN R,LV=(0)
        LR      2,1               Address of storage for GENCB.
        GENCB   BLK=EXLST,        Indirect notation for length
                LENGTH=(*,        of work area.
                LENGTH),
        .       WAREA=(2)
        .
        .
LENGTH  DS      F                 Contains the length of GENCB's
                                  work area.
```

The SHOWCB macro's parameters are:

- AREA and LENGTH specify the area, which begins on a fullword boundary, and its length, four bytes, that is to be used for the display.
- FIELDS specifies that the maximum length of an exit list is to be displayed. Because only EXLLEN is specified, the EXLST parameter is omitted.

The GENCB macro specifies a work area in which an exit list is to be generated. The length of the work area is located at LENGTH, where the maximum length of an exit list was put as a result of the SHOWCB macro.

SHOWCB Macro (Display Fields of a Request Parameter List)

The syntax of the SHOWCB macro used to display fields in a request parameter list is:

[*label*]	**SHOWCB**	**RPL**=*address* **,AREA**=*address* **,LENGTH**=*number* **,FIELDS=([ACB][,AIXPC][,AREA][,AREALEN]** **[,ARG][,ECB][,FDBK][,FTNCD]** **[,KEYLEN][,MSGAREA]** **[,MSGLEN]** **[,NXTRPL][,RBA]** **[,RECLEN]** **[,RPLLEN]** **[,TRANSID])**

The parameters of the SHOWCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. Appendix B, "Operand Notation" on page 129, gives all the ways of coding each subparameter for the macros that work at execution.

label
is 1 to 8 characters that provide a symbolic address for the SHOWCB macro.

RPL=*address*
specifies the address of the request parameter list whose fields are to be displayed. If you used the RPL macro with a label, you can specify the label here. The RPL parameter is optional when you want to display the length of a request parameter list (FIELDS=RPLLEN). (All VSAM request parameter lists have the same length, so you need not specify the address of a particular one.)

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

LENGTH=*number*
specifies the length, in bytes, of the work area that you are providing for VSAM to display the indicated fields in. Each request parameter list field requires a fullword. If the area is not large enough for all the fields, VSAM doesn't display any of their contents and returns an error code (see "Control Block Manipulation Macro Return Codes and Reason Codes" on page 8).

FIELDS=([ACB][,AIXPC][,AREA][,AREALEN][,ARG]
[,ECB][,FDBK][,FTNCD][,KEYLEN]
[,MSGAREA][,MSGLEN]
[,NXTRPL][,RBA][,RECLEN]
[,RPLLEN][,TRANSID])
specifies the fields whose contents are to be displayed. Figure 14 explains the keywords you can code in the FIELDS parameter for a request parameter

list. Some fields (each indicated by an asterisk (*) in Figure 14) are meaningful only if the requests have been completed; therefore, you must wait until the request has completed (for example, by issuing a CHECK if the request is asynchronous) before issuing SHOWCB.

Figure 14 (Page 1 of 2). FIELDS Operand Keywords for a Display Request Parameter List

Keyword	Fullwords	Description of the Field
ACB	1	Address of the access method control block that relates the request parameter list to the data
AIXPC*	1	Number of alternate index pointers
AREA	1	Address of the work area that the program uses to process a data record for the access as defined by the request parameter list
AREALEN	1	Length of the work area whose address is given in AREA
ARG	1	Address of the field containing a search argument, if search arguments are being used
ECB*	1	Address of an event control block, if any, in which VSAM indicates the completion of requests defined by the request parameter list
FDBK*	1	Reason code that VSAM puts into the feedback field to describe the error detected for the preceding request. (The meaning of this code depends on the contents of register 15, which indicates whether the request was successful or failed because of a logical or physical error. See "Record Management Return Codes and Reason Codes" on page 10)
FTNCD*	1	Code that describes the function in which a logical or physical error occurred; indicates whether the upgrade set may have been modified incorrectly by the preceding request (The meaning of this code depends on the contents of register 15, which indicates whether the request was successful or failed because of a logical or physical error. See "Record Management Return Codes and Reason Codes" on page 10)
KEYLEN	1	Length of the search argument, if a generic key is used for a search argument
MSGAREA*	1	Address of the area, if any, into which VSAM puts physical error messages
MSGLEN	1	Length of the message area, if any
NXTRPL	1	Address of the next request parameter list, if another one is chained to this one
RBA*	1	Relative byte address of the most recently processed record; you could use it to record the RBAs of records that you are retrieving or storing sequentially or by key
RECLEN*	1	Length of the data record, access to which is defined by the request parameter list

Figure 14 (Page 2 of 2). FIELDS Operand Keywords for a Display Request Parameter List

Keyword	Fullwords	Description of the Field
RPLLEN	1	Length of a request parameter list
TRANSID	1	Number that relates modified buffers in a buffer pool; described in *VSAM Administration Guide*

Example: SHOWCB Macro (Display a Physical Error Message)

In this example, a SHOWCB macro is used to display a physical error message. This example assumes that there is no SYNAD routine (or the SYNAD exit is inactive), in which case, VSAM returns control to your program following the last executable instruction if a physical error occurs. Register 15 indicates a physical error (12), and the feedback field in the request parameter list contains a code identifying the error; the message area contains more details about the error. Register 1 points to the request parameter list.

```
REQUEST  RPL    MSGAREA=
                MESSAGES,
                MSGLEN=128

  ⋮
         SHOWCB AREA=MSGADDR,
                FIELDS=MSGAREA,
                LENGTH=4,
                RPL=REQUEST
         LTR    15,15
         BNZ    CHECKO

  ⋮
CHECKO   ...                     Display failed.

  ⋮
MESSAGES DS    CL128             For VSAM to give you a detailed
                                 message about a physical error.
MSGADDR  DS    F                 For displaying the address of
                                 the message area with SHOWCB.
```

The RPL macro in this example provides for a message area, MESSAGES, of 128 bytes to be used for any physical error message.

The SHOWCB macro's parameters are:

- AREA and LENGTH specify a 4-byte area, MSGADDR, to be used for displaying the address of the message area for the associated request parameter list.
- FIELDS specifies that the address of the message area is to be displayed.
- RPL specifies the name, REQUEST, of the request parameter list for which the message area address is to be displayed.

TESTCB Macro (Test Fields of an Access Method Control Block)

Only one keyword can be specified each time you issue TESTCB. The syntax of the TESTCB macro used to test a field in an access method control block is:

[*label*]	TESTCB	ACB=*address*
		[,ERET=*address*]
		[,OBJECT=DATA|INDEX]
		,{ATRB=([ESDS][,KSDS][,REPL]
		[,RRDS][,SPAN][,SSWD][,WCK])|
		ATRB=UNQ|
		CATALOG={YES|NO}|
		CRA={SCRA|UCRA}|
		MACRF=([ADR][,AIX][,CFX][,CNV][,DDN]
		[,DFR][,DIR][,DSN][,GSR][,ICI][,IN]
		[,KEY][,LSR][,NCI][,NDF][,NFX][,NIS]
		[,NRM][,NRS][,NSR][,NUB][,OUT][,RST]
		[,SEQ][,SIS][,SKP][,UBF])|
		OFLAGS=OPEN|
		OPENOBJ={PATH|BASE|AIX}|
		ACBLEN=*number*|
		AVSPAC=*number*|
		BSTRNO=*number*|
		BUFND=*number*|
		BUFNI=*number*|
		BUFNO=*number*|
		BUFSP=*number*|
		CINV=*number*|
		DDNAME=*ddname*|
		ENDRBA=*number*|
		ERROR=*number*|
		EXLST=*address*|
		FS=*number*|
		KEYLEN=*number*|
		LRECL=*number*|
		MAREA=*address*|
		MLEN=*number*|
		NCIS=*number*|
		NDELR=*number*|
		NEXCP=*number*|
		NEXT=*number*|
		NINSR=*number*|
		NIXL=*number*|
		NLOGR=*number*|
		NRETR=*number*|
		NSSS=*number*|
		NUPDR=*number*|
		PASSWD=*address*|
		RKP=*number*|
		STMST=*address*|
		STRNO=*number*}

The subparameters of the TESTCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. Appendix B, "Operand Notation" on page 129, gives all the ways of coding each subparameter for the macros that work at execution.

ACB=*address*
specifies the address of the access method control block whose information you want to test. You may omit it only if you're testing the length of an access method control block (ACBLEN=number). (All VSAM access method control blocks have the same length.)

ERET=*address*
specifies the address of a routine to which VSAM is to give control if, because of an error, it is unable to test for the condition you specify. For example, testing AVSPAC in an access method control block for an unopened data set would fail. VSAM indicates in register 15 whether it could do the test and, if not, indicates in register 0 the reason it couldn't. (The reasons are discussed under "Control Block Manipulation Macro Return Codes and Reason Codes" on page 8.) A failure trying to execute TESTCB indicates a basic logical problem in the processing program, so the error routine would probably issue an ABEND. If it lets the program continue, it must branch to the continuation point itself, and not return to VSAM.

OBJECT=DATA|INDEX]
specifies whether you want to test a field for data or for index.

ATRB=([ESDS][,KSDS][,REPL]
[,RRDS][,SPAN][,SSWD][,WCK])
specifies, for an open data set, the attribute that is to be tested for, as follows:

ESDS
entry-sequenced data set

KSDS
key-sequenced data set

REPL
some portion of the index is replicated

RRDS
relative record data set

SPAN
data set contains spanned records

SSWD
sequence set is adjacent to the data

WCK
write operations for the data set are being verified

ATRB=UNQ
specifies, for an open alternate index or path, that the alternate index requires unique keys. The test for ATRB=UNQ must be made with a separate TESTCB macro. VSAM examines the path control blocks for the UNQ attribute; and also examines the base cluster's control blocks for the other attributes. If other attributes are tested for, VSAM examines the base clus-

ter's control blocks for all attributes: The test for ATRB=UNQ would give inaccurate results when applied to the base cluster's control blocks.

CATALOG=YES|NO
specifies that a test is to be made to determine, any time, whether or not the access method control block specifies a catalog data set.

CRA=SCRA|UCRA
specifies that a test is to be made to determine, any time, whether catalog recovery area control blocks are to be built in system storage or user storage.

MACRF=([ADR][,AIX][,CFX][,CNV][,DDN][,DFR][,DIR]
[,DSN][,GSR][,ICI][,IN][,KEY][,LSR]
[,NCI][,NDF][,NFX][,NIS][,NRM][,NRS][,NSR]
[,NUB][,OUT][,RST][,SEQ][,SIS][,SKP][,UBF])
specifies that a test is to be made to determine, at any time, what subparameter or combination of subparameters is being used for processing.

OFLAGS=OPEN
specifies that a test is to be made to determine, after open, whether the data set identified by the control block has been opened.

OPENOBJ=PATH|BASE|AIX
specifies that a test is to be made to determine, after open, whether an opened object is a path, a base cluster, or an alternate index.

The remaining parameters represent fields in an access method control block that can be compared with the value specified. These fields are the same as those that can be displayed by using the SHOWCB macro and are described in Figure 13 on page 100.

If you omit a routine to handle error conditions, you can examine register 15 following TESTCB by using a branch table, for example, but don't alter the PSW condition code that VSAM set to indicate the result of a test until you've had a chance to test it.

Note: If a user issues a TESTCB for a non-VSAM and non-VTAM ACB, unpredictable results will occur.

Example: TESTCB Macro (Test for Data Set Attributes)

In this example, a TESTCB macro is used to determine whether a data set is a key sequenced or an entry-sequenced data set.

```
LIST     RPL

  ⋮
         SHOWCB AREA=DATAFACT,
                FIELDS=ACB,
                LENGTH=4,
                RPL=LIST
         LTR    15,15
         BNZ    CHECKO
         TESTCB ACB=(*,            Is the data set key sequenced?
                DATAFACT),
                ATRB=KSDS,
                ERET=CHECKO
         BE     KEYSEQ             Yes.

  ⋮
KEYSEQ   ...                       Data set is key sequenced.

CHECKO   ...                       Display or test failed.

  ⋮
DATAFACT DS     F                  For displaying address of
                                   access method control block.
```

The SHOWCB macro's parameters are:

- AREA and LENGTH specify a 4-byte area, DATAFACT, aligned on a fullword boundary, to be used for the display.
- FIELDS and RPL specify that the address of the access method control block in the LIST request parameter list is to be displayed.

The TESTCB macro's parameters are:

- ACB specifies that a field in the access method control block, the address of which is located at DATAFACT, is to be tested. The SHOWCB macro put the address of the access method control block at DATAFACT.
- ATRB specifies that the access method control block is to be tested to determine whether it is a key-sequenced data set.
- ERET specifies that a routine named CHECKO is to be given control if an error occurs that makes it impossible to make the test.

There is no need to examine the feedback field in an EODAD routine, because it can be assumed to contain the end-of-data-set indication.

TESTCB Macro (Test Fields of an Exit List)

The syntax of the TESTCB macro used to test fields in an exit list is:

<table>
<tr><td>[<i>label</i>]</td><td><b>TESTCB</b></td><td><b>EXLST</b>=<i>address</i>
[,<b>ERET</b>=<i>address</i>]
,{<b>EODAD</b>={<b>0</b>|([<i>address</i>][,<b>A</b>|<b>N</b>][,<b>L</b>])}|
<b>JRNAD</b>={<b>0</b>|([<i>address</i>][,<b>A</b>|<b>N</b>][,<b>L</b>])}|
<b>LERAD</b>={<b>0</b>|([<i>address</i>][,<b>A</b>|<b>N</b>][,<b>L</b>])}|
<b>SYNAD</b>={<b>0</b>|([<i>address</i>][,<b>A</b>|<b>N</b>][,<b>L</b>])}}
[,<b>EXLLEN</b>=<i>number</i>]</td></tr>
</table>

The parameters of the TESTCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. Appendix B, "Operand Notation" on page 129, gives all the ways of coding each parameter for the macros that work at execution.

label
is 1 to 8 characters that provide a symbolic address for the TESTCB macro.

EXLST=*address*
specifies the address of the exit list whose information you want to test. You may omit it only if you're testing the maximum length of an exit list (EXLLEN=number). The TESTCB macro does not support the UPAD user exit.

ERET=*address*
specifies the address of a routine to which VSAM is to give control if, because of an error, it is unable to test for the condition you specify. For example, testing AVSPAC in an access method control block for an unopened data set would fail. VSAM indicates in register 15 whether it could do the test and, if not, indicates in register 0 the reason it couldn't. (The reasons are discussed under "Control Block Manipulation Macro Return Codes and Reason Codes" on page 8.) A failure trying to execute TESTCB indicates a basic logical problem in the processing program, so the error routine would probably issue an ABEND. If it lets the program continue, it must branch to the continuation point itself, and not return to VSAM.

EODAD={**0**|([*address*][,**A**|**N**][,**L**])}|
JRNAD={**0**|([*address*][,**A**|**N**][,**L**])}|
LERAD={**0**|([*address*][,**A**|**N**][,**L**])}|
SYNAD={**0**|([*address*][,**A**|**N**][,**L**])}
specifies the exit about which you are asking a yes-no question. If you code more than one parameter for an exit name, each must equal the corresponding value in the control block for you to get an equal condition. The values that can be tested are:

0 specifies that a test is to be made to determine whether an entry is provided for the exit in the exit list.

address
specifies that a test is to be made to determine whether this is the address of the exit. Tests for an address result in an equal, unequal,

high, low, not-high, or not-low condition. Tests for a combination of an address and A, N, or L result in an equal or unequal condition.

A|N
specifies that a test is to be made to determine whether an exit is active (A) or not active (N). Tests for A or N result in an equal or unequal condition.

L specifies that a test is to be made to determine whether the address is the location of an 8-byte field containing the name of a module to be loaded rather than the entry point of the routine. Tests for L result in either an equal or unequal condition.

EXLLEN=*number*
specifies either the maximum length that an exit list can have (if you don't code the EXLST parameter) or the actual length of the exit list indicated by the EXLST parameter. If you specify an exit, you may not also specify EXLLEN; if you specify EXLLEN, you may not also specify an exit.

If you omit a routine to handle error conditions, you can examine register 15 following TESTCB by using a branch table, for example, but don't alter the PSW condition code that VSAM set to indicate the result of a test until you've had a chance to test it.

Example: TESTCB Macro (Use a Branch Table)

In this example, a TESTCB macro is used to test whether ENDPROC is the routine supplied for the EODAD exit in the exit list EXITS, and whether the EODAD exit is active. A branch table is used to determine whether the test is successful.

```
           TESTCB EODAD=(ENDPROC,A)   Is ENDPROC supplied and
                  EXLST=EXITS         is the exit active?
           B      *+4(15)
```

If the test was made successfully, register 15 contains 0 and the next instruction is executed.

```
           B      TEST1
```

If it was unsuccessful, register 15 contains 4 and the next instruction is executed.

```
           ABEND  2,DUMP
TEST1      BNE    NO
YES        ...                Yes; ENDPROC is supplied and active.
NO         ...                ENDPROC isn't supplied, or the exit
                              isn't active.
```

TESTCB Macro (Test Fields of a Request Parameter List)

The syntax of the TESTCB macro to test fields in a request parameter list is:

[*label*]	**TESTCB**	**RPL**=*address* [,**ERET**=*address*] ,{**AIXFLAG**=**AIXPKP**| **AIXPC**=*number*| **FTNCD**=*number*| **IO**=**COMPLETE**| **OPTCD**=([**ADR**][,**ARD**][,**ASY**][,**BWD**] [,**CNV**][,**DIR**][,**FKS**][,**FWD**] [,**GEN**][,**KEQ**][,**KEY**][,**KGE**][,**LOC**] [,**LRD**][,**MVE**][,**NSP**][,**NUP**][,**SEQ**] [,**SKP**][,**SYN**][,**UPD**])| **ACB**=*address*| **AREA**=*address*| **AREALEN**=*number*| **ARG**=*address*| **ECB**=*address*| **FDBK**=*number*| **KEYLEN**=*number*| **MSGAREA**=*address*| **MSGLEN**=*number*| **NXTRPL**=*address*| **RBA**=*number*| **RECLEN**=*number*| **RPLLEN**=*number*| **TRANSID**=*number*}

The parameters of the TESTCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. Appendix B, "Operand Notation" on page 129, gives all the ways of coding each parameter for the macros that work at execution.

where:

label
is 1 to 8 characters that provide a symbolic address for the TESTCB macro.

RPL=*address*
specifies the address of the request parameter list whose information you want to test. You may omit it only if you're testing the length of a request parameter list (RPLLEN=*number*). (All request parameter lists have the same length.)

ERET=*address*
specifies the address of a routine to which VSAM is to give control if, because of an error, it is unable to test for the condition you specify. For example, testing AVSPAC in an access method control block for an unopened data set would fail. VSAM indicates in register 15 whether it could do the test and, if not, indicates in register 0 the reason it couldn't. (The reasons are discussed under "Control Block Manipulation Macro Return

Codes and Reason Codes" on page 8.) A failure trying to execute TESTCB indicates a basic logical problem in the processing program, so the error routine would probably issue an abend. If it lets the program continue, it must branch to the continuation point itself, and not return to VSAM.

AIXFLAG=AIXPKP
specifies that prime-key pointers are used rather than RBAs.

AIXPC=*number*
specifies the pointer count.

FTNCD=*number*
specifies whether the upgrade set is correct or may have been modified by a request. These codes are described under "Component Codes" on page 11.

IO=COMPLETE
specifies that a test is to be made to determine whether an asynchronous request has been completed. (When you issue a CHECK macro, you suspend processing until a request has been completed if it hasn't yet been completed.)

OPTCD=([,ADR][,ARD][,ASY][,BWD][,CNV][,DIR][,FKS]
[,FWD][,GEN][,KEQ][,KEY][,KGE][,LOC][,LRD]
[,MVE][,NSP][,NUP][,SEQ][,SKP][,SYN][,UPD]
specifies that a test is to be made to determine what subparameter or combination of subparameters is being used for the request. See Figure 16 on page 132 for a description of these subparameters.

The remaining parameters specify fields in a request parameter list and values; the contents of a field are to be compared to the specified value. These fields are the same as those that can be displayed by using a SHOWCB macro. (See Figure 14 for an explanation of these fields.) Fields can be tested at the same time they are displayed.

You may specify only one keyword. If you code a list of option codes (for example, OPTCD=(KEY,DIR)), each of them must equal the corresponding value in the control block for you to get an equal condition.

If you omit a routine to handle error conditions, you can examine register 15 following TESTCB by using a branch table, for example, but don't alter the PSW condition code that VSAM set to indicate the result of a test until you've had a chance to test it.

Example: TESTCB Macro (Test a Request Parameter List)

```
         TESTCB RPL=(3),
                RECLEN=80
         BE     NOCHNGE
CHANGE   ...                    Because the record length in the
                                request parameter list was not 80,
                                the length indicator must be
                                modified so that it is 80.
NOCHNGE  ...                    Because the record length in the
                                request parameter list was 80, no
                                change is required.
```

The TESTCB macro's parameters are:

- RPL specifies that the address of the request parameter list to be tested is contained in register 3.
- RECLEN specifies that the record length indicated in the request parameter list is to be tested to determine whether it is 80.

VERIFY Macro (Synchronize End of Data)

The syntax of the VERIFY macro is:

[*label*]	**VERIFY**	**RPL**=*address*

where:

label
is 1 to 8 characters that provide a symbolic address for the VERIFY macro.

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

The following parameter and subparameter is required for VERIFY:

- In the RPL, **OPTCD=(CNV,...)** must be specified.

Before you can verify a data set that has been extended, you must close it and reopen it to obtain new extent information.

After verifying a data set, positioning must be established with a POINT macro or with a GET macro with RPL OPTCD=DIR.

The VERIFY macro is an invalid function for a linear data set (LDS). Because an LDS has no control information, VSAM cannot examine the contents of the control interval (CI) for the LDS being processed. Therefore, VSAM record management fails any request to verify an LDS.

WRTBFR Macro (Write Buffer)

The syntax of the WRTBFR macro is:

WRTBFR	**RPL=***address* **,TYPE={ALL|CHK|DRBA|DS|LRU(***percent***)|TRN}**

RPL=*address*

specifies the address of the request parameter list that defines the WRTBFR request. An RPL need not be built especially for the WRTBFR—WRTBFR may use an inactive RPL that defines other request(s) (GET, PUT, and so forth) for a data set that is using the resource pool. The following RPL parameters have meaning for WRTBFR:

- **ACB=***address*
- **ARG=***address*

  For TYPE=DRBA, the address of a 4-byte field that contains the RBA to be located and written.

- **ECB=***address*
- **OPTCD={ASY|SYN}**

  WRTBFR can be issued synchronously (SYN) or asynchronously (ASY). A CHECK or ENDREQ must be issued to synchronize an asynchronous WRTBFR request.

- **TRANSID=***number*

  Specifies a number from 0 to 31.

All other RPL parameters are ignored. RPLs are assumed not to be chained.

If the ACB to which the RPL is related has MACRF=GSR, the program that issues WRTBFR must be in supervisor state with protection key 0 to 7.

TYPE={ALL|CHK|DRBA|DS|LRU(*percent***)|TRN}**

specifies which buffers are to be written.

Note: Before using WRTBFR TYPE=CHK|DRBA|TRN, be sure to release all buffers. VSAM defers processing until all buffers are released. For details about releasing buffers, see *VSAM Administration*.

ALL

specifies that all modified, unwritten index and data buffers in each buffer pool in the resource pool are to be written. No buffers will be invalidated. Closing all the data sets that use a resource pool causes the same buffers to be written.

CHK

is as TRN (below), but, if an error occurs in writing buffers, transaction IDs continue to be associated with the buffers. WRTBFR TYPE=CHK could be used by a checkpoint routine to record checkpoint information and leave buffers for which an error occurred as they were for continued processing.

DRBA

specifies that one of the data set's data buffers is to be written. The buffer to be written is identified with the RBA pointed to by the RPL ARG address.

DS

specifies that, for the data set defined by the ACB to which the WRTBFR's RPL is related, all modified, unwritten index and data buffers are to be written and all buffers are marked empty, that is, invalidated. Therefore, WRTBFR TYPE=DS should be issued only after all VSAM requests for the data set have been quiesced. Otherwise, unpredictable results may occur.

LRU(*percent*)

specifies that some of the modified buffers in each buffer pool in the resource pool are to be written. The percent is the percentage of buffers in each pool that are to be examined for possible writing. The least recently used buffers are examined. (If percent is coded in register notation, only registers 1 and 13 may not be used.)

TYPE=LRU is used for writing some modified buffers, without respect to a particular data set or transaction ID, to ensure that buffers will be available for GET requests (without having to wait for buffers to be written).

TRN

specifies that all buffers in a buffer pool that have been modified by requests with the transaction ID specified in the WRTBFR's RPL are to be written. Transaction IDs are no longer associated with these buffers if WRTBFR completes successfully.

Appendix A. List, Execute, and Generate Forms of Macros

BLDVRP, DLVRP, GENCB, MODCB, SHOWCB, and TESTCB macros build a parameter list describing in codes the actions indicated by the operands you specify and pass the list to VSAM to take the indicated action. The list, execute, and generate forms of BLDVRP, DLVRP, GENCB, MODCB, SHOWCB, and TESTCB allow you to write reentrant programs, to share parameter lists, and to modify a parameter list before using it.

Following is a brief description of the list, execute, and generate forms:

- The list form is used to build the parameter list either inline (referred to as a *simple list*) or in an area remote from the macro expansion (referred to as a *remote list*). Both the simple- and the remote-list forms allow you to build a single parameter list that can be shared.
- The execute form is used to modify a parameter list and to pass it to VSAM for action.
- The generate form is used to build the parameter list in a remote area and to pass it to VSAM for action.

The list, execute, and generate forms of the BLDVRP, DLVRP, GENCB, MODCB, SHOWCB, and TESTCB macros have the same syntax as the standard forms, with the exception of:

- An additional keyword, MF
- Keywords that are required in the standard form may be optional in the list, execute, and generate forms or may not be allowed in the execute form. The meaning of the keywords, however, and the notation that may be used to express addresses, names, numbers, and option codes are the same.

The sections that follow describe the syntax of the MF keyword and the use of list, execute, and generate forms. They also indicate the optional and invalid operands.

List-Form Keyword

The syntax of the MF keyword for the list form is:

MF={L|(L,*address*[,*label*])}

where:

L specifies that this is the list form of the macro.

address
specifies the address of a remote area in which the parameter list is to be built. The area must begin on a fullword boundary. You can specify the address in register notation or as an expression valid for a relocatable A-type address constant or a direct or indirect S-type address constant.

label
: is a unique name that is used in an EQU instruction in the expansion of the macro; label is equated to the length of the parameter list. You do not have to know the length of the parameter list if you code label; the expansion of the macro determines the amount of storage required.

Because the MF=L expansion does not include executable code, register notation and expressions that generate S-type address constants cannot be used.

If you code MF=L, the parameter list is built inline, which means that the program is not reentrant if the parameter list is modified at execution.

If you code MF=(L,address), the parameter list is built in the remote area specified, and the area must be large enough for the parameter list.

The size, in fullwords, of a parameter list is:

- For GENCB, 4, plus 3 times the number of ACB, EXLST, or RPL keywords specified (plus 1 for DDNAME, EODAD, JRNAD, LERAD, or SYNAD)
- For MODCB, 3, plus 3 times the number of ACB, EXLST, or RPL keywords specified (plus 1 for DDNAME, EODAD, JRNAD, LERAD, or SYNAD)
- For SHOWCB, 5, plus 2 times the number of fields specified in the FIELDS operand
- For TESTCB, 8 (plus 1 for either DDNAME, STMST, EODAD, JRNAD, LERAD, or SYNAD)

If you code MF=(L,address,label), the parameter list is built in the remote area specified. The expansion of the macro equates label with the length of the parameter list.

Execute-Form Keyword

The syntax of the MF keyword for the execute form is:

MF=(E,*address*)

where:

E specifies that this is the execute form of the macro.

address
: is the address of the parameter list.

The expansion of the execute form of the macro results in executable code that causes:

1. A parameter list to be modified if requested
2. Control to be passed to a routine that satisfies the request

You may not use the execute form to add an entry to a parameter list. If you try to add an entry, you will receive a return code of 8 in register 15.

Generate-Form Keyword

The syntax of the MF keyword for the generate form is:

MF=(G,*address*[,*label*]**)**

where:

G specifies that this is the generate form of the macro.

address
: specifies the address of a remote area in which the parameter list is to be built. The area must begin on a fullword boundary.

label
: is a unique name that is used in an EQU instruction in the expansion of the macro; label is equated to the length of the parameter list. You do not have to know the length of the parameter list if you code label; the expansion of the macro determines the amount of storage required.

If you code MF=(G,address), the parameter list is built in the remote area specified.

If you code MF=(G,address,label), the parameter list is built in the remote area specified. The expansion of the macro equates the length of the parameter list to label.

List, Execute and Generate Formats

List Form of BLDVRP

Note: If FIX is specified, DLVRP must be issued by the same task that issues BLDVRP. STRNO is optional in the list form of BLDVRP, but, if it is not specified, it must be specified in the execute form.

The syntax of the list form of BLDVRP is:

BLDVRP	**BUFFERS=(***size***(***number***),***size***(***number***),...)** **,MF=L** [**,FIX={BFR|IOB|(BFR,IOB)}**] [**,KEYLEN=***length*] [**,STRNO=***number*] [**,TYPE={LSR|GSR}**]

Execute Form of BLDVRP

Note: The address is the address of the parameter list built by a list form of BLDVRP. If you use register notation, you may use register 1, and a register between 2 and 12. Register 1 is used to pass the parameter list to VSAM. BUFFERS may not be specified in the execute form of BLDVRP, because this operand affects the length of the parameter list.

The syntax of the execute form of BLDVRP is:

BLDVRP	**MF=(E,**_address_**)** [**,KEYLEN=**_length_] [**,STRNO=**_number_] [**,TYPE=**{**LSR|GSR**}]

Execute Form of DLVRP

Note: There is no list form for DLVRP, because DLVRP works with BLDVRP: It uses the parameter list associated with BLDVRP. The address is the address of the parameter list built by a list form of BLDVRP. If you use register notation, use register 1 to pass the address of the parameter list to VSAM.

The syntax of the execute form of DLVRP is:

DLVRP	**MF=(E,**_address_**)** **,TYPE=**{**LSR|GSR**}

List Form of GENCB

The syntax of the list form of GENCB is:

[_label_]	**GENCB**	**BLK=**{**ACB|EXLST|RPL**} [**,AM=VSAM**] [**,COPIES=**_number_] [,_keyword_={_address|name|number|option_},...] [**,LENGTH=**_number_] **,MF=**{**L|(L,**_address_[,_label_]**)**} [**,WAREA=**_address_]

Execute Form of GENCB

The syntax of the execute form of GENCB is:

[_label_]	**GENCB**	**BLK=**{**ACB|EXLST|RPL**} [**,AM=VSAM**] [**,COPIES=**_number_] [,_keyword_={_address|name|number|option_},...] [**,LENGTH=**_number_] **,MF=(E,**_address_**)** [**,WAREA=**_address_]

Generate Form of GENCB

The syntax of the generate form of the GENCB macro is:

<table>
<tr><td>[label]</td><td>GENCB</td><td>BLK={ACB|EXLST|RPL}
[,AM=VSAM]
[,COPIES=number]
[,keyword=address|name|number|option},...]
[,LENGTH=number]
,MF=(G,address[,label])
[,WAREA=address]</td></tr>
</table>

List Form of MODCB

The syntax of the list form of MODCB is:

<table>
<tr><td>[label]</td><td>MODCB</td><td>{ACB|EXLST|RPL{=address
,keyword={address|name|number|option},...
,MF={L|(L,address[,label])}</td></tr>
</table>

Execute Form of MODCB

Note: If the execute form of MODCB is used and EXLST is used as a keyword to be processed, the block must be identified by ACB=.

The syntax of the execute form of MODCB is:

<table>
<tr><td>[label]</td><td>MODCB</td><td>[{ACB|EXLST|RPL}=address]
[,keyword={address|name|number|option},...]
,MF=(E,address)</td></tr>
</table>

Generate Form of MODCB

The syntax of the generate form of MODCB is:

<table>
<tr><td>[label]</td><td>MODCB</td><td>{ACB|EXLST|RPL{=address
,keyword={address|name|number|option},...
,MF=(G,address[,label])</td></tr>
</table>

List Form of SHOWCB

The syntax of the list form of SHOWCB is:

<table>
<tr><td>[label]</td><td>SHOWCB</td><td>[{ACB|EXLST|RPL}=address]
,AREA=address
,FIELDS=(keyword[,keyword,...])
,LENGTH=number
,MF={L|(L,address[,label])}
,OBJECT={DATA|INDEX}]</td></tr>
</table>

Execute Form of SHOWCB

The syntax of the execute form of SHOWCB is:

[*label*]	**SHOWCB**	[{**ACB**|**EXLST**|**RPL**}=*address* ,**AREA**=*address* ,**MF=(E,***address***)** [,**OBJECT**={**DATA**|**INDEX**}]

Generate Form of SHOWCB

The syntax of the generate form of SHOWCB is:

[*label*]	**SHOWCB**	[{**ACB**|**EXLST**|**RPL**}=*address*] ,**AREA**=*address* ,**FIELDS=(***keyword*[,*keyword*,...]**)** ,**LENGTH**=*number* ,**MF=(G,***address*[,*label*]**)** [,**OBJECT**={**DATA**|**INDEX**}]

List Form of TESTCB

Note: If the execute form of TESTCB is used and EXLST is used as a keyword to be processed, the block must be identified by ACB=.

The syntax of the list form of TESTCB is:

[*label*]	**TESTCB**	[{**ACB**|**EXLST**|**RPL**}=*address*] [,**ERET**=*address*] ,*keyword*={*address*|*name*|*number*|*option*} ,**MF**={**L**|**(L,***address*[,*label*]**)**} [,**OBJECT**={**DATA**|**INDEX**}]

Execute Form of TESTCB

Note: If the execute form of TESTCB is used and EXLST is used as a keyword to be processed, the block must be identified by ACB=.

The syntax of the execute form of TESTCB is:

[*label*]	**TESTCB**	[{**ACB**|**EXLST**|**RPL**}=*address*] [,**ERET**=*address*] [,*keyword*={*address*|*name*|*number*| *option*}] ,**MF=(E,***address***)** [,**OBJECT**={**DATA**|**INDEX**}]

Generate Form of TESTCB

The syntax of the generate form of TESTCB is:

[*label*]	**TESTCB**	[{**ACB**|**EXLST**|**RPL**}=*address*] [,**ERET**=*address*] ,*keyword*={*address*|*name*|*number*|*option*} ,**MF=(G**,*address*[,*label*]**)** [,**OBJECT**={**DATA**|**INDEX**}]

Use of List, Execute, and Generate Forms

Figure 15 indicates which forms of GENCB, MODCB, SHOWCB, and TESTCB should be used in reentrant/nonreentrant and shared/nonshared environments.

	Reentrant	**Nonreentrant**

Shared	MF=(L,*address*[,*label*])
	MF=(E,*address*)
		MF=L
		MF=(E,*address*)

Nonshared	MF=(G,*address*[,*label*])	Standard Form

Figure 15. Reentrant Programming

The figure shows that:

- To share parameter lists in a reentrant program, the remote-list form should be used in conjunction with the execute form.
- To share parameter lists in a nonreentrant program, the simple-list form should be used in conjunction with the execute form.
- If you do not intend to share parameter lists, the generate form should be used in reentrant programs and the standard form should be used for nonreentrant programs.

Examples of Generate, List, and Execute Forms in Reentrant Environments

The examples that follow illustrate how the list, execute, and generate forms work.

Example: Generate Form (Reentrant)

In this example, the generate form of GENCB is used to create a default request parameter list (RPL) in a reentrant environment.

```
LA         10,LEN1          Get length of the parameter list.
GETMAIN    R,LV=(10)        Get storage for the area in which
                            the parameter list is to be built.
LR         2,1              Save address of parameter-list
                            area.
GENCB      BLK=RPL,
           MF=(G,(2),LEN1)
```

The macro expansion equates LEN1 to the length of the parameter list, as follows:

```
+LEN1 EQU 16
```

The parameter list will be built in the area acquired by the GETMAIN macro and pointed to by register 2. This list is used by VSAM to build the RPL. VSAM returns the RPL address in register 1 and the RPL length in register 0. If the WAREA and LENGTH parameters are used, the RPL will be built at the WAREA address.

Example: Remote-List Form (Reentrant)

In this example, the remote-list form of MODCB is used to build a parameter list that will later be used to modify the MACRF bits in the access method control block ANYACB.

```
LA         8,LEN2             Get length of the parameter
                              list.
GETMAIN    R,LV=(8)           Get storage for the area in
                              which the parameter list is to
                              be built.
LR         3,1                Save address of the
                              parameter-list area.
MODCB      ACB=ANYACB,
           MACRMF=(L,(3),LEN2)
```

The macro expansion equates the length of the parameter list to LEN2, as follows:

```
+LEN2 EQU 24
```

This parameter list is built in the remote area pointed to by register 3. The list will be used by VSAM to modify the ACB when an execute form of MODCB is issued (see next example). The list form only creates a parameter list; it does not modify the ACB.

Example: Execute Form (Reentrant)

In this example, the execute form of MODCB is used to modify the address of the access method control block and MACRF codes in the parameter list created by the remote-list form of MODCB in the previous example.

```
MODCB  ACB=MYACB,MACRF=(ADR,SEQ,OUT),MF=(E,(3))
```

The parameter list pointed to by register 3 is changed so that the ACB and MACRF parameter values in the execute form override those in the list form. The access method control block, MYACB, is then modified to MACRF=ADR,SEQ,OUT). The access method control block at ANYACB is not changed by either of these examples.

Appendix B. Operand Notation

Operands with GENCB, MODCB, SHOWCB, and TESTCB

The addresses, names, numbers, and options required with operands in GENCB, MODCB, SHOWCB, and TESTCB can be expressed in a variety of ways:

- An **absolute numeric expression**, for example, STRNO=3 and COPIES=10.
- A **code or a list of codes separated by commas and enclosed in parentheses**, for example, OPTCD=KEY or OPTCD=(KEY,DIR,IN).
- A **character string**, for example, DDNAME=DATASET.
- A **register from 2 through 12 that contains an address or numeric value**, for example, SYNAD=(3). Equated labels can be used to designate a register, for example, SYNAD=(ERR), where the following equate statement has been included in the program: ERR EQU 3.
- An **expression of the form (S,scon)**, where scon is an expression valid for an S-type address constant, including the base-displacement form. The contents of the base register will be added to the displacement to obtain the value of the keyword. For example, if the value of the keyword being represented is a numeric value (that is, COPIES, LENGTH, RECLEN), the contents of the base register will be added to the displacement to determine the numeric value. If the value of the keyword being represented is an address constant (that is, WAREA, EXLST, EODAD, ACB), the contents of the base register will be added to the displacement to determine the value of the address constant.
- An **expression of the form (*,scon)**, where scon is an expression valid for an S-type address constant, including the base-displacement form. The address specified by scon is **indirect**, that is, it is the address of an area that contains the value of the keyword. The contents of the base register will be added to the displacement to determine the address of the fullword of storage that contains the value of the keyword.

  If an indirect S-type address constant is used, the value it points to must meet the following criteria:

  - If it is a numeric quantity or an address, it must occupy a fullword of storage.
  - If it is an alphameric character string, it must occupy two words of storage, be left aligned, and be filled on the right with blanks.
- An **expression valid for a relocatable A-type address constant**, for example, AREA=MYAREA+4.

The specified keyword determines the type of expressions that can be used. Additionally, register and S-type address constants cannot be used when MF-L is specified.

The tables containing the individual macro operand notations have been deleted from this release. This information may be obtained from the individual macro descriptions shown in Chapter 2, "VSAM Macro Formats and Examples".

Appendix C. Building Parameter Lists

The standard forms of GENCB, MODCB, SHOWCB, and TESTCB build a parameter list, put its address in register 1, and pass control to a VSAM routine to generate, modify, display, or test an access method control block, exit list, or request parameter list. Other forms of the macros only build the parameter list (list forms) or only pass control to VSAM (execute forms).

You can avoid using a macro to build the parameter list by building it yourself and issuing the execute form of the macro to pass control to VSAM. This chapter explains how to build the parameter lists for GENCB, MODCB, SHOWCB, and TESTCB. The rules for combinations of codes in a parameter list are the same as the rules for combinations of operands in a macro.

You can avoid issuing the execute form of the macro by coding the linkage instructions that pass control directly to the VSAM control block manipulation routine. Before passing control, you must build the parameter list yourself.

The Format of the Parameter Lists

A parameter list for GENCB, MODCB, SHOWCB, or TESTCB is a list of fullword addresses. The first address points to a header entry that identifies the type of request and type of control block and gives other general information about the request. Each of the rest of the addresses in the parameter list points to an element entry that identifies the information you want to generate, modify, display, or test.

The fullwords in the parameter list must be contiguous, and the last one must have a 1 in its first bit. The header entry and each element entry may be separate from each other. Figure 16 on page 132 gives the formats of the header and element entries for the four request types.

Building Header and Element Entries

Five assembler macros are provided for building entries.
IDAGENC, IDAMODC, IDASHOW, and IDATEST help you build a header entry for generation, modification, display, or test. IDAELEM helps you build an element entry.

[*label*]	**IDAGENC**	**[DSECT={YES|NO}]**
[*label*]	**IDAMODC**	**[DSECT={YES|NO}]**
[*label*]	**IDASHOW**	**[DSECT={YES|NO}]**
[*label*]	**IDATEST**	**[DSECT={YES|NO}]**
[*label*]	**IDAELEM**	**[DSECT={YES|NO}]**

DSECT={YES|NO}
Indicates whether a DSECT statement is to be generated. If you intend to build entries in a continuous area, you could have only the first of the macros generate a DSECT statement and use a single register to address the whole area.

Header Entry / Element Entry

Generation GENCB

	Header Entry		
0	GENBTC Control-block type	GENFTC Function type	GENCOP Number of copies of block
4	GENUSA (optional) Address of area provided for generation		
8	GENUSL (optional) Length of area	(reserved)	

	Element Entry	
0	ELEMKWTC Keyword type	(reserved)
4	Keyword value	
8	(required for some keywords)*	

Modification MODCB

	Header Entry		
0	MODBTC Control-block type	MODFTC Function type	(reserved)
4	MODBLAD Address of control block to be modified		

	Element Entry	
0	ELEMKWTC Keyword type	(reserved)
4	Keyword value	
8	(required for some keywords)*	

Display SHOWCB

	Header Entry		
0	SHOWBTC (optional) Control-block type	SHOWFTC Function type	SHOWOBJ Object type (data or index)
4	SHOWBLAD (optional) Address of control block to be displayed		
8	SHOWUSA Address of area provided for display		
12	SHOWUSL Length of area	(reserved)	

	Element Entry	
0	ELEMKWTC Keyword type	(reserved)

Test TESTCB

	Header Entry		
0	TESTBTC (optional) Control-block type	TESTFTC Function type	TESTOBJ Object type (data or index)
4	TESTBLAD (optional) Address of control block to be tested		
8	TESTERET (optional) Address of error-analysis routine (ERET)		
12	(reserved)		

	Element Entry	
0	ELEMKWTC Keyword type	(reserved)
4	Keyword value	
8	(required for some keywords)*	

*Second fullword required for keyword value of DDNAME, STMST, EODAD, JRNAD, LERAD, and SYNAD.

Figure 16. Format of Header and Element Entries for GENCB, MODCB, SHOWCB, and TESTCB Parameter Lists

These macros generate labeled DS statements that give the layout of an entry and EQU statements that equate a label with a numeric code. You can symbolically encode an entry with a series of move instructions. The macros are self-documenting— inspect a listing of their expansions and you can see which labels to code in your move instructions. (You can list the macros as they appear in the macro library.)

To generate an exit list with LERAD and SYNAD exits, you could code a GENCB of the standard form:

GENCB BLK=EXLST,LERAD=(LOGERR,L),SYNAD=PHYSERR

The following example shows how to achieve the same effect by building the parameter list and entries yourself and issuing a GENCB of the execute form.

```
LA      5,NTRYAREA      Set up base register for the
                        entries.
USING   5,GENC          GENC is the first label in the
                        work area.
```

Build the list of addresses that point to the entries.

```
ST      5,PLIST         Address of the header entry.
LA      6,GENLEN(,5)    Address of the first element entry.
                        GENLEN is equated to the length of
                        a header element for generation.
ST      6,PLIST+4
LA      6,ELEMLLEN(,6)  Address of the second element entry.
                        ELEMLLEN is equated to the length
                        of an element entry for an exit list.
ST      6,PLIST+8
OI      PLIST+8,X'80'   End-of-list indicator.
```

Build the header entry.

```
MVI    GENBTC,GENXLST   Indicate the blocktype-exit list.
MVI    GENFTC,GENFTYP   Indicate the function
                        type-generation.
MVI    GENCOP+1,X'01'   Indicate the number of copies
                        of the exit list to be generated.
MVI    ELEMKWTC+1,      Indicate the keyword
       ELEMLEAD         type-LERAD.
LA     6,LOGERR         Address of the name of the
                        logical error analysis module.
ST     6,ELEMPTR
MVI    ELEMXFLG,        Indicate the presence of an
       ELEMXL+ELEMXADR  address ELEMPTR and that the
                        exit routine is to be loaded.
```

Build the second element entry.

```
LA     5,ELEMLLEN(,5)   Align the DSECT with the
                        second element entry. ELEMLLEN
                        is equated to the length of an
                        element entry for an exit list.
MVI     ELEMKWTC+1,     Indicate the keyword
        ELEMSYAD        type-SYNAD.
LA      6,PHYSERR       Address of the entry point of the
                        physical error analysis routine.
ST      6,ELEMPTR
MVI     ELEMXFLG,       Indicate the presence of an
        ELEMXADR        address in ELEMPTR.
```

Pass control to VSAM.

```
          GENCB   MF=(E,PLIST)
          LTR     15,15           Generation successful?
          BNZ     CHECKO          No.
   ⋮
CHECKO    ABEND   1,DUMP          Register 0 indicates the error.
```

Physical error analysis exit routine.

```
PHYSERR . . .
```

Work areas and constants.

```
LOGERR    DC      CL8'LEMOD'      Name of the logical error
                                  analysis module to be loaded.
PLIST     DC      3F'0'           List of entry addresses.
                                  3 addresses are required:
                                  1 for the header and 2 for
                                  the elements (1 for LERAD and
                                  1 for SYNAD).
NTRYAREA  DC      9F'0'           Work area for header and
                                  element entries. The header
                                  for GENCB is 3 fullwords,
                                  and so are the LERAD
                                  and SYNAD elements.
```

DSECT with labels for the header and element entries.

```
          IDAGENC                 Header entry. A DSECT
                                  statement is generated, and
                                  register 5 is used to address
                                  NTRYAREA with these labels.
          IDAELEM DSECT=NO        Element entry. Element labels
                                  are part of the same DSECT as
                                  the header labels.
```

Passing Control Directly to VSAM

You can avoid using the execute form of GENCB, MODCB, SHOWCB, and TESTCB by building your own linkage instructions. You first build a parameter list, as described in the previous section, and put its address in register 1. Then you pass control to VSAM using the following instructions:

```
L       15,16              Put the address of the CVT
                           into register 15.
L       15,256(,15)        Put the address of the AMCBS
                           control block into register 15.
L       15,12(,15)         Put the address of the control
                           block manipulation routine into
                           register 15.
BALR    14,15
          or               Branch to the routine
BAL     14,xx(,15)
```

The BALR 14,15 instruction is used when the specific function (GENCB, MODCB, SHOWCB, or TESTCB) is not known, or when the control block type (ACB, EXLST, or RPL) is not known. The user-built parameter list contains the function code and control block type code.

The BAL 14,xx(,15) instruction is used to increase your program's performance. The "xx" is a decimal value that identifies a function (GENCB, MODCB, SHOWCB, or TESTCB) and a control block type (ACB, EXLST, or RPL).

Decimal Value of xx	Function	Control Block
8	GENCB	ACB
12	GENCB	RPL
16	GENCB	EXLST
20	Reserved	
24	MODCB	ACB
28	MODCB	RPL
32	MODCB	EXLST
36	Reserved	
40	SHOWCB	ACB
44	SHOWCB	RPL
48	SHOWCB	EXLST
52	Reserved	
56	TESTCB	ACB
60	TESTCB	RPL
64	TESTCB	EXLST
68	Reserved	
72	SHOWCB or TESTCB	Block length keywords only
76[1]	SHOWCB	RECLEN field of an RPL
80[1]	MODCB	RECLEN field of an RPL

[1] Register 1 points to an RPL when xx is 76 or 80. See the following section for details.

When VSAM returns to your program, register 15 contains a completion code. Register 15 contains a zero value if the task completed successfully. Otherwise, register 15 and register 0 contain codes that identify the reason VSAM could not complete the task.

Modifying and Displaying the RECLEN Field of an RPL Directly

You can modify or display the RECLEN field (that is, the record length) of an RPL without issuing a SHOWCB or MODCB macro, and without building a parameter list.

To modify a RPL's RECLEN field, you first put the address of the RPL in register 1, and the value to be set in the RECLEN field in register 0. Next, you code the instructions that put the address of the VSAM control block manipulation routine into register 15, then branch to the routine:

```
L      15,16              Put the address of the CVT into
                          register 15.
L      15,256(,15)        Put the address of the AMCBS
                          control block into register 15.
L      15,12(,15)         Put the address of the control
                          block manipulation routine
                          into register 15.
BAL    14,80(,15)         Branch to the routine.
```

When VSAM returns to your program, register 15 contains a completion code. Register 15 contains a zero value if the field was modified correctly. Otherwise, register 15 and register 0 contain codes that identify the reason VSAM could not complete the task.

To display the contents of a RPL's RECLEN field, you first put the address of the RPL in register 1. Next, you code the instructions that put the address of the VSAM control block manipulation routine into register 15, and then branch to the routine:

```
L       15,16             Put the address of the CVT into
                          register 15.
L       15,256(,15)       Put the address of the AMCBS
                          control block into register 15.
L       15,12(,15)        Put the address of the control
                          block manipulation routine into
                          register 15.
BAL     14,76(,15)        Branch to the routine.
```

When VSAM returns to your program, register 15 contains a completion code. Register 15 contains a zero value if the field is displayed correctly, and register 0 contains the value of the RPL's RECLEN field. When register 15 is not zero, register 15 and register 0 contain codes that identify the reason VSAM could not complete the task.

Glossary of Terms and Abbreviations

The following terms are defined as they are used in this book. If you do not find the term you are looking for, see the index or the *IBM Dictionary of Computing,* SC20-1699.

access method services. A multifunction service program that is used to define VSAM data sets and allocate space for them, convert indexed-sequential data sets to key-sequenced data sets, modify data set attributes in the catalog, reorganize data sets, facilitate data portability between operating systems, create backup copies of data sets, help make inaccessible data sets accessible, list the records of data sets and catalogs, define and build alternate indexes, and convert OS CVOLs and VSAM catalogs to integrated catalog facility catalogs.

acquire. To allocate space on a staging drive and to stage data from an MSS cartridge to the staging drive.

addressed-direct access. The retrieval or storage of a data record identified by its RBA, independent of the record's location relative to the previously retrieved or stored record. (*See also* keyed-direct access, addressed- sequential access, and keyed-sequential access.)

addressed-sequential address. The retrieval or storage of a data record in its entry sequence relative to the previously retrieved or stored record. (*See also* keyed-sequential access, addressed-direct access, and keyed-direct access.)

alternate index. A collection of index entries organized by the alternate keys of its associated base data records. It provides an alternate means of locating records in the data component of a cluster on which the alternate index is based.

alternate index cluster. The data and index components of an alternate index.

alternate key. One or more consecutive characters taken from a data record and used to build an alternate index or to locate one or more base data records via an alternate index. (*See also* generic key, key, and key field.)

APF. (*See* authorized program facility.)

application. As used in this publication, the use to which an access method is put or the end result that it serves; contrasted to the internal operation of the access method.

authorized program facility. A facility that permits the identification of programs that are authorized to use restricted functions.

base cluster. A key-sequenced or entry-sequenced data set over which one or more alternate indexes are built.

base RBA. The RBA stored in the header of an index record that is used to calculate the RBAs of data or index control intervals governed by the index record.

catalog. (*See* master catalog and user catalog.)

catalog recovery area. An entry-sequenced file that exists on each volume owned by a recoverable catalog, including the catalog itself. The CRA contains records that are duplicates of the catalog entries describing the volume and the files it contains.

CBIC. Control blocks in common, a facility that allows a user to open a VSAM data set so the VSAM control blocks are placed in the common service area (CSA) of the MVS operating system. This provides the capability for multiple memory accesses to a single VSAM control structure for the same VSAM data set.

chained RPL. (*See* RPL string.)

CI. (*See* control interval.)

CIDF. (*See* control interval definition field.)

cluster. A named structure consisting of a group of related components (for example, a data component with its index component). A cluster may consist of a single component. (*See also* base cluster and alternate index cluster.)

collating sequence. An ordering assigned to a set of items, such that any two sets in that assigned order can be collated.

component. A named, cataloged collection of stored records. A component, the lowest member of the hierarchy of data structures that can be cataloged, contains no named subsets.

control area. A group of control intervals used as a unit for formatting a data set before adding records to it. Also, in a key-sequenced data set, the set of control intervals pointed to by a sequence-set index record; used by VSAM for distributing free space and for placing a sequence-set index record adjacent to its data.

control area split. The movement of the contents of some of the control intervals in a control area to a newly created control area, to facilitate the insertion or lengthening of a data record when there are no remaining free control intervals in the original control area.

control interval. A fixed-length area of auxiliary storage space in which VSAM stores records. It is the unit of information transmitted to or from auxiliary storage by VSAM.

control interval access. The retrieval or storage of the contents of a control interval.

control interval definition field. In VSAM, the 4-byte control information field at the end of a control interval that gives the displacement from the beginning of the control interval to free space and the length of the free space. If the length is 0, the displacement is to the beginning of the control information.

control interval split. The movement of some of the stored records in a control interval to a free control interval, to facilitate the insertion or lengthening of a record that won't fit in the original control interval.

control volume. A volume that contains one or more indexes of the catalog.

CRA. (*See* catalog recovery area.)

cross memory. A synchronous method of communication between address spaces.

CVOL. (*See* control volume.)

DASD. (*See* direct access storage device.)

data record. A collection of items of information from the standpoint of its use in an application, as a user supplies it to VSAM for storage.

data 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.)

DD statement. data definition statement

direct access. The retrieval or storage of data by a reference to its location in a data set rather than relative to the previously retrieved or stored data. (*See also* addressed-direct access and keyed-direct access.)

direct access storage device. A device in which the access time is effectively independent of the location of the data.

EBDIC. Extended binary-coded decimal interchange code. A coded character set consisting of 8-bit coded characters.

entry sequence. The order in which data records are physically arranged (according to ascending RBA) in auxiliary storage, without respect to their contents. (Contrast with key sequence.)

entry-sequenced data set. A data set whose records are loaded without respect to their contents, and whose RBAs cannot change. Records are retrieved and stored by addressed access, and new records are added at the end of the data set.

EOD. end of data

EOKR. end-of-key range

EOV. end of volume

field. In a record or a control block, a specified area used for a particular category of data or control information.

free control interval pointer list. In a sequence-set index record, a vertical pointer that gives the location of a free control interval in the control area governed by the record.

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.

GENDSP. An option of LOCATE to obtain the control interval number of the catalog record of each object.

generation data group. A collection of data sets that are kept in chronological order; each data set is called a generation data set.

generic key. A high-order portion of a key, containing characters that identify those records that are significant for a certain application. For example, it might be desirable to retrieve all records whose keys begin with the generic key AB, regardless of the full key values.

global shared resources. An option for sharing I/O buffers, I/O-related control blocks, and channel programs among VSAM data sets in a resource pool that serves all address spaces in the system.

GSR. (*See* global shared resources.)

header, index record. In an index record, the 24-byte field at the beginning of the record that contains control information about the record.

header entry. In a parameter list of GENCB, MODCB, SHOWCB, or TESTCB, the entry that identifies the type of request and control block and gives other general information about the request.

horizontal pointer. In the header of an index record, the RBA of the index record in the same level as this one that contains keys next in ascending sequence after the keys in this one.

ICF. (*See* integrated catalog facility.)

index. As used in this publication, an ordered collection of pairs, each consisting of a key and a pointer, used by VSAM to sequence and locate the records of a key-sequenced data set.

index level. A set of index records that order and give the location of all the control intervals in the next lower level or in the data set that it controls.

index record. A collection of index entries that are retrieved and stored as a group. (Contrast to data record.)

index record header. In an index record, the 24-byte field at the beginning of the record that contains control information about the record.

index replication. The use of an entire track of direct access storage to contain as many copies of a single index record as possible; reduces rotational delay.

index set. The set of index levels above the sequence set. The index set and the sequence set together comprise the index.

integrated catalog facility. The name of the catalog associated with the Data Facility Licensed program.

ISAM. indexed sequential access method

ISAM interface. A set of routines that allow a processing program coded to use ISAM (indexed sequential access method) to gain access to a key-sequenced data set.

JCL. (*See* job control language.)

job catalog. A catalog made available for a job by means of the JOBCAT DD statement.

job control language. A problem-oriented language designed to express statements in a job that are used to identify the job or describe its requirements to an operating system.

job step catalog. A catalog made available for a job by means of the STEPCAT DD statement.

key. One or more characters within an item of data that are used to identify it or control its use. As used in this publication, one or more consecutive characters taken from a data record, used to identify the record and establish its order with respect to other records. (*See also* key field and generic key.)

key 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 VSAM file (data set) whose records are loaded in key sequence and controlled by an index. Records are retrieved and stored by keyed access or by addressed access, and new records are inserted in key sequence by means of distributed free space. Relative byte addresses of records can change because of control interval or control area splits.

keyed-direct access. The retrieval or storage of a data record by use of either an index that relates the record's key to its relative location in the data set or a relative record number, independent of the record's location relative to the previously retrieved or stored record. (*See also* addressed-direct access, keyed-sequential access, and addressed-sequential access.)

keyed-sequential access. The retrieval or storage of a data record in its key or relative record sequence relative to the previously retrieved or stored record, as defined by the sequence set of an index. (*See also* addressed-sequential access, keyed-direct access, and addressed-direct access.)

| **LDS**. (*See* linear data set.)

level number. For the index of a key-sequenced data set, a binary number in the header of an index record that indicates the index level to which the record belongs.

| **linear data set**. a linearly ordered data set whose
| order is preserved in storage by using sequential allo-
| cation.

local shared resources. An option for sharing I/O buffers, I/O-related control blocks, and channel programs among VSAM data sets in a resource pool that serves one partition or address space.

LSR. (*See* local shared resources.)

master catalog. A catalog that contains extensive data set and volume information that VSAM requires to locate data sets, to allocate and deallocate storage space, to verify the authorization of a program or operator to gain access to a data set, and to accumulate usage statistics for data sets.

operating system. Software that controls the execution of programs; an operating system may provide services such as resource allocation, scheduling, input/output control, and data management.

password. A unique string of characters stored in a catalog that a program, a computer operator, or a terminal user must supply to meet security requirements before a program gains access to a data set.

path. A named, logical entity composed of one or more clusters (an alternate index and its base cluster, for example).

physical record. A physical unit or recording on a medium. For example, the physical unit between address markers on a disk.

pointer. An address or other indication of location. For example, an RBA is a pointer that gives the relative location of a data record or a control interval in the data set to which it belongs.

prime index. The index component of a key-sequenced data set that has one or more alternate indexes. (*See also* index and alternate index.)

prime key. (*See* key.)

QSAM. (*See* queued sequential access method.)

queued sequential access method. An extended version of the basic sequential access method (BSAM). When this method is used, a queue is formed of input data blocks that are awaiting processing or output data blocks that have been processed and are awaiting transfer to auxiliary storage or to an output device.

RACF. Resource Access Control Facility.

random access. (*See* direct access.)

RBA. Relative byte address. The displacement (expressed as a fullword binary integer) of a data record or a control interval from the beginning of the data set to which it belongs; independent of the manner in which the data set is stored.

RDF. (*See* record definition field.)

record. (*See* index record, data record.)

record definition field. A field stored as part of a stored record segment; it contains the control information required to manage stored record segments within a control interval.

relative byte address. (*See* RBA.)

relative record data set. A data set whose records are loaded into fixed-length slots.

relative record number. A number that identifies not only the slot, or data space, in a relative record data set but also the record occupying the slot. Used as the key for keyed access to a relative record data set.

replication. (*See* index replication.)

resource pool, VSAM. (*See* VSAM resource pool.)

reusable data set. A VSAM data set that can be reused as a work file, regardless of its old contents. Must not be a base cluster.

RPL string. A set of chained RPLs (the set may contain one or more RPLs) used to gain access to a VSAM data set by action macros (GET, PUT, etc). Two or more RPL strings may be used for concurrent direct or sequential requests made from a processing program or its subtasks.

SAM. (*See* sequential access method.)

security. (*See* data security.)

sequence checking. The process of verifying the order of a set of records relative to some field's collating sequence.

sequence set. The lowest level of the index of a key-sequenced data set; it gives the locations of the control intervals in the data set and orders them by the key sequence of the data records they contain. The sequence set and the index set together comprise the index.

sequential access. The retrieval or storage of a data record in either its entry sequence, its key sequence, or its relative record number sequence, relative to the previously retrieved or stored record. (*See also* addressed-sequential access and keyed-sequential access.)

sequential access method. An access method for storing or retrieving data blocks in a continuous sequence, using either a sequential access or a direct access device.

shared resources. A set of functions that permit the sharing of a pool of I/O-related control blocks, channel programs, and buffers among several VSAM data sets open at the same time.

skip-sequential access. Keyed-sequential retrieval or storage of records here and there throughout a data set, skipping automatically to the desired record or collating position for insertion: VSAM scans the sequence set to find a record or a collating position. Valid for processing in ascending sequences only.

slot. For a relative record data set, the data area addressed by a relative record number which may contain a record or be empty.

spanned record. A logical record whose length exceeds control interval length, and as a result, crosses, or spans, one or more control interval boundaries within a single control area.

SRB. Service request block. A system control block used for dispatching tasks.

step catalog. A catalog made available for a step by means of the STEPCAT DD statement.

terminal monitor program. In TSO, a program that accepts and interprets commands from the terminal, and causes the appropriate command processors to be scheduled and executed.

time sharing option. An optional configuration of the operating system that provides conversational time sharing from remote stations.

TMP. (*See* terminal monitor program.)

transaction ID. A number associated with each of several request parameter lists that define requests belonging to the same data transaction.

TSO. (*See* time sharing option.)

update number. For a spanned record, a binary number in the second RDF of a record segment that indicates how many times the segments of a spanned record should be equal. An inequality indicates a possible error.

upgrade set. All the alternate indexes that VSAM has been instructed to update whenever there is a change to the data component of the base cluster.

user buffering. The use of a work area in the processing program's address space for an I/O buffer; VSAM transmits the contents of a control interval between the work area and direct access storage without intermediary buffering.

user catalog. An optional catalog used in the same way as the master catalog and pointed to by the master catalog. It also lessens the contention for the master catalog and facilitates volume portability.

vertical pointer. A pointer in an index record of a given level that gives the location of an index record in the next lower level or the location of a control interval in the data set controlled by the index.

virtual storage access method. An access method for direct or sequential processing of fixed and variable-length records on direct access devices. The records in a VSAM data set or file can be organized in logical sequence by a key field (key sequence), in the physical sequence in which they are written on the data set or file (entry sequence), or by relative record number.

VSAM. (*See* virtual storage access method.)

VSAM resource pool. A virtual storage area that is used to share I/O buffers, I/O-related control blocks, and channel programs among VSAM data sets. A resource pool is local or global; it serves tasks in one partition or address space or tasks in all address spaces in the system.

VSAM shared information. Blocks that are used for cross-system sharing.

VSI. (*See* VSAM shared information.)

Index

MVS/370
VSAM Administration
Macro Instruction Reference

Reader's Comment Form

GC26-4074-2

This manual is part of a library that serves as a reference source for system analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: Do not use this form to request IBM publications. If you do, your order will be delayed because publications are not stocked at the address printed on the reverse side. Instead, you should direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

If you have applied any technical newsletters (TNLs) to this book, please list them here: ____________________

Comments (please include specific chapter and page references) :

If you want a reply, please complete the following information:

Name ____________________ Date ____________________

Company ____________________ Phone No. (______) ____________________

Address ____________________

____

Thank you for your cooperation. No postage is necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail them directly to the address in the Edition Notice on the back of the title page.)

Note: Staples can cause problems with automatic mail-sorting equipment.
Please use pressure-sensitive or other gummed tape to seal this form.

GC26-4074-2

Reader's Comment Form

Fold and tape Please do not staple Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 40 ARMONK, NY

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department J58
P.O. Box 49023
San Jose, CA 95161-9945

Fold and tape Please do not staple Fold and tape

IBM®

MVS/370
VSAM Administration
Macro Instruction Reference

Reader's Comment Form

GC26-4074-2

This manual is part of a library that serves as a reference source for system analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: Do not use this form to request IBM publications. If you do, your order will be delayed because publications are not stocked at the address printed on the reverse side. Instead, you should direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

If you have applied any technical newsletters (TNLs) to this book, please list them here: ____________________

Comments (please include specific chapter and page references) :

If you want a reply, please complete the following information:

Name ____________________ Date ____________________

Company ____________________ Phone No. (______) ____________________

Address ____________________

____

Thank you for your cooperation. No postage is necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail them directly to the address in the Edition Notice on the back of the title page.)

Note: Staples can cause problems with automatic mail-sorting equipment.
Please use pressure-sensitive or other gummed tape to seal this form.

Reader's Comment Form

Fold and tape   Please do not staple   Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS MAIL   PERMIT NO. 40   ARMONK, NY

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department J58
P.O. Box 49023
San Jose, CA 95161-9945

Fold and tape   Please do not staple   Fold and tape

IBM®

Program Number
5665-295

File Number
S370-34

GC26-4074-2

Printed in U.S.A.