GC26-4056-1
File No. S370-30

Program Product

MVS/370 System Programming Library: Data Management

Data Facility Product 5665-295

Release 1.1

IBM

Second Edition (October 1983)

This is a major revision of, and makes obsolete, GC26-4056-0.

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

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

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

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

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

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

PREFACE

This publication provides information for system programmers about MVS/370 Data Facility Product, and how to modify and extend the data management capabilities of the operating system.

ORGANIZATION

This publication contains 11 chapters and 5 appendixes:

- "Chapter 1. Using Catalog Management Macro Instructions," contains information on the macro instructions used for retrieving catalog information from OS CVOLs, and for adding, deleting, and updating catalog entries for non-VSAM data sets.
- "Chapter 2. Maintaining the Volume Table of Contents (VTOC)," describes the structure of the VTOC and VTOC index, and discusses how to use system macros to read a data set control block, rename a data set, or delete a data set from the VTOC.
- "Chapter 3. Executing Your Own Channel Programs (EXCP)," describes how to use the EXCP macro to control device characteristics and data organization with your own channel programs.
- "Chapter 4. Using XDAP to Read and Write to Direct Access Devices," describes how to use the XDAP macro to read, verify, and update blocks without using an access method.
- "Chapter 5. Password Protecting Your Data Sets," contains information on system password protection and how to create and maintain the PASSWORD data set.
- "Chapter 6. Exit Routines," describes some of the IBM-supplied exits for installation-written routines and authorized user programs.
- "Chapter 7. System Macro Instructions," contains the system macros used to refer to, validate, and modify system data areas.
- "Chapter 8. Maintaining SYS1.IMAGELIB," describes how to add a UCS or FCB image to the system image library, and how to maintain the UCS image tables.
- "Chapter 9. JES2 Support for the IBM 1403, 3203 Model 5, and 3211 Printers," describes the JES2 support for UCS alias names and the 3211 indexing feature.
- "Chapter 10. CATALOG, SCRATCH, and RENAME Dummy Modules," contains a description of the dummy modules for CATALOG, SCRATCH, and RENAME, and how to replace them.
- "Chapter 11. Specifying Buffer Numbers for DASD Data Sets," contains performance considerations for using the BUFNO keyword of the DCB macro to allocate BSAM buffers.
- "Appendix A. VTOC Access Macros," contains the format and description of the four VTOC access macros: CVAFDIR, CVAFDSM, CVAFSEQ, and CVAFTST.
- "Appendix B. Examples of VTOC Access Macros," contains examples of how to use the VTOC access macros in your programs.
- "Appendix C. Return Codes from VTOC Access Macros," contains the return codes generated by the four VTOC access macros.

- "Appendix D. VTOC Error Messages and Associated Codes," contains the error messages and field codes issued by the Common VTOC Access Facility (CVAF).

- "Appendix E. Example of an Open Exit Module," contains a program listing for IFG0EX0B, an installation-written exit routine that takes control during OPEN for a DCB.

The operating system provides simpler ways (for example, access method services, job control language, utility programs, access method routines) to perform most of the tasks discussed in this book. The information presented here is intended to provide greater flexibility in using the data management capabilities.

PREREQUISITE KNOWLEDGE

In order to use this book efficiently, you should be familiar with the following topics:

- Assembler language

- Standard program linkage conventions

- Catalog management for OS CVOLs

- The utility programs IEHLIST, IEHMOVE, and IEHPROGM

- Data management access methods and macro instructions

REQUIRED PUBLICATIONS

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

- OS/VS-DOS/VSE-VM/370 Assembler Language contains more information on coding in assembler language.

- OS/VS2 Supervisor Services and Macro Instructions contains a description of standard linkage conventions.

- MVS/370 Access Method Services Reference for the Integrated Catalog Facility or MVS/370 Access Method Services Reference for VSAM Catalogs describes how to maintain catalogs.

- MVS/370 Utilities describes how to use IEHLIST to maintain the VTOC, IEHMOVE to maintain OS CVOLs, and IEHPROGM to protect data sets.

- MVS/370 Data Management Services and MVS/370 Data Management Macro Instructions contain information on using access methods and macro instructions to do input and output.

More specific prerequisite reading is listed at the beginning of some chapters, as it relates to the particular topic.

RELATED PUBLICATIONS

Within the text, references are made to the publications listed in the table below.

Short Title	Publication Title	Order Number
Access Method Services Reference	MVS/370 Access Method Services Reference for the Integrated Catalog Facility	GC26-4051

Short Title	Publication Title	Order Number
Access Method Services Reference	MVS/370 Access Method Services Reference for VSAM Catalogs	GC26-4059
Catalog Users Guide	MVS/370 Catalog Users Guide	GC26-4053
Checkpoint/Restart	MVS/370 Checkpoint/Restart	GC26-4054
CVAF Diagnosis Reference	MVS/370 Common VTOC Access Facility Diagnosis Reference	SY26-3933
DADSM and CVAF Diagnosis Guide	MVS/370 DADSM and Common VTOC Access Facility Diagnosis Guide	SY26-3918
DADSM Diagnosis Reference	MVS/370 DADSM Diagnosis Reference	SY26-3919
Data Facility Product: General Information	MVS/370 Data Facility Product: General Information	GC26-4050
Data Management Macro Instructions	MVS/370 Data Management Macro Instructions	GC26-4057
Data Management Services	MVS/370 Data Management Services	GC26-4058
Debugging Handbook	OS/VS2 MVS System Programming Library: Debugging Handbook, Volumes 1-3	GC28-1047 GC28-1048 GC28-1049
Device Support Facilities User's Guide and Reference	Device Support Facilities User's Guide and Reference	GC35-0033
IBM System/370 Principles of Operation	IBM System/370 Principles of Operation	GA22-7000
IBM 2821 Control Unit Component Description	IBM 2821 Control Unit Component Description	GA24-3312
IBM 3203 Printer Component Description and Operator's Guide	IBM 3203 Printer Component Description and Operator's Guide	GA33-1515

Short Title	Publication Title	Order Number
IBM 3211 Printer, 3216 Interchangeable Train Cartridge, and 3811 Printer Control Unit Component Description and Operator's Guide	IBM 3211 Printer, 3216 Interchangeable Train Cartridge, and 3811 Printer Control Unit Component Description and Operator's Guide	GA24-3543
IBM 3800 Printing Subsystem Programmer's Guide	IBM 3800 Printing Subsystem Programmer's Guide	GC26-3846
IBM 4245 Printer Model 1 Component Description and Operator's Guide	IBM 4245 Printer Model 1 Component Description and Operator's Guide	GA33-1541
JCL	OS/VS2 MVS JCL	GC28-0692
Linkage Editor and Loader	MVS/370 Linkage Editor and Loader	GC26-4061
Magnetic Tape Labels and File Structure	MVS/370 Magnetic Tape Labels and File Structure	GC26-4064
Message Library: System Messages	OS/VS Message Library: VS2 System Messages	GC28-1002
Network Job Entry Facility for JES2	OS/VS2 MVS System Programming Library: Network Job Entry Facility for JES2	SC23-0003
OS/VS-DOS/VSE-VM/370 Assembler Language	OS/VS-DOS/VSE-VM/370 Assembler Language	GC33-4010
OS/VS2 I/O Supervisor Logic	OS/VS2 I/O Supervisor Logic	SY26-3823
Open/Close/EOV Logic	MVS/370 Open/Close/EOV Logic	LY26-3924
RACF General Information Manual	Resource Access Control Facility (RACF): General Information Manual	GC28-0722
Supervisor Services and Macro Instructions	OS/VS2 MVS Supervisor Services and Macro Instructions	GC28-0683
System Generation Reference	MVS/370 System Generation Reference	GC26-4063

Short Title	Publication Title	Order Number
System Programming Library: Initialization and Tuning Guide	OS/VS2 MVS System Programming Library: Initialization and Tuning Guide	GC28-0681
System Programming Library: JES2	OS/VS2 MVS System Programming Library: JES2	GC23-0002
System Programming Library: JES3	OS/VS2 MVS System Programming Library: JES3	GC28-0608
System Programming Library: Service Aids	OS/VS2 MVS System Programming Library: Service Aids	GC28-0674
System Programming Library: Supervisor	OS/VS2 MVS System Programming Library: Supervisor	GC28-0628
TSO Command Language Reference	OS/VS2 TSO Command Language Reference	GC28-0646
Utilities	MVS/370 Utilities	GC26-4065
VSAM Reference	MVS/370 VSAM Reference	GC26-4074

NOTATIONAL CONVENTIONS

A uniform system of notation describes the format of data management macro instructions. This notation is not part of the language; it simply provides a basis for describing the structure of the commands.

The command format illustrations in this book use the following conventions:

- Brackets [ ] indicate an optional parameter.
- 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 these items may be selected.
- An ellipsis (...) indicates that multiple entries of the type immediately preceding the ellipsis are allowed.
- Other punctuation (parentheses, commas, spaces, etc.) must be entered as shown. A space is indicated by a blank.
- **BOLDFACE** type indicates the exact characters to be entered, except as described in the bulleted notes above. Such items must be entered exactly as illustrated.
- Lowercase underscored type specifies fields to be supplied by the user.
- **BOLDFACE UNDERSCORED** type indicates a default option. If the parameter is omitted, the underscored value is assumed.

- Parentheses ( ) must enclose subfields if more than one is specified. If only one subfield is specified, you may omit the parentheses.

ADDRESS AND REGISTER CONVENTIONS

The following describes the meaning of each notation used to show how an operand can be coded:

symbol
When this notation is shown, the operand can be any valid assembler-language symbol.

(0)
When this notation is shown, general register 0 can be used as an operand. When used as an operand in a macro instruction, the register must be specified as the decimal digit 0 enclosed in parentheses as shown above.

(1)
When this notation is shown, general register 1 can be used as an operand. When used as an operand in a macro instruction, the register must be specified as the decimal digit 1 enclosed in parentheses as shown above. When register 1 is used, the instruction loaded into the register is not included in the macro expansion.

(2-12)
When this notation is shown, the operand specified can be any of the general registers 2 through 12. All registers as operands must be coded in parentheses; for example, if register 3 is coded, it is coded as (3). When one of the registers 2 through 12 is used, it can be coded as a decimal digit, symbol (equated to a decimal digit), or an expression that results in a value of 2 through 12.

RX-Type Address
When this notation is shown, the operand can be specified as any valid assembler-language RX-type address. The following shows examples of each valid RX-type address:

Name	Operation	Operand
ALPHA1	L	1,39(4,10)
ALPHA2	L	REG1,39(4,TEN)
BETA1	L	2,ZETA(4)
BETA2	L	REG2,ZETA(REG4)
GAMMA1	L	2,ZETA
GAMMA2	L	REG2,ZETA
GAMMA3	L	2,=F'1000'
LAMBDA1	L	3,20(,5)

Both ALPHA instructions specify explicit addresses; REG1 and TEN have been defined as absolute symbols. Both BETA instructions specify implied addresses, and both use index registers. Indexing is omitted from the GAMMA instructions. GAMMA1 and GAMMA2 specify implied addresses. The second operand of GAMMA3 is a literal. LAMBDA1 specifies an explicit address with no indexing.

A-Type Address
When this notation is shown, the operand can be specified as any address that can be written as a valid assembler-language A-type address constant. An A-type address constant can be written as an absolute value, a relocatable symbol, or a relocatable expression. Operands that require an A-type address are inserted into an A-type address constant during the macro expansion process. For more details about A-type address constants, refer to OS/VS-DOS/VSE-VM/370 Assembler Language.

absexp

When this notation is shown, the operand can be an absolute value or expression. An absolute expression can be an absolute term or an arithmetic combination of absolute terms. An absolute term can be a nonrelocatable symbol, a self-defining term, or the length attribute reference. For more details about absolute expressions, refer to OS/VS-DOS/VSE-VM/370 Assembler Language.

relexp

When this notation is shown, the operand can be a relocatable symbol or expression. A relocatable symbol or expression is one whose value changes by n if the program in which it appears is relocated n bytes away from its originally assigned area of storage. For more details about relocatable symbols and expressions, refer to OS/VS-DOS/VSE-VM/370 Assembler Language.

SUMMARY OF AMENDMENTS

RELEASE 1.1, OCTOBER 1983

NEW DEVICE SUPPORT

Information to support the following new device has been added to Figure 34 on page 142:

- 4245 Printer

The chapter, "Maintaining SYS1.IMAGELIB," has been added. This chapter replaces the section formerly titled, "Adding a UCS Image or FCB Image to the System Image Library." The new chapter contains the information previously found in the old chapter, plus a description of the UCS image table, and procedures for updating the image table for the 4245 printer.

SERVICE CHANGES

Chapter 2, formerly titled "Controlling Space on DASD Volumes," has been renamed, "Managing the Volume Table of Contents (VTOC)."

Minor service changes have also been made throughout the manual.

CONTENTS

FIGURES

CHAPTER 1. USING CATALOG MANAGEMENT MACRO INSTRUCTIONS

Using catalog management macro instructions, you can do the following things:

- Retrieve information from an ICF catalog, a VSAM catalog, or an OS CVOL
- Catalog non-VSAM data sets in an ICF catalog, a VSAM catalog, or an OS CVOL
- Uncatalog non-VSAM data sets from an ICF catalog, a VSAM catalog, or an OS CVOL
- Recatalog non-VSAM data sets in an ICF catalog, a VSAM catalog, or an OS CVOL
- Read a block from an OS CVOL
- Build an index in an OS CVOL
- Build a generation index in an OS CVOL
- Delete an index from an OS CVOL
- Assign an alias to a high-level index in an OS CVOL
- Delete an index alias from an OS CVOL
- Connect two OS CVOLs
- Disconnect two OS CVOLs

Before using the information in this chapter, you should be familiar with the following publications:

- OS/VS-DOS/VSE-VM/370 Assembler Language contains information you will need to code programs in the assembler language.
- Access Method Services Reference tells how to use programs that offer some of the same services as OS CVOL management macros plus additional services that the macros cannot provide.
- JCL tells how to catalog and uncatalog non-VSAM data sets using job control language statements.
- Catalog Users Guide tells how to use OS CVOLs.

Specifications for coding the macro instructions are presented with each function to be performed. Accompanying the descriptions are coding examples and programming notes; exception return codes follow the coding examples. In the functional descriptions, offsets into data areas are numbered from zero (the first byte is byte zero).

CATALOG ORDER OF SEARCH

The order in which catalogs are searched when an entry is to be located is:

1. If a specific catalog is specified in a macro, only that catalog is searched. If the entry is not found, a "no entry found" error is returned to the user.

2. Any user catalog(s) specified in the current job step with a STEPCAT DD statement is searched. If more than one catalog is specified for the job step, the catalogs are searched in order of concatenation. If the entry is found, no other catalog is searched.

   If a STEPCAT catalog is specified and the entry is not found, the JOBCAT catalog is not searched. The catalog search continues with step 3 below.

   If no STEPCAT catalog is specified for the job step, and a user catalog is specified for the current job with a JOBCAT DD statement, the JOBCAT catalog(s) is searched. If more than one catalog is specified for the job, the catalogs are searched in order of concatenation. If the entry is found, no other catalog is searched. Otherwise,

3. If the entry is identified with a qualified entryname and its first qualifier is the same as:

   - The name of a user catalog, or

   - The alias of a user catalog, or

   - The alias of an OS CVOL,

   the user catalog or OS CVOL so identified is searched. If the entry is found, no other catalog is searched. Otherwise,

4. The master catalog is searched. If the entry is not found, a "no entry found" error is returned to the user.

RETURN CODE CONSIDERATIONS

The interpretation of catalog management return codes depends on whether the request is initiated using a CAMLST macro or a catalog parameter list (CPL), and whether the request is satisfied in an integrated catalog facility (ICF) catalog, a VSAM catalog, or an OS CVOL.

If CAMLST is used and the request is satisfied in an OS CVOL, register 15 contains the OS CVOL return code and registers 0 and 1 may further describe the return code meaning. If CAMLST is used and the request is satisfied in an ICF or a VSAM catalog, register 15 contains the OS CVOL return code, register 0 the VSAM return code, and register 1 is zero.

If a CPL is used and the request is satisfied in an OS CVOL, register 15 contains the VSAM return code, register 0 is not meaningful, and register 1 is nonzero. If a CPL is used and the request is satisfied in an ICF or a VSAM catalog, register 15 contains the VSAM return code. The return code, reason code, and module identification can also be found in the CPL. These codes are explained in Message Library: System Messages under message IDC3009I.

Note that, regardless of which parameter list is used, if the request is satisfied in an ICF or a VSAM catalog, register 1 is zero, and if the request is satisfied in an OS CVOL, register 1 contains X'08' in the high-order byte and may contain return information in the low-order byte.

RETRIEVING INFORMATION FROM A CATALOG

To read an entry from a catalog, use the LOCATE and CAMLST macro instructions. You may specify the entry you want to read into your work area by using either (1) the fully or partially qualified name of a data set, or (2) the relative block address (TTR) of the block within an OS CVOL containing the entry. If you specify a fully qualified data set name, a list of volumes on which the data set resides will be read into your work area. This volume list always begins with a 2-byte entry that is the number of volumes in the list. If the data set resides on more than 20 volumes and is cataloged in an OS CVOL, the address of a volume control block will follow the volume list entries. (See Figure 5 on page 28 for an explanation of the control block.)

Note: There is a restriction when CAMLST is used to locate a data set that is over 20 volumes in length and on a VSAM catalog. Only the information from the first 20 volumes is returned.

If you specify a partially qualified data set name, the first block in the OS CVOL pointed to by the lowest-level index specified will be read into your work area. This is true if you specify two or more qualifiers, or if you specify the CVOL-RELEXP parameter in the CAMLST macro. Register 15 will contain return code 12. If you specify a single qualifier and do not include the CVOL-RELEXP parameter, the OS CVOL identifier 'SYSCTLG.Vyyyyyy' is read into your work area (the area previously occupied by the data set name). You may then insert 'yyyyyy' as the CVOL-RELEXP parameter in the CAMLST and reissue the LOCATE.

If you specify a relative block address (TTR), the block at that relative address in the CVOL catalog will be read into your work area.

You must add a step when specifying either an unqualified name or the highest level of a partially qualified name to retrieve information from an OS CVOL. You receive, instead, the volume information for the OS CVOL that is found in the master catalog. In addition, the single qualifier name that you specified is replaced by the SYSCTLG.Vyyyyyy name. You may then use that information to specify the OS CVOL volume serial number in CAMLST so that the search starts in the OS CVOL and gives you the information that you expected.

See Figure 1 on page 24 through Figure 8 on page 31 for descriptions of the contents of volume control block and the other catalog data areas.

RETRIEVING INFORMATION BY DATA SET NAME (LOCATE AND CAMLST NAME)

When you specify a data set name, a volume list is built in your work area. A volume list consists of an entry for each volume on which part of the data set resides; it is preceded by a 2-byte field that contains a count of the number of volumes in the list. The count field is followed by a variable number of 12-byte entries. Each 12-byte entry consists of a 4-byte device code, a 6-byte volume serial number, and a 2-byte data set sequence number. As many as 20 of these 12-byte entries can be built in your work area. (Device codes are presented in the UCBTYP data area description of Debugging Handbook.)

If the named data set is stored on only one volume, bytes 252 through 254 of your area may contain the relative track address of the DSCB for that data set; otherwise, these bytes are zero. Byte 255 contains zeros.

If the data set is cataloged in an OS CVOL and resides on more than five volumes, the volume list in your work area is really a volume control block (VCB) that has been read into your work area. In a VCB, the count field contains the number of volume entries in this VCB and any following VCBs. Thus a count of 41 indicates two following VCBs with counts of 21 and one, respectively. The relative track address (TTR) of the next VCB is in bytes 252 through 254 of your work area. The last VCB for a data set has binary zeros in bytes 252 through 254.

The macro format is:

[<u>symbol</u>] <u>listname</u>	LOCATE CAMLST	<u>listname-addrx</u> NAME ,<u>dsname-relexp</u> ,[<u>cvol-relexp</u>] ,<u>area-relexp</u>

<u>listname-addrx</u>
points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

NAME
this operand must be coded as shown to retrieve information from a catalog by name.

<u>dsname-relexp</u>
specifies the virtual storage location of a fully qualified data set name. The area that contains the name must be 44 bytes long. The name may be defined by a C-type Define Constant (DC) instruction.

<u>cvol-relexp</u>
specifies the virtual storage location of the 6-byte volume serial number of the OS CVOL to which this catalog request is directed. For a discussion of the effect of specifying or omitting this operand, see "Catalog Order of Search" on page 2.

<u>area-relexp</u>
specifies the virtual storage location of your 265-byte work area, which you must define. The work area must begin on a doubleword boundary.

Example: In the following example, the catalog entry containing a list of the volumes on which data set A.B resides is read into virtual storage.

```
          LOCATE    INDAB                   READ CATALOG ENTRY
*                                           FOR DATA SET A.B
*                                           INTO VIRTUAL STORAGE
*                                           AREA NAMED LOCAREA.
*                                           LOCAREA MAY ALSO
*                                           CONTAIN A 3-BYTE
*                                           TTR AND THE 6-BYTE
*                                           OS CVOL SERIAL NUMBER
         Check Return Codes

INDAB     CAMLST    NAME,AB,,LOCAREA
AB        DC        CL44'A.B'
LOCAREA   DS        0D
          DS        265C
```

The LOCATE macro instruction points to the CAMLST macro instruction. NAME, the first operand of CAMLST, specifies that the system is to search for a catalog entry using the name of a data set. AB, the second operand, specifies the virtual storage location of a 44-byte area into which you have placed the fully qualified name of a data set. LOCAREA, the fourth operand, specifies a 265-byte area you have reserved in virtual storage.

After execution of these macro instructions, the 265-byte area contains a volume list or a volume control block for the data set A.B.

Control will be returned to your program at the next executable instruction after the LOCATE macro instruction. If the block has been successfully read from the catalog, register 15 will contain zeros. Otherwise, register 15 will contain one of the following return codes.

Code **Meaning**

4(04) Either the required catalog does not exist or it cannot be opened or there is a closed chain of OS CVOL pointers.

8(08) One of the following happened:

- The entry was not found. Register 0 contains the number of valid index levels if in an OS CVOL. Register 0 contains the catalog return code if in an ICF or a VSAM catalog.
- The user is not authorized to perform this operation. Register 0 contains hexadecimal 38.
- A generation data group (GDG) alias was found. Register 0 contains the number of valid index levels. The alias name was replaced by the true name.

12(0C) One of the following happened:

- An index or generation data group base entry was found when the list of qualified names was exhausted. Register 0 contains the number of valid index levels. The work area contains the first block of the specified index.
- An alias entry was found. The alias name was replaced in the user parameter list by the true name.
- An invalid low-level GDG name was found.

16(10) A data set exists at other than the lowest index level specified. Register 0 contains the number of the index level where the data set was encountered.

20(14) A syntax error exists in the name.

24(18) One of the following happened:

- Permanent I/O error occurred. Register 0 contains the VSAM or ICF return code or 0 if in an OS CVOL.
- Nonzero ESTAE return code.
- Error in parameter list.

28(1C) Relative track address supplied to LOCATE routine is outside of the SYSCTLG data set extents.

32(20) Reserved.

Note: See Message Library: VS2 System Messages, Section IDC3009I, for documentation of ICF catalog and VSAM catalog return codes.

RETRIEVING INFORMATION BY GENERATION DATA SET NAME (LOCATE AND CAMLST NAME)

You specify the name of a generation data set by using the fully qualified generation index name and the relative generation number of the data set. The value of a relative generation number reflects the position of a data set in a generation data group. The following values can be used:

- Zero—specifies the latest data set (highest generation number) cataloged in a generation data group.
- Negative number—specifies a data set cataloged before the latest data set.
- Positive number—specifies a data set not yet cataloged in the generation data group.

When you use zero or a negative number as the relative generation number, a volume list (or a volume control block) is placed in your work area, and the relative generation number is replaced by the absolute generation name.

When you use a positive number as the relative generation number, an absolute generation name is created and replaces the relative generation number. Zeros are read into the first 256 bytes of your work area, because there are no entries in the catalog.

The format is:

[symbol] listname	LOCATE CAMLST	list-addrx NAME ,dsname-relexp ,[cvol-relexp] ,area-relexp

list-addrx
: points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

NAME
: this operand must be coded as shown in order to read a block from the catalog by generation data set name.

dsname-relexp
: specifies the virtual storage location of the name of the generation index and the relative generation number. The area that contains these must be 44 bytes long. The name may be defined by a C-type define constant (DC) instruction.

cvol-relexp
: specifies the virtual storage location of the 6-byte volume serial number of the OS CVOL to which this catalog request is directed. For a discussion of the effect of specifying or omitting this operand, see "Catalog Order of Search" on page 2.

area-relexp
: specifies the virtual storage location of your 265-byte work area, which you must define. The work area must begin on a doubleword boundary. The first 256 bytes of the work area will contain a volume list that is built from the catalog. If the data set resides on one volume, bytes 252 through 254 may contain the relative track address of the DSCB. This address is relative to the beginning of the volume.

Example: In the following example, the list of volumes that contain generation data set A.PAY(-3) is read into virtual storage.

```
          LOCATE    INDGX                        READ CATALOG ENTRY
*                                                FOR DATA SET A.PAY(-3)
*                                                INTO YOUR STORAGE
*                                                AREA NAMED LOCAREA
         Check Return Codes
INDGX     CAMLST    NAME,APAY,,LOCAREA
APAY      DC        CL44'A.PAY(-3)'
LOCAREA   DS        0D
          DS        265C
```

The LOCATE macro instruction points to the CAMLST macro instruction. NAME, the first operand of CAMLST, specifies that the system is to search the catalog for a catalog entry by using the name of a data set. APAY, the second operand, specifies the virtual storage location of a 44-byte area into which you have placed the name of the generation index and the relative generation number of a data set in the generation data group. LOCAREA, the fourth operand, specifies a 265-byte area you have reserved to receive the catalog information.

After execution of these macro instructions, the system will have replaced the relative generation number that you specified in your 44-byte area with the data set's absolute generation name. Control will be returned to your program at the next executable instruction after the LOCATE macro instruction. If the entry has been located and read successfully, register 15 will contain zeros. Otherwise, register 15 will contain a return code. For a description of the contents of the work area or the meaning of the exception return codes, see "Retrieving Information by Data Set Name (LOCATE and CAMLST NAME)" on page 3.

RETRIEVING INFORMATION BY ALIAS (LOCATE AND CAMLST NAME)

For each of the preceding functions, you can specify an alias as the name of a data set. Each function is performed exactly as previously described, with one exception: The alias name specified is replaced by the true name.

Note: Aliases are not allowed for generation data sets cataloged in OS CVOLs.

The format is:

[symbol] listname	**LOCATE** **CAMLST**	list-addrx **NAME** ,dsname-relexp ,[cvol-relexp] ,area-relexp

list-addrx
: points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

NAME
: this operand must be coded as shown to retrieve information from a catalog.

dsname-relexp

specifies the virtual storage location of a fully qualified data set name, the first or only name of which is the alias. The area that contains the name must be 44 bytes long. The name may be defined by a C-type define constant (DC) instruction.

cvol-relexp

specifies the virtual storage location of the 6-byte volume serial number of the OS CVOL to which this catalog request is directed. For a discussion of the effect of specifying or omitting this operand, see "Catalog Order of Search" on page 2.

area-relexp

specifies the virtual storage location of your 265-byte work area, which you must define. The work area must begin on a doubleword boundary. The first 256 bytes of the work area will contain a volume list that is read from a catalog. If the data set resides on one volume, bytes 252 through 254 may contain the relative track address of the DSCB. This address is relative to the beginning of the volume.

Example: In the following example, the catalog entry containing a list of the volumes on which data set A.B.C resides is read into virtual storage (data set A.B.C, however, is addressed by an alias name, X.B.C).

```
          LOCATE    INDAB                    READ CATALOG ENTRY
*                                            FOR DATA SET X.B.C
*                                            INTO VIRTUAL STORAGE
*                                            AREA NAMED LOCAREA.
         Check Return Codes

INDAB     CAMLST    NAME,ABC,,LOCAREA
ABC       DC        CL44'X.B.C'
LOCAREA   DS        0D
          DS        265C
```

The LOCATE macro instruction points to the CAMLST macro instruction. NAME, the first operand of CAMLST, specifies that the system is to search the catalog for an entry using the name of a data set. ABC, the second operand, specifies the virtual storage location of a 44-byte area into which you have placed the fully qualified name of a data set (in this case, data set A.B.C is addressed by its alias X.B.C). LOCAREA, the fourth operand, specifies a 265-byte area you have reserved in virtual storage.

For information on return codes and the contents of your work area after execution, see "Retrieving Information by Data Set Name (LOCATE and CAMLST NAME)" on page 3.

READING A BLOCK BY RELATIVE BLOCK ADDRESS (LOCATE AND CAMLST BLOCK)

You can read any block in an OS CVOL by specifying, in the form TTR, the identification of the block and its location relative to the beginning of the catalog. TT is the number of tracks from the beginning of the catalog; R is the record number of the desired block on the track.

The format is:

[*symbol*] *listname*	LOCATE CAMLST	*list-addrx* BLOCK ,*ttr-relexp* ,*cvol-relexp* ,*area-relexp*

list-addrx
points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

BLOCK
you must code this operand as shown.

ttr-relexp
specifies the virtual storage location of a 3-byte relative block address (TTR). This address indicates the position relative to the beginning of the catalog data set, of the track containing the block (TT), and the block identification (R) on that track.

cvol-relexp
specifies the virtual storage location of a 6-byte volume serial number for the volume to be processed.

area-relexp
specifies the virtual storage location of your 265-byte work area, which you must define. The work area must begin on a doubleword boundary. The first 256 bytes of the work area will contain the block that is read from the catalog, and the last 6 bytes will contain the serial number of the volume on which the block was found. If the data set resides on one volume, bytes 252 through 254 will contain the relative track address of the DSCB.

Example: In the following example, the block at the location indicated by TTR is read into virtual storage.

```
          LOCATE   BLK

         Check Return Codes

BLK       CAMLST   BLOCK,TTR,VOLSER,LOCAREA
*                                     READ A BLOCK INTO
*                                     VIRTUAL STORAGE AREA
TTR       DC       H'5'               RELATIVE TRACK 5
          DC       X'03'              BLOCK 3 ON TRACK
VOLSER    DC       C'111111'          VOLUME SERIAL OF OS CVOL
LOCAREA   DS       0D                 NAMED LOCAREA
          DS       265C               LOCAREA ALSO CONTAINS
*                                     6-BYTE SERIAL NO.
```

The LOCATE macro instruction points to the CAMLST macro instruction. BLOCK, the first operand of CAMLST, specifies that the system is to search the catalog for the block indicated by TTR, the second operand. VOLSER, the third operand, specifies the virtual storage location of a 6-byte volume serial number for the volume to be processed. LOCAREA, the fourth operand, specifies a 265-byte area you have reserved in virtual storage.

After execution of these macro instructions, the 265-byte area contains: the 256-byte block and the 6-byte serial number of the volume on which the block was found (in bytes 259 through 264).

Control will be returned to your program at the next executable instruction following the LOCATE macro instruction. If the index block at the address you specified has been successfully located and read into your work area, register 15 will contain zeros. Otherwise, register 15 will contain one of the exception return codes described under "Retrieving Information by Data Set Name (LOCATE and CAMLST NAME)" on page 3.

BUILDING AND DELETING INDEXES

You handle OS CVOL indexes—build them, delete them, and so forth—by using combinations of the INDEX and CAMLST macro instructions.

BUILDING AN INDEX (INDEX AND CAMLST BLDX)

To build a new OS CVOL index structure and add it to the catalog, you may create each level of the index separately. (You can also create index levels while you are cataloging a data set onto those index levels. To create each level of the index, use the INDEX and CAMLST macro instructions.)

These two macro instructions can also be used to add index levels to existing index structures.

The format is:

[symbol]	INDEX	list-addrx
listname	CAMLST	BLDX ,namerelexp [,cvol-relexp]

list-addrx
points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

BLDX
this operand must be coded as shown.

namerelexp
specifies the virtual storage location of the fully qualified name of a data set or index level. The name cannot exceed 44 characters. If the name is less than 44 characters, it must be followed by at least one blank. The name may be defined by a C-type define constant (DC) instruction.

cvol-relexp
specifies the virtual storage location of a 6-byte volume serial number of the OS CVOL to which this catalog request is directed. For a discussion of the effect of specifying or omitting this operand, see "Catalog Order of Search" on page 2.

Example: In the following example, index structure A.B.C is built on the OS CVOL whose serial number is 000045.

Each INDEX macro instruction points to an associated CAMLST macro instruction. BLDX, the first operand of CAMLST, specifies that an index level be built. The second operand specifies the virtual storage location of the area into which you have placed the fully qualified name of an index level. The third operand specifies the virtual storage location of the area into which you have placed the 6-byte serial number of the volume on which the index level is to be built.

```
           INDEX    INDEXA          BUILD INDEX A
          Check Return Codes
           INDEX    INDEXB          BUILD INDEX STRUCTURE
*                                   A.B
          Check Return Codes
           INDEX    INDEXC          BUILD INDEX STRUCTURE
*                                   A.B.C
          Check Return Codes
INDEXA     CAMLST   BLDX,ALEVEL,VOLNUM
INDEXB     CAMLST   BLDX,BLEVEL,VOLNUM
INDEXC     CAMLST   BLDX,CLEVEL,VOLNUM
VOLNUM     DC       CL6'000045'     VOLUME SERIAL NUMBER
ALEVEL     DC       CL2'A'          INDEX STRUCTURE NAMES
BLEVEL     DC       CL4'A.B'        FOLLOWED BY A BLANK
CLEVEL     DC       CL6'A.B.C'      WHICH DELIMITS FIELDS
```

Control will be returned to your program at the next executable instruction following the INDEX macro instruction. If the index has been built successfully, register 15 will contain zeros. Otherwise, register 15 will contain one of the following exception return codes:

Code **Meaning**

4(04) The OS CVOL does not exist or cannot be opened.

8(08) One of the following happened:

- The existing catalog structure is inconsistent with the operation requested. If the error was detected while processing in an OS CVOL, register 0 has the number of valid index levels and register 1 has the return code that would have resulted if a LOCATE macro had been issued on the same entry name. If the error was detected during the master catalog search process, register 0 contains the catalog return code and register 1 contains zero.

- The user is not authorized to perform the operation. Register 0 contains 56 (decimal); register 1 contains 0.

12(0C) An attempt was made to build an index or generation index that has an alias or has indexes or data sets cataloged under it. The index is unchanged.

16(10) The qualified name specified when building an index or generation index implies an index structure that does not exist; the high-level index, specified when connecting control volumes, does not exist.

20(14) Space is not available on the specified OS CVOL.

24(18) Not used with the INDEX macro instruction.

28(1C) A permanent I/O error was found when processing the catalog, or a nonzero return code from ESTAE was encountered.

BUILDING A GENERATION INDEX (INDEX AND CAMLST BLDG)

You build a generation index in an OS CVOL by using the INDEX and CAMLST macro instructions. All higher levels of the index must exist. If the higher levels of the index are not in the catalog, you must build them. How to build an index has been explained previously.

The format is:

[symbol]	INDEX	list-addrx
listname	CAMLST	BLDG ,namerelexp ,[cvol-relexp] ,,[DELETE] ,[EMPTY] ,number-absexp

list-addrx
: points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

BLDG
: this operand must be coded as shown.

namerelexp
: specifies the virtual storage location of the fully qualified name of a data set or index level. The name cannot exceed 44 characters. If the name is less than 44 characters, it must be followed by at least one blank. The name may be defined by a C-type define constant (DC) instruction.

cvol-relexp
: specifies the virtual storage location of a 6-byte volume serial number of the OS CVOL to which this catalog request is directed. For a discussion of the effect of specifying or omitting this operand, see "Catalog Order of Search" on page 2.

DELETE
: specifies that all data sets on direct access volumes that are removed from a generation data group are to be deleted, that is, the space allocated to the data set(s) is to be made available for reallocation. A SCRATCH macro instruction will be issued by the catalog management routines to delete the data set, which will be deleted from the volume if there are no conditions preventing deletion (for example, expiration date not passed, password not verified, volume not mounted, permanent I/O error encountered while trying to delete the data set).

EMPTY
: specifies that references to all data sets in a generation data group cataloged in the generation index are to be removed from the index when the number of entries specified is exceeded.

number-absexp
: specifies the number of data sets to be included in a generation data group. This number must be specified, and cannot exceed 255.

Example: In this example, generation index D is built on the OS CVOL, serial number 000045. The higher-level indexes A.B.C already exist. When the number of generation data sets in the generation index D exceeds four, the oldest data set is uncataloged. When the DELETE operand has been specified and the data set has been successfully uncataloged, the catalog management routines issue a SCRATCH macro (see "Chapter 2. Managing the Volume Table of Contents (VTOC)" on page 33) to delete the data set. If there are no conditions preventing the data set from being deleted (for example, the expiration date was not passed, the password could not be verified, or a permanent I/O error was encountered when trying to delete the data set), the data set will be deleted.

```
           INDEX    GENINDX        BUILD GENERATION INDEX
          Check Return Codes

GENINDX    CAMLST   BLDG,DLEVEL,VOLNUM,,DELETE,,4
DLEVEL     DC       CL8'A.B.C.D '   ONE BLANK, DELIMITER
VOLNUM     DC       CL6'000045'
```

The INDEX macro instruction points to the CAMLST macro instruction. BLDG, the first operand of CAMLST, specifies that a generation index is to be built. DLEVEL, the second operand, specifies the virtual storage location of an area into which you have placed the fully qualified name of a generation index. VOLNUM, the third operand, specifies the virtual storage location of the area into which you have placed the 6-byte serial number of the volume on which the generation index is to be built. DELETE, the fifth operand, specifies that all data sets dropped from the generation data group are to be deleted. The final operand, 4, specifies the number of data sets that are to be maintained in the generation data group. Control will be returned to your program at the next executable instruction following the INDEX macro instruction. If the generation index was built successfully, register 15 contains zeros. Otherwise, register 15 will contain one of the exception return codes described under "Building an Index (INDEX and CAMLST BLDX)" on page 10.

DELETING AN INDEX (INDEX AND CAMLST DLTX)

You can delete any number of index levels from an existing OS CVOL index structure. Each level of the index is deleted separately. Generation indexes are also removed this way. (You can also delete index levels automatically when you uncatalog a data set.) You delete each level of the index by using the INDEX and CAMLST macro instructions.

If an index level either has an alias, or has other index levels or data sets cataloged under it, it cannot be deleted.

The format is:

[,symbol] listname	**INDEX** **CAMLST**	list-addrx **DLTX** ,namerelexp [,cvol-relexp]

list-addrx
: points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

DLTX
: this operand must be coded as shown.

namerelexp
: specifies the virtual storage location of the fully qualified name of a data set or index level. The name cannot exceed 44 characters. If the name is less than 44 characters, it must be followed by at least one blank. The name may be defined by a C-type define constant (DC) instruction.

cvol-relexp
specifies the virtual storage location of a 6-byte volume serial number of the OS CVOL to which this catalog request is directed. For a discussion of the effect of specifying or omitting this operand, see "Catalog Order of Search" on page 2.

Example: In the following example, index level C is deleted from index structure A.B.C.

```
            INDEX    DELETE              DELETE INDEX LEVEL
*                                        C FROM INDEX STRUCTURE
*                                        A.B.C

           Check Return Codes

DELETE      CAMLST   DLTX,LEVELC
LEVELC      DC       CL6'A.B.C'          ONE BLANK FOR
*                                        DELIMITER
```

The INDEX macro instruction points to the CAMLST macro instruction. DLTX, the first operand of CAMLST, specifies that an index level be deleted. LEVELC, the second operand, specifies the virtual storage location of the area into which you have placed the fully qualified name of the index structure whose lowest level is to be deleted. Control will be returned to your program at the next executable instruction following the INDEX macro instruction. If the index level(s) was successfully deleted, register 15 contains zeros. Otherwise, register 15 contains one of the exception return codes described under "Building an Index (INDEX and CAMLST BLDX)" on page 10.

ASSIGNING AN ALIAS FOR AN INDEX (INDEX AND CAMLST BLDA)

For OS CVOLs you assign an alias to an index level by using the INDEX and CAMLST macro instructions. An alias can be assigned only to a high level index; for example, index A of index structure A.B.C can have an alias, but index B cannot. Assigning an alias to a high level index effectively provides aliases for all data sets cataloged under that index. An alias cannot be assigned to a generation index.

The format is:

[symbol] listname	**INDEX** **CAMLST**	list-addrx **BLDA** ,index namerelexp ,[cvol-relexp] ,alias namerelexp

list-addrx
points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

BLDA
this operand must be coded as shown.

index namerelexp
specifies the virtual storage location of the name of a high-level index. The area that contains the name must be 8 bytes long. The name may be defined by a C-type define constant (DC) instruction.

cvol-relexp
specifies the virtual storage location of a 6-byte volume serial number of the OS CVOL catalog to which this catalog

request is directed. For a discussion of the effect of specifying or omitting this operand, see "Catalog Order of Search" on page 2.

alias namerelexp
: specifies the virtual storage location of the name that is to be used as an alias for a high-level index. The area that contains the name must be 8 bytes long. The name may be defined by a C-type define constant (DC) instruction.

Example: In the following example, high-level index A is assigned an alias of X.

```
           INDEX      ALIAS                 BUILD AN ALIAS FOR
*                                           A HIGH LEVEL INDEX

          Check Return Codes

ALIAS      CAMLST     BLDA,DSNAME,,DSALIAS
DSNAME     DC         CL8'A'                MUST BE 8-BYTE FIELDS
DSALIAS    DC         CL8'X'
```

The INDEX macro instruction points to the CAMLST macro instruction. BLDA, the first operand of CAMLST, specifies that an alias be built. DSNAME, the second operand, specifies the virtual storage location of an 8-byte area into which you have placed the name of the high-level index to be assigned an alias. DSALIAS, the fourth operand, specifies the virtual storage location of an 8-byte area into which you have placed the alias to be assigned.

Control will be returned to your program at the next executable instruction following the INDEX macro instruction. If the alias has been successfully assigned, register 15 will contain zeros. Otherwise, register 15 will contain one of the exception return codes described under "Building an Index (INDEX and CAMLST BLDX)" on page 10.

DELETING AN ALIAS FOR AN INDEX (INDEX AND CAMLST DLTA)

For OS CVOLs you can delete an alias previously assigned to a high-level index by using the INDEX and CAMLST macro instructions.

The format is:

[symbol] listname	**INDEX** **CAMLST**	list-addrx **DLTA** ,alias namerelexp [,cvol-relexp]

list-addrx
: points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

DLTA
: this operand must be coded as shown.

alias namerelexp
: specifies the virtual storage location of the name that is used as an alias for a high-level index. The area that contains the name must be 8 bytes long. The name may be defined by a C-type define constant (DC) instruction.

cvol-relexp
specifies the virtual storage location of a 6-byte volume serial number of the OS CVOL catalog to which this catalog request is directed. For a discussion of the effect of specifying or omitting this operand, see "Catalog Order of Search" on page 2.

Example: In the following example, alias X, previously assigned as an alias for index level A, is deleted.

```
            INDEX     DELALIAS              DELETE AN ALIAS FOR
*                                           A HIGH LEVEL INDEX

           Check Return Codes

DELALIAS    CAMLST    DLTA,ALIAS
ALIAS       DC        CL8'X'                MUST BE 8-BYTE FIELD
```

The INDEX macro instruction points to the CAMLST macro instruction. DLTA, the first operand of CAMLST, specifies that an alias be deleted. ALIAS, the second operand, specifies the virtual storage location of the 8-byte area into which you have placed the alias to be deleted.

CONNECTING AND DISCONNECTING OS CVOLS

You connect and disconnect OS CVOLs by using combinations of the INDEX and CAMLST macro instructions.

CONNECTING OS CVOLS (INDEX AND CAMLST LNKX)

You connect two OS CVOLs by using the INDEX and CAMLST macro instructions.

You must supply the serial number of the volume to be connected and the high-level index name that will be used to associate the two volumes. If the index name is an alias of an OS CVOL pointer entry in the master catalog, then the serial number of the "from" volume may be omitted. Otherwise, you must supply the serial numbers of both volumes and the name of a high-level index associated with the volume to be connected.

The result of connecting OS CVOLs is that the volume serial number of the OS CVOL connected and the name of a high-level index are entered into the volume index of the volume to which it was connected. This entry is called a control-volume pointer.

The format is:

[symbol] listname	**INDEX** **CAMLST**	list-addrx **LNKX** ,index namerelexp ,[cvol-relexp] ,new cvol-relexp

list-addrx
points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

LNKX
this operand must be coded as shown.

index namerelexp
specifies the virtual storage location of the name of a high-level index. The area that contains the name must be 8 bytes long. The name may be defined by a C-type define constant (DC) instruction.

cvol-relexp
specifies the virtual storage location of a 6-byte volume serial number of the OS CVOL catalog to which this catalog request is directed. For a discussion of the effect of specifying or omitting this operand, see "Catalog Order of Search" on page 2.

new cvol-relexp
specifies the virtual storage location of the 4-byte device code and 6-byte volume serial number of the control volume that is to be connected to another OS CVOL.

Example: In the following example, the OS CVOL whose serial number is 001555 is connected to the OS CVOL numbered 000155. The name of the high-level index is HIGHINDX.

```
           INDEX    CONNECT            CONNECT TWO OS CVOLS
*                                      WHOSE SERIAL NUMBERS ARE
*                                      000155 and 001555.
*                                      3330 DISK DEVICE CODE
          Check Return Codes

CONNECT    CAMLST   LNKX,INDXNAME,OLDCVOL,NEWCVOL
*
INDXNAME   DC       CL8'HIGHINDX'
OLDCVOL    DC       CL6'000155'
NEWCVOL    DC       X'30C0200D'
           DC       CL6'001555'
```

The INDEX macro instruction points to the CAMLST macro instruction. LNKX, the first operand of CAMLST, specifies that control volumes be connected. INDXNAME, the second operand, specifies the virtual storage location of the 8-byte area into which you have placed the name of the high-level index of the volume to be connected. OLDCVOL, the third operand, specifies the virtual storage location of a 6-byte area into which you have placed the serial number of the volume to which you are connecting.

NEWCVOL, the fourth operand, specifies the virtual storage location of a 10-byte area into which you have placed the 4-byte hexadecimal device code of the volume to be connected followed by the 6-byte area to contain the volume serial number of the volume to be connected.

Control will be returned to your program at the next executable instruction following the INDEX macro instruction. If the OS CVOLs have been successfully connected, register 15 will contain zeros. Otherwise, register 15 will contain one of the exception return codes described under "Building an Index (INDEX and CAMLST BLDX)" on page 10.

DISCONNECTING OS CVOLS (INDEX AND CAMLST DRPX)

You disconnect two OS CVOLs by using the INDEX and CAMLST macro instructions.

The result of disconnecting OS CVOLs is that the OS CVOL pointer is removed from the volume index of the volume from which you are disconnecting.

The format is:

[symbol]	INDEX	list-addrx
listname	CAMLST	DRPX ,index namerelexp [,cvol-relexp]

list-addrx
: points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

DRPX
: this operand must be coded as shown.

index namerelexp
: specifies the virtual storage location of the name of a high-level index. The area that contains the name must be 8 bytes long. The name may be defined by a C-type define constant (DC) instruction.

cvol-relexp
: specifies the virtual storage location of a 6-byte volume serial number of the OS CVOL catalog to which this catalog request is directed. For a discussion of the effect of specifying or omitting this operand, see "Catalog Order of Search" on page 2.

Example: In the following example, the OS CVOL that contains the high-level index HIGHINDX is disconnected from the OS CVOL pointed to by the entry 'HIGHINDX' in the master catalog.

```
          INDEX    DISCNECT                 DISCONNECT TWO
*                                           OS CVOLS
         Check Return Codes

DISCNECT  CAMLST   DRPX,INDXNAME
INDXNAME  DC       CL8'HIGHINDX'            MUST BE 8-BYTE FIELD
```

The INDEX macro instruction points to the CAMLST macro instruction. DRPX, the first operand of CAMLST, specifies that OS CVOLs be disconnected. INDXNAME, the second operand, specifies the virtual storage location of the 8-byte area into which you have placed the name of the high-level index of the OS CVOL to be disconnected.

Control will be returned to your program at the next executable instruction following the INDEX macro instruction. If the OS CVOLs were successfully disconnected, register 15 will contain zeros. Otherwise, register 15 will contain one of the exception return codes described under "Building an Index (INDEX and CAMLST BLDX)" on page 10.

WORKING WITH NON-VSAM DATA SET CATALOG ENTRIES

You can catalog, uncatalog, and recatalog non-VSAM data sets in OS CVOLs, ICF catalogs, and VSAM catalogs by using combinations of the CATALOG and CAMLST macro instructions. CATALOG macro instructions are used to point to CAMLST macro instructions; CAMLST macro instructions are used to specify cataloging options.

To catalog non-VSAM data sets in ICF or VSAM catalogs, the search algorithm is the same as that given in the section "Order of Catalog Selection for DEFINE" in the Access Method Services Reference. To uncatalog or recatalog non-VSAM data sets in ICF or VSAM catalogs, the search algorithm is the same as that given in the section "Order of Catalog Search for DELETE" in *Access Method Services Reference*.

CATALOGING A NON-VSAM DATA SET (CATALOG AND CAMLST CAT)

The format of the CATALOG and CAMLST macros is:

[symbol]	CATALOG	list-addrx
listname	CAMLST	CAT[BX] ,name-relexp ,[cvol-relexp] ,vol list-relexp [,DSCBTTR=dscb ttr-relexp]

list-addrx
: points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

CAT[BX]
: this operand must be coded as shown. Either CAT or CATBX may be coded; but, in either case, missing indexes within an OS CVOL are always automatically created.

name-relexp
: specifies the virtual storage location of the fully qualified name of a data set. The name cannot exceed 44 characters. If the name is less than 44 characters, it must be followed by at least one blank. The name may be defined by a C-type define constant (DC) instruction.

cvol-relexp
: specifies the virtual storage location of the 6-byte volume serial number of the OS CVOL catalog to which this catalog request is directed. For a discussion of the effect of specifying or omitting this operand, see "Building an Index (INDEX and CAMLST BLDX)" on page 10.

vol list-relexp
: specifies the virtual storage location of an area that contains a volume list. The list must begin on a halfword boundary and consist of an entry for each volume on which the data set is stored. The first two bytes of the list indicate the number of entries in the volume list; the number cannot be zero. Each 12-byte volume list entry consists of a 4-byte device code, a 6-byte volume serial number, and a 2-byte data set sequence number. The sequence number is always zero for direct access volumes. (Device codes are presented in *Debugging Handbook*.)

DSCBTTR=dscb ttr-relexp
: specifies the virtual storage location of the 3-byte relative track address (TTR) of the format-1 data set control block (DSCB) for a data set that resides on only one volume. The address is relative to the beginning of the volume.

Programming Considerations for Multiple-Step Jobs

When you are executing multiple-step jobs, it is preferable to catalog or uncatalog data sets using JCL, instead of using IEHPROGM or a user program. Since ALLOCATION/UNALLOCATION monitors data sets during job execution, and it is not aware of the functions performed by the user programs, conflicting functions can be performed or GDG orientation can be lost.

UNALLOCATION recatalogs existing cataloged data sets at job termination. This action occurs because the data set is opened sometime during the job and the DSCB TTR was not found in the catalog entry. Therefore, if you are using the CAMLST macro to uncatalog and then catalog data sets with new volume information, be sure to include the DSCB TTR.

Example: In the following example, the non-VSAM data set named A.B.C is cataloged. The data set is stored on two volumes.

```
         CATALOG ADDABC              CATALOG DATA SET A.B.C.
        Check Return Codes
ADDABC   CAMLST     CAT,DSNAME,,VOLUMES
DSNAME   DC         CL6'A.B.C'       ONE BLANK FOR DELIMITER
VOLUMES  DC         H'2'             DATA SET ON TWO VOLUMES
         DC         X'30C0200D'      3330 DISK DEVICE CODE
         DC         CL6'000014'      VOLUME SERIAL NUMBER
         DC         H'0'             DATA SET SEQUENCE NUMBER
         DC         X'30C0200D'      3330 DISK DEVICE CODE
         DC         CL6'000015'      VOLUME SERIAL NUMBER
         DC         H'0'             SEQUENCE NUMBER
```

The CATALOG macro instruction points to the CAMLST macro instruction. CAT, the first operand of CAMLST, specifies that a data set is to be cataloged. DSNAME, the second operand, specifies the virtual storage location of the area in which the data set name A.B.C was placed. VOLUMES, the fourth operand, specifies the virtual storage location of the volume list that was built.

Control will be returned at the instruction following the CATALOG macro instruction. If A.B.C was successfully cataloged, register 15 will contain zeros. Otherwise, register 15 will contain one of the following return codes:

Code **Meaning**

4(04) Either the required catalog does not exist, it is not open, or the "do not allocate" bit is on.

8(08) One of the following happened:

- The existing catalog structure is inconsistent with the operation requested. If the error was detected while processing in an OS CVOL, register 0 has the number of valid index levels and register 1 has the return code that would have resulted if a LOCATE macro had been issued for the same entry name. If the error was detected in an ICF or a VSAM catalog, register 0 contains the catalog return code and register 1 contains zero.
- The user is not authorized to perform the operation. Register 0 contains decimal 56 (X'36') and register 1 contains zero.

12(0C) Not used with the CATALOG macro instruction.

16(10) The index structure necessary to catalog the data set does not exist.

20(14) There is insufficient space on the catalog data set.

24(18) An attempt was made to catalog an improperly named generation data set, or the generation index is full and the named data set is older than any currently in the index.

28(1C) One of the following happened:

- A permanent I/O or unrecoverable error was encountered.
- An error was found in a parameter list.
- An I/O error occurred in an OS CVOL.
- There was a nonzero return code from ESTAE.

UNCATALOGING A NON-VSAM DATA SET (CATALOG AND CAMLST UNCAT)

When the UNCAT or UCATDX operand of the CAMLST macro instruction is used, a data set reference and unneeded indexes, with the exception of the highest-level index, are removed.

The format of the CATALOG and CAMLST macros is:

[symbol] listname	**CATALOG** **CAMLST**	list-addrx **UNCAT** or **UCATDX** ,name-relexp [,cvol-relexp]

list-addrx
points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

UNCAT or **UCATDX**
this operand must be coded as shown. Either UNCAT or UCATDX may be coded but in either case unneeded indexes, with the exception of the highest-level index, are always removed along with the data set reference. .

name-relexp
specifies the virtual storage location of the fully qualified name of a data set or index level. The name cannot exceed 44 characters. If the name is less than 44 characters, it must be followed by at least one blank. The name may be defined by a C-type define constant (DC) instruction.

cvol-relexp
specifies the virtual storage location of the 6-byte volume serial number of the OS CVOL catalog to which this catalog request is directed. For a discussion of the effect of specifying or omitting this operand, see "Catalog Order of Search" on page 2.

In the following example, the catalog entry for data set A.B.C is removed from a catalog. In an OS CVOL, index B is removed unless it contains references to other data sets. Index A remains because it is the highest-level index.

```
         CATALOG REMOVE                REMOVE REFERENCES TO
*                                      DATA SET A.B.C FROM
*                                      CATALOG
        Check Return Codes

REMOVE   CAMLST     UNCAT,DSNAME
DSNAME   DC         CL6'A.B.C'         ONE BLANK FOR DELIMITER
```

The CATALOG macro instruction points to the CAMLST macro instruction. UNCAT, the first operand of CAMLST, specifies that references to a data set be removed from the catalog. DSNAME, the second operand, specifies the virtual storage location of an area into which you have placed the fully qualified name of the data set whose references are to be removed.

Control will be returned to your program at the instruction following the CATALOG macro instruction. If your data set has been successfully uncataloged, register 15 will contain zeros. Otherwise, register 15 will contain one of the return codes described under "Cataloging a Non-VSAM Data Set (CATALOG and CAMLST CAT)" on page 19.

RECATALOGING A NON-VSAM DATA SET (CATALOG AND CAMLST RECAT)

You can recatalog a cataloged non-VSAM data set by using the CATALOG and CAMLST macro instructions. Recataloging is usually necessary if a data set is extended to a new volume.

As in the original cataloging procedure, you must build a complete volume list in virtual storage. This volume list consists of an entry for each volume on which the data set resides. The first 2 bytes of the list indicate the number of entries in the list; the number may not be zero. Each 12-byte volume pointer consists of a 4-byte device code, a 6-byte volume serial number, and a 2-byte data set sequence number. The sequence number is always zero for direct access volumes. (Device codes are presented in Debugging Handbook.)

The format of the CATALOG and CAMLST macros is:

[symbol] listname	**CATALOG** **CAMLST**	list-addrx **RECAT** ,name-relexp ,[cvol-relexp] ,vol list-relexp [,**DSCBTTR=**dscb ttr-relexp]

list-addrx
points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

RECAT
this operand must be coded as shown.

name-relexp
specifies the virtual storage location of the fully qualified name of a data set. The name cannot exceed 44 characters. If the name is less then 44 characters, it must be followed by at least one blank. The name may be defined by a C-type define constant (DC) instruction.

cvol-relexp
specifies the virtual storage location of the 6-byte volume serial number of the OS CVOL catalog to which this catalog request is directed. For a discussion of the effect of specifying or omitting this operand, see "Catalog Order of Search" on page 2.

vol list-relexp
specifies the virtual storage location of an area that contains a volume list. The area must begin on a half-word boundary.

DSCBTTR=dscb ttr-relexp
specifies the virtual storage location of the 3-byte relative track address (TTR) of the identifier (format-1) DSCB for a data set that resides on only one volume. The address is relative to the beginning of the volume.

Example: In the following example, the two-volume data set named A.B.C is recataloged to add a third volume. An entry is added to the volume list, which previously contained only two entries.

```
          CATALOG RECATLG                  RECATALOG DATA SET
*                                          A.B.C ADDING A NEW
*                                          VOLUME

         Check Return Codes

RECATLG   CAMLST    RECAT,DSNAME,,VOLUMES
DSNAME    DC        CL6'A.B.C '            FOR DELIMITER ONE BLANK
VOLUMES   DC        H'3'                   THREE VOLUMES
          DC        X'30C0200D'            3330 DISK DEVICE CODE
          DC        CL6'000014'            VOLUME SERIAL NUMBER
          DC        H'0'                   SEQUENCE NUMBER
          DC        X'30C0200D'            3330 DISK DEVICE CODE
          DC        CL6'000015'            VOLUME SERIAL NUMBER
          DC        H'0'                   SEQUENCE NUMBER
          DC        X'30C0200D'            3330 DISK DEVICE CODE
          DC        CL6'000016'            VOLUME SERIAL NUMBER
          DC        H'0'                   SEQUENCE NUMBER
```

The CATALOG macro instruction points to the CAMLST macro instruction. RECAT, the first operand of CAMLST, specifies that a data set be recataloged. DSNAME, the second operand, specifies the virtual storage location of an area into which you have placed the fully qualified name of the data set to be recataloged. VOLUMES, the fourth operand, specifies the virtual storage location of the volume list you have built.

Control will be returned to your program at the instruction following the CATALOG macro instruction. If the data set has been successfully recataloged, register 15 will contain zeros. Otherwise, register 15 will contain one of the return codes described under "Cataloging a Non-VSAM Data Set (CATALOG and CAMLST CAT)" on page 19.

OS CVOL ENTRY FORMATS

This section describes the format and contents of each of the entries that may appear in the OS CVOL.

OS CVOL VOLUME INDEX CONTROL ENTRY

Field 1	Field 2	Field 3
X'0000000000000001' Name	TTR of last block in volume index	X'05' Count

0 8 11 12

Field 4	Field 5	Field 6	Field 7	Field 8
TTR of last block in SYSCTLG data set	X'00'	TTR of first unused block in SYSCTLG data set	X'00'	Unused bytes

12 15 16 19

<————————Total Length: 22 Bytes————————>

Field 1: Name (8 bytes)—contains only a hexadecimal 1 to ensure that this entry is the first entry in the first block of the index.

Field 2: Last-block address (3 bytes)—contains the relative track address (TTR) of the last block in the volume index.

Field 3: Halfword count (1 byte)—contains a hexadecimal 5 to indicate that 5 halfwords follow.

Field 4: Catalog upper limit (3 bytes)—contains the relative track address (TTR) of the last block in the catalog data set.

Field 5: Zero field (1 byte)—contains binary zeros.

Field 6: First-available-block address (3 bytes)—contains the relative track address (TTR) of the unused block in the catalog that is closest to the beginning of the catalog data set.

Field 7: Zero field (1 byte)—contains binary zeros.

Field 8: Unused (2 bytes)

Figure 1. The OS CVOL Volume Index Control Entry

OS CVOL INDEX CONTROL ENTRY

Field 1	Field 2	Field 3	Field 4	Field 5	Field 6
X'0000000000000001' Name	TTR of last block in this index	X'03' Count	TTR of first block in this index	Alias count	Unused bytes
0	8	11	12	15	16

<——————————Total Length: 18 Bytes——————————>

This index control entry is similar to a volume index control entry, but it only contains information about the index, which it begins. It is 18 bytes long and contains six fields.

Field 1: Name (8 bytes)—contains only a hexadecimal 1 to ensure that this entry, because it has the lowest binary name value, is the first entry in the first block of the index.

Field 2: Last block address (3 bytes)—contains the relative track address (TTR) of the last block assigned to this index.

Field 3: Halfword count (1 byte)—contains a hexadecimal 3 to indicate that 3 halfwords follow.

Field 4: Index lower limit (3 bytes)—contains the relative track address (TTR) of the block in which this entry appears.

Field 5: Number of aliases (1 byte)—contains the binary count of the number of aliases assigned to the high-level index. If the index is not a high-level index, this field is zero.

Field 6: Unused (2 bytes)

Figure 2. The OS CVOL Index Control Entry

OS CVOL INDEX LINK ENTRY AND INDEX POINTER ENTRY

Index Link Entry

Field 1	Field 2	Field 3
X'FFFFFFFFFFFFFFFF' Name	TTR of next block in index (or zero if no next block)	X'00' Count

0 8 11

<——————Total Length: 12 Bytes——————>

Index Pointer Entry

Field 1	Field 2	Field 3
Index name (padded to right with blanks if necessary)	TTR of index	X'00' Count

0 8 11

<——————Total Length: 12 Bytes——————>

The index link and index pointer entries are similar. An index link entry is used to chain several blocks of an index together, and an index pointer entry is used to chain an index to the next lower-level index. An index link entry is always the last entry in any index block. These blocks contain three fields and are 12 bytes long.

Field 1: Name (8 bytes)—contains the name of the index to which this entry points. If the entry is an index link entry, the name field contains X'FF FF FF FF FF FF FF FF'.

Field 2: Address (3 bytes)—contains either the relative block address (TTR) of the first block of the next level index if it is an index pointer entry, or the relative block address (TTR) of the next block of the same level index if it is an index link entry.

Field 3: Halfword count (1 byte)—contains 1 byte of binary zeros to indicate that the entry ends here.

Figure 3. The OS CVOL Index Link and Index Pointer Entries

OS CVOL DATA SET POINTER ENTRY

```
┌──────────────────────────┬──────────────────┬──────────────────┬───────────────┐
│ Field 1                  │ Field 2          │ Field 3          │ Field 4       │
│                          │                  │                  │               │
│ Lowest-level name of     │ DSCB             │ Count            │ Volume        │
│ data set or complemented │ TTR or           │                  │ count         │
│ generation number        │ zeros            │                  │               │
│ (if part of GDG)         │                  │                  │               │
└──────────────────────────┴──────────────────┴──────────────────┴───────────────┘
0                          8                  11                 12              14

┌───────────────────┬───────────────────────────────────┬───────────────────────┐
│ Field 5           │ Field 6                           │ Field 7               │
│                   │                                   │                       │
│ Device            │ Serial number                     │ Data set sequence     │
│ Code              │ of volume on                      │ number (zero for      │
│                   │ which data                        │ direct access)        │
│                   │ set resides                       │                       │
└───────────────────┴───────────────────────────────────┴───────────────────────┘
14                  18                                  24
|                                                                               |
└───────────────────────────────────────v───────────────────────────────────────┘
                               Repeated for each volume

<──────────/ /──────────Total Length: 26 to 74 Bytes──────────────────────────>
```

The data set pointer entry can appear in any index. It contains the simple name of a data set and from one to five 12-byte fields, each of which identifies a volume on which the named data set resides. If the data set resides on more than five volumes, a volume control block pointer entry is substituted for the data set pointer entry. A volume control block pointer entry points to a volume control block or chain of volume control blocks that point to the volumes that contain the data set.

The data set pointer entry varies in length. The length is determined by the formula 14 + 12m, where m is the number of volumes containing the data set. The variable m can be from one to five. The data set pointer entry can appear in any index, and it contains seven fields.

Field 1: Name (8 bytes)—contains the simple name of the data set whose volumes are identified in field 5. If part of a GDG, these names have the format GxxxxV00, where xxxx is the complement of the GDG number.

Field 2: DSCB TTR (3 bytes)—contains the track address (TTR) of the data set control block if the data set resides on one volume. If the data set resides on more than one volume, this field contains binary zeros.

Field 3: Halfword count (1 byte)—contains the binary count of the number of halfwords that follow. The number is found by the formula 6m + 1, where m is the number of volumes on which the data set resides. The variable m can be from one to five.

Field 4: Volume count (2 bytes)—contains the binary count of the number of volumes identified in field 5 of this entry.

Field 5: Device code (4 bytes)—contains the device code of the device on which the volume with the volume serial number in field 6 can be mounted.

Field 6: Volume serial number (6 bytes)—contains the volume serial number of one of the volumes of the data set.

Field 7: Data set sequence number (2 bytes)—contains the sequence number of the data set on a magnetic tape volume. It is zero for any other device class.

Figure 4. The OS CVOL Data Set Pointer Entry

OS CVOL VOLUME CONTROL BLOCK POINTER ENTRY

Field 1	Field 2	Field 3	Field 4
Lowest level of data set name	TTR of volume control block	X'01' Count	X'0000' Dummy data entry
0	8	11	12

<——————————Total Length: 14 Bytes——————————>

The volume control block pointer entry is used instead of a data set pointer entry when the data set resides on more than five volumes. This entry points to a volume control block, which, in turn, describes the data set. The entry is 14 bytes long.

Field 1: Name (8 bytes)—contains the last name of the qualified name of the data set identified by this entry.

Field 2: Address (3 bytes)—contains the relative block address (TTR) of the volume control block identifying the volumes containing the data set named in field 1.

Field 3: Halfword count (1 byte)—contains a hexadecimal 1 to indicate that 1 halfword follows.

Field 4: Zero field (2 bytes)—contains hexadecimal zeros.

Figure 5. The OS CVOL Volume Control Block Pointer Entry

VOLUME CONTROL BLOCK

Field 1	Field 2	Field 3	Field 4
Count	Device Code	Serial number of volume n	Data set sequence number for the volume described in field 5. Zero for direct access

0 m m+4 m+10

Repeated once for each volume; maximum of 20

Field 5	Field 6	Field 7
Ten bytes of zeros	TTR of next volume control block, or zero if none	X'00'

242 252 255

<——Total Length: 256 Bytes——/ /——>

A volume control block contains the description of all the volumes of a data set that resides on more than five volumes. If a data set resides on less than six volumes, a volume control block is not built and the volumes are described in a data set pointer entry. One volume control block can describe as many as 20 volumes. Volume control blocks may be chained together to catalog a data set residing on more than 20 volumes.

The volume control block is always 256 bytes long, regardless of the number of volumes described.

Field 1: Volume count (2 bytes)—the first volume control block contains the binary count of the total number of volumes on which the data set resides. The value of this field is reduced by 20 for each subsequent volume control block. If, for example, the data set resides on 61 volumes, there will be four volume control blocks for the data set. The volume count field of each will contain 61, 41, 21, or 1, respectively.

Fields 2, 3, 4: Volume identification (12 to 240 bytes)—contains from 1 to 20 entries, each of which identifies a volume on which the data set resides. Each entry contains a 4-byte device code, a 6-byte volume serial number, and a 2-byte data set sequence number. The data set sequence number is zero for data sets on direct access volumes.

Field 5: Zero field (10 bytes)—contains binary zeros.

Field 6: Chain address (3 bytes)—contains the relative block address (TTR) of the next volume control block, if additional blocks are needed to describe the data set. If this is the last volume control block for the data set, this field will be set to binary zeros.

Field 7: Zero field (1 byte)—contains binary zeros.

Figure 6. The OS CVOL Volume Control Block

OS CVOL POINTER ENTRY

Field 1	Field 2	Field 3
Name of index on other OS CVOL	Dummy Pointer field: zeros	X'05' Count

0 8 11 12

Field 4	Field 5
Device code of OS CVOL	Serial number of OS CVOL

12 16

Total Length: 22 Bytes

The OS CVOL pointer entry is used to indicate that a particular index resides on a volume other than the system residence volume.
OS CVOL pointer entries can exist only in the volume index.
They are 22 bytes long.

Field 1: Name (8 bytes)—contains a high-level index name that appears in the volume index of the OS CVOL identified in fields 4 and 5.

Field 2: Address (3 bytes)—contains zeros, because this entry references no other entry in the catalog.

Field 3: Halfword count (1 byte)—contains the hexadecimal value 5 to indicate that 5 halfwords follow.

Field 4: OS CVOL device code (4 bytes)—contains the device code of the specified control volume.

Field 5: OS CVOL serial number (6 bytes)—contains the volume serial number of the OS CVOL which has an entry in its volume index of the same name as this entry.

Figure 7. The OS CVOL Pointer Entry

OS CVOL POINTER ENTRY (OLD)

Until Release 17 of OS MFT/MVT, the OS CVOL pointer entry was the same as the present OS CVOL pointer, except that there was no field 4 (device code); the OS CVOL pointer entry was 18 bytes long. After Release 17, the OS CVOL pointer entry is 22 bytes long. This is mentioned because some OS CVOLs may still contain entries in the old format and the catalog management routines may still check for them.

OS CVOL GENERATION INDEX POINTER ENTRY

Field 1	Field 2	Field 3	Field 4	Field 5	Field 6
Name	TTR	Count	Flags	Maximum Count	Current Count
0	8	11	12	13	14

<——————————Total Length: 16 Bytes——————————>

A generation index pointer entry is the entry that identifies a generation data group (GDG). It represents the next to the lowest-level of a group of generation data set names. It is created by using the BLDG macro.

Field 1: Name (8 bytes)—this name represents the GDG level that is next to the lowest level of GDG data set names.

Field 2: Address (3 bytes)—contains the relative track address (TTR) of the first block of the level containing the lowest-level GDG names. These names have the format GxxxxV00, where xxxx is a complement of the GDG number.

Field 3: Count (1 byte)—X'02' identifies this entry and indicates the number of halfwords that follow this field.

Field 4: Flags (1 byte)—indicates the options specified by the creator of the GDG.

X'02'=DELETE option.

X'01'=EMPTY option.

Field 5: Maximum Count (1 byte)—a binary number that specifies the maximum number of generations allowed in the generation index at one time.

Field 6: Current Count (2 bytes)—the binary count of the number of generations currently cataloged in the generation data group (GDG).

Figure 8. The OS CVOL Generation Index Pointer Entry

OS CVOL ALIAS NAME

Field 1	Field 2	Field 3	Field 4
		X'04'	
Alias Name	TTR pointer	Count	True Name

0 8 11 12

<——————————Total Length: 20 Bytes——————————>

An alias entry defines an alternative name for the high-level qualifier of a data set name.

Field 1: Name (8 bytes)—contains the alias of the high-level index whose relative track address is found at field 2.

Field 2: Address (3 bytes)—contains the relative track address (TTR) of the first block of the index named in field 4.

Field 3: Count (1 byte)—identifies this entry and contains the binary count of the number of halfwords that follow. The number is X'04'.

Field 4: True name (8 bytes)—contains the name of the index whose alias appears in field 1.

Figure 9. The OS CVOL Alias Name

CHAPTER 2. MANAGING THE VOLUME TABLE OF CONTENTS (VTOC)

The direct access device storage management (DADSM) routines control allocation of space on direct access volumes through the volume table of contents (VTOC) of that volume, and through the VTOC index if one exists. This chapter gives an overview of the VTOC and the VTOC index, and discusses how to use system macros to access the VTOC and VTOC index.

THE VTOC

The VTOC is a data set on a direct access volume that describes the contents of that volume. It resides in a single extent (that is, it is a continuous data set), anywhere on the volume after cylinder 0, track 0. Its address is located in the VOLVTOC field of the standard volume label (see Figure 10).

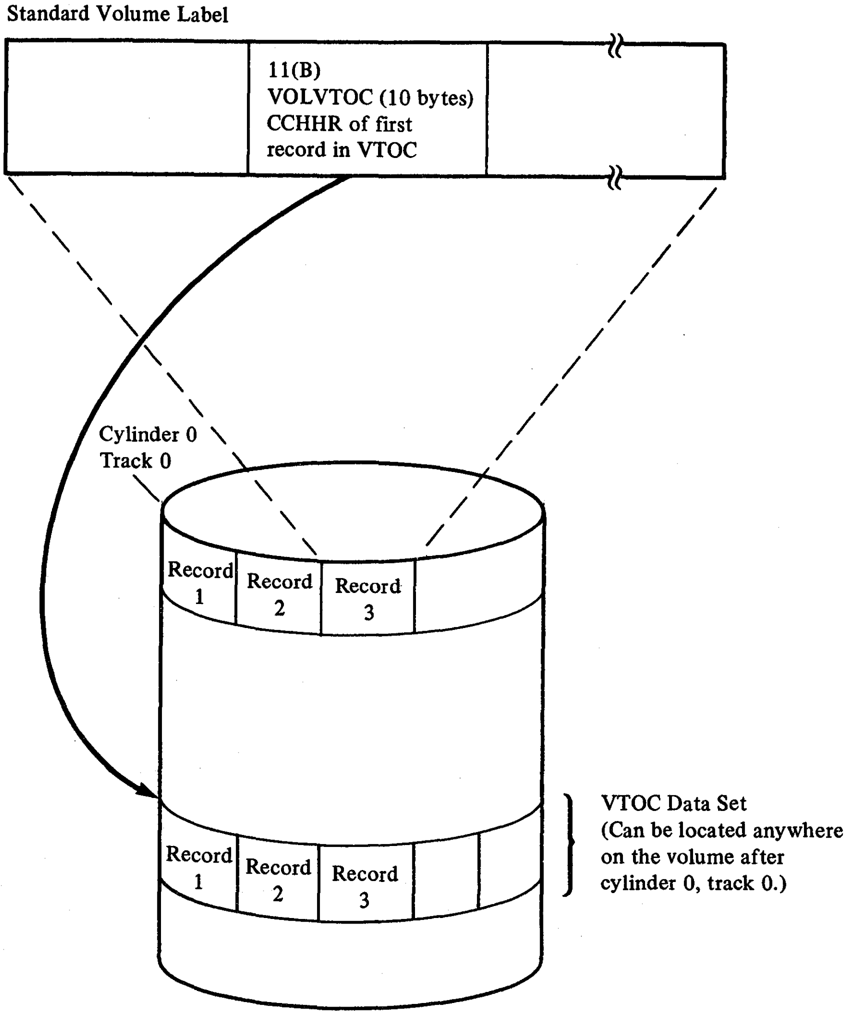


Figure 10. Locating the Volume Table of Contents (VTOC)

The VTOC is composed of 140-byte data set control blocks (DSCBs) that correspond either to a data set or VSAM data space currently residing on the volume, or to contiguous, unassigned tracks on the volume. DSCBs for data sets or data spaces describe their characteristics and the characteristics of the tracks on which they reside. DSCBs for contiguous, unassigned tracks indicate their location.

DATA SET CONTROL BLOCK (DSCB) FORMAT TYPES

The VTOC has seven different kinds of DSCBs. This section lists the different kinds of DSCBs, what they are used for, how many exist on a volume, and how they are found.

The first record in every VTOC is the VTOC (format-4) DSCB that describes (1) the device that the volume resides on, (2) the attributes of the volume itself, and (3) the size and contents of the VTOC data set itself.

The format-4 DSCB is followed by a free-space (format-5) DSCB, which for a nonindexed VTOC lists the extents on the volume that have not been allocated to a data set or VSAM data space. Each format-5 DSCB contains 26 extents. If there are more than 26 available extents on the volume, another format-5 DSCB will be built for every 26 extents. The format-5 DSCBs are chained using the last field of each format-5 DSCB. An indexed VTOC does not use format-5 DSCBs for describing free space; however, one empty format-5 DSCB is provided to allow a basis for converting back to a nonindexed VTOC.

The third and subsequent DSCBs in the VTOC do not necessarily occupy contiguous space, nor do they have any prescribed sequence.

A data set or VSAM data space is defined by one or more DSCBs in the VTOC of each volume on which it resides. The number of DSCBs needed to define a data set or VSAM data space is determined by (1) the organization of the data set (ISAM data sets need a format-2 DSCB to describe the index) and (2) the number of extents the data set or VSAM data space occupies (a format-3 DSCB is needed to describe the 4th through the 16th extents; additional format-3 DSCBs may be required to describe the extents for a VSAM data set cataloged in an ICF catalog). Figure 11 on page 37 shows the general makeup of a VTOC and the DSCBs needed to define two types of data sets (ISAM and non-ISAM).

Data set A (in Figure 11 on page 37) is an ISAM data set; three DSCBs, a format-1, format-2, and format-3, are required. Data sets B, C, and D could be sequential, partitioned, or direct data sets or VSAM data spaces. Data set B has more than three extents and therefore requires both a format-1 and a format-3 DSCB.

Data sets C and D have three or fewer extents and need only a format-1 DSCB. The format-6 DSCB, pointed to by the format-4 DSCB, is used to keep track of the extents allocated in order to be shared by two or more data sets (split-cylinder data sets). For example, if data sets C and D share an extent made up of one or more cylinders, this extent would be described in the format-6 DSCB. Note that split-cylinder data sets cannot be allocated, but existing split-cylinder data sets can still be processed.

Format-0 DSCB

NAME: Free VTOC Record

FUNCTION: The unused records in the VTOC, which contains 140 bytes of binary zeros. To delete a DSCB from the VTOC, a format-0 DSCB is written over it.

HOW MANY: One for every unused 140-byte record on the VTOC. The DS4DSREC field of the format-4 DSCB is a count of the number of format-0 DSCBs on the VTOC. This field is not maintained for an indexed VTOC.

HOW FOUND: Search on key equal to X'00' (sometimes X'00000000') for a nonindexed VTOC; for an indexed VTOC, the VTOC map of DSCBs is used to find a format-0 DSCB.

Format-1 DSCB

NAME: Identifier

FUNCTION: Describes the first three extents of a data set or VSAM data space.

HOW MANY: One for every data set or data space on the volume, except the VTOC.

HOW FOUND: Search on key equal to the data set name. For an indexed VTOC, a CCHHR pointer for each data set name is in the VTOC index.

Format-2 DSCB

NAME: Index

FUNCTION: Describes the indexes of an ISAM data set.

HOW MANY: One for every ISAM data set (for a multivolume ISAM data set, a format-2 DSCB exists only on the first volume).

HOW FOUND: Chained from a format-1 DSCB that represents the data set.

Format-3 DSCB

NAME: Extension

FUNCTION: Describes the 4th through 16th extents of a data set or VSAM data space. Data sets and VSAM data spaces are restricted to 16 extents per volume. VSAM data sets cataloged in an ICF catalog may be extended to a maximum of 123 extents, in which case there may be up to ten format-3 DSCBs.

HOW MANY: One for each data set or VSAM data space on the volume that has more than three extents. There may be up to ten for a VSAM data set cataloged in an ICF catalog.

HOW FOUND: Chained from a format-2 or a format-1 DSCB that represents the data set or VSAM data space. In the case of a VSAM data set cataloged in an ICF catalog, the chain may be from a preceding format-3 DSCB.

Format-4 DSCB

NAME: VTOC

FUNCTION: Describes the extent and contents of the VTOC and provides volume and device characteristics. If the VTOC is indexed, certain fields of this DSCB are not maintained by DADSM. See "Structure of an Indexed VTOC."

HOW MANY: One on each volume.

HOW FOUND: VOLVTOC field of the standard volume label contains its address. It is always the first record in the VTOC.

Format-5 DSCB

NAME: Free Space

FUNCTION: On a nonindexed VTOC, describes the space on a volume that has not been allocated to a data set or to a VSAM data space (available space). For an indexed VTOC, format-5 is zero, and the volume pack space map describes the available space.

HOW MANY: One for every 26 non-contiguous extents of available space on the volume for a nonindexed VTOC; for an indexed VTOC, there is only one.

HOW FOUND: The first format-5 DSCB on the volume is always the second DSCB of the VTOC. If there is more than one format-5 DSCB, it will be chained from the previous format-5 DSCB via the DS5PTRDS field of each format-5 DSCB.

Format-6 DSCB

NAME: Shared Extent

FUNCTION: Describes the extents shared by two or more data sets (split-cylinder extents).

HOW MANY: One for every 26 split-cylinder extents on the VTOC.

HOW FOUND: The address of the first format-6 DSCB is contained in the DS4F6PTR field of the format-4 DSCB. If there is more than one format-6 DSCB on the volume, it will be chained from the previous format-6 DSCB via the DS6PTRDS field of the format-6 DSCB.

ALLOCATING AND RELEASING SPACE

The DADSM allocate and extend routines assign tracks and cylinders on direct access volumes for new data sets and VSAM data spaces. The DADSM extend routine obtains additional space for a data set or VSAM data space that has already exceeded its original, primary allocation. The DADSM scratch and partial release routines are used to release space that is no longer needed on a direct access volume.

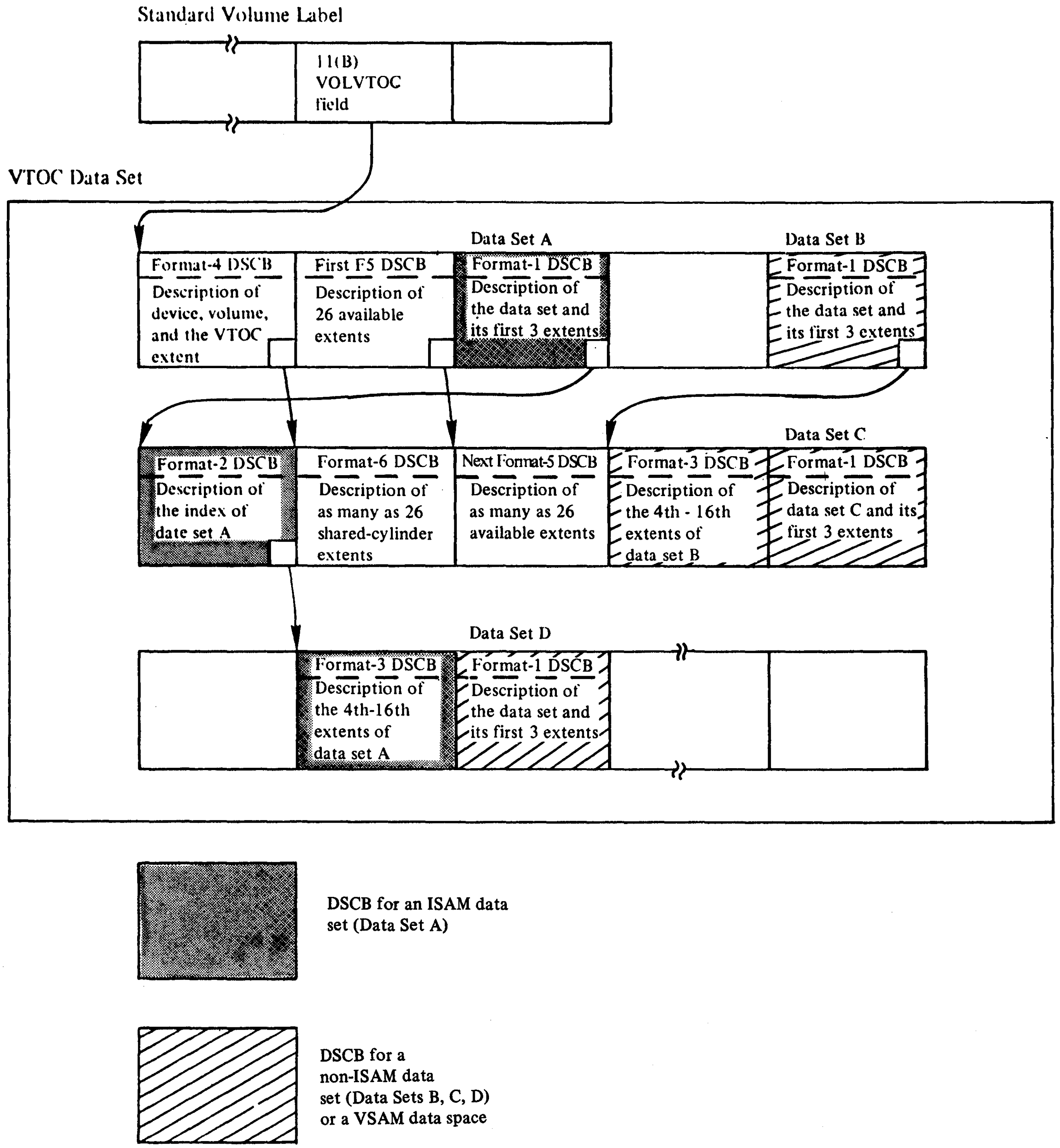
The DADSM routines allocate and release space by adding, deleting, and modifying the DSCBs. When space is needed on a volume, the allocate routines search the appropriate DSCBs for enough contiguous, available tracks to satisfy the request. If there are not enough contiguous tracks, the request is filled using as many as five noncontiguous groups of free tracks. The appropriate DSCBs are modified to reflect the assignment of the tracks.

When space is released, the scratch routines free the DSCBs of the deleted data set or data space. For a nonindexed VTOC, to indicate that the tracks containing the affected data set or data space can be reallocated, a free space (format-5) DSCB is built, or modified if existent. For an indexed VTOC, the index is updated.

THE VTOC INDEX

The VTOC index is a physical-sequential data set, residing on the same volume as the VTOC. It contains an index of data set names of Format-1 DSCBs in the VTOC, as well as free space information. The index is searched instead of the hardware keys.

The VTOC index is optional. It can be built over an existing VTOC and inactivated so that the VTOC is processed without using the index.


Standard Volume Label
11(B)
VOLVTOC
field
VTOC Data Set
Data Set A
Data Set B
Format-4 DSCB
Description of device, volume, and the VTOC extent
First F5 DSCB
Description of 26 available extents
Format-1 DSCB
Description of the data set and its first 3 extents
Format-1 DSCB
Description of the data set and its first 3 extents
Data Set C
Format-2 DSCB
Description of the index of date set A
Format-6 DSCB
Description of as many as 26 shared-cylinder extents
Next Format-5 DSCB
Description of as many as 26 available extents
Format-3 DSCB
Description of the 4th - 16th extents of data set B
Format-1 DSCB
Description of data set C and its first 3 extents
Data Set D
Format-3 DSCB
Description of the 4th-16th extents of data set A
Format-1 DSCB
Description of the data set and its first 3 extents
DSCB for an ISAM data set (Data Set A)
DSCB for a non-ISAM data set (Data Sets B, C, D) or a VSAM data space

Note: Empty boxes in the VTOC data set represent free VTOC Records (Format-0 DSCBs)

Figure 11. Contents of VTOC—DSCBs Describing Data Sets

Each VTOC index is formatted by Device Support Facilities with physical blocks 2048 bytes in length. These physical blocks are the VTOC index records (VIRs), the basic structural units of the index. The kind of information they contain depends on the part of the index they belong to.

Several different kinds of records, each built from one or more VIRs, are in a VTOC index:

- The VTOC index entry record (VIER), which is used to access format-1 DSCBs and the format-4 DSCB

- The VTOC pack space map (VPSM), which shows what space has been allocated on a disk pack

- The VTOC index map (VIXM), which shows which VIRs have been allocated in the VTOC index

- The VTOC map of DSCBs (VMDS), which shows which DSCBs have been allocated in the VTOC

AN EXAMPLE OF A VTOC AND ITS INDEX

A format-1 DSCB in the VTOC contains the name and extent information of the VTOC index. The name of the index must be 'SYS1.VTOCIX.xxxxxxxx', where 'xxxxxxxx' can be anything valid in a data set name and is generally the serial number of the volume containing the VTOC and its index. The name must be unique within the system to avoid ENQ contention. The relationship of a VTOC to its index is shown in Figure 12. Each of the components of the index is discussed separately in the following sections.

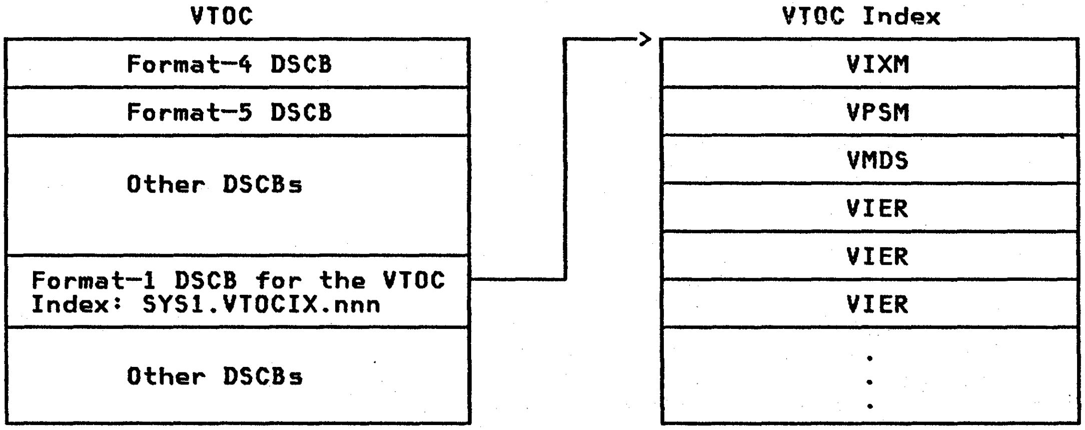


Figure 12. Relationship of a VTOC to Its Index

THE VTOC INDEX ENTRY RECORD (VIER)

VIERs have these characteristics:

- A VIER uses one VIR and contains variable-length index entries. The number of VIERs in an index is variable, depending on the number of data sets on the volume.

- VIERs in a VTOC index may be on one or many levels. All index entries in a VIER are at the same index level. VIERs have a hierarchic relationship. Index entries in higher-level VIERs point to lower-level VIERs. Index entries in level-one VIERs (those at the lowest level) point to format-1 DSCBs for data sets on the volume.

- A higher-level VIER is created when the fourth lower-level VIER is created. When that new higher-level VIER is filled with pointers to lower-level VIERs, a new VIER at the same level is created. Again, when the fourth VIER at the same level is created, a VIER at a still higher level is created, adding another level to the index.

Contents of VIER Fields

Each VIER contains a header and sections (see Figure 13). The VIER header contains:

- A field identifying the VTOC index record as a VIER.
- The relative byte address (RBA) of the VIER.
- A pointer to a VIER at the same level (hence, a "horizontal" pointer). The VIER pointed to contains index entries whose keys are greater than any key in the pointing VIER.
- The level number (LVL) of this VIER.
- The number (SECNO) of sections (a VIER contains eight sections).
- The length (SECL) of the sections (each section is 246 bytes in length).
- The offsets to the first-used and the last-used sections.
- The 44-byte high key of the VIER.

Each section contains:

- An offset to the last entry in the section (or zero if the section is empty)
- Index entries

0(00)	EBCDIC Characters "VIER"			Index Header
4(04)	RBA of This VIER			
8(08)	Horizontal Pointer			
12(0C)	Old Horizontal Pointer			
16(10)	LVL	FLG1	Reserved	
20(14)	PTRL	SECNO	SECL	
24(18)	Offset to First-Used Section			
28(1C)	Offset to Last-Used Section			
32(20)	Highest Key in This VIER			
76(4C)	Section 1			8 Sections Containing Index Entries
	. . .			
	Section 8			

Figure 13. Format of the VTOC Index Entry Record (VIER)

Format of a VTOC Index Entry

The format of an index entry is:

FLG	KEYL	Unused	Record Pointer	Key

Name	Offset	Bytes	Description
VXEFLG	00(00)	1	Flag byte
VXEKEYL	01(01)	1	Length of the VXEKEY field
VXEFC	02(02)	1	Unused
VXERPTR	03(03)	4 or 5	Record pointer

VXEKEY	07(07)
	or
	08(08)
		1 to 44	Name of a data set, if a level—one VIER; if not, the high key in the header of a lower—level VIER

Each index entry contains:

- A flag byte.
- A keylength field (which contains a value of 1 to 44, depending on the length of the data set name).
- A record pointer (VXERPTR) that is one of the following:
  - In level-one VIERs, the 5-byte CCHHR of the format-1 or format-4 DSCB that represents the data set whose name is the key in the entry
  - In other VIERs, the 4-byte RBA of the lower-level VIER whose high key is the key in the entry
- A key which for level 1 VIERS is the data set name, and for level 2 or higher VIERs is the high key of a lower-level VIER. Trailing blanks are suppressed in the VTOC index entry.

When a VIER Is Created

The first level-one VIER is created when the VTOC index is created. Subsequent VIERs are created when a data set name is to be added to the VTOC index but the VIER to which it should be added is full. A new VIER is created in the following manner:

- A new VIER is allocated.
- Half of the sections from a full VIER (those containing the highest keys) are moved into the new VIER, leaving each VIER half empty.
- The new index entry is added to one of the two VIERs, depending on its key.

A Tree of Linked VIERs

Figure 14 on page 41 shows how VIERS are related to each other. Note that the VIERs (which are simplified here—only the high key is shown in the header) form a type of "tree structure."

How a Format-1 DSCB Is Found

In the search for the format-1 DSCB for a particular data set, one path along the tree structure is followed.

As seen in Figure 13 on page 39, a field in the header of a VIER contains the highest key of any index entry in that VIER.

Beginning with this field in the first high-level VIER, the following search logic is used: Is the key of the data set (the

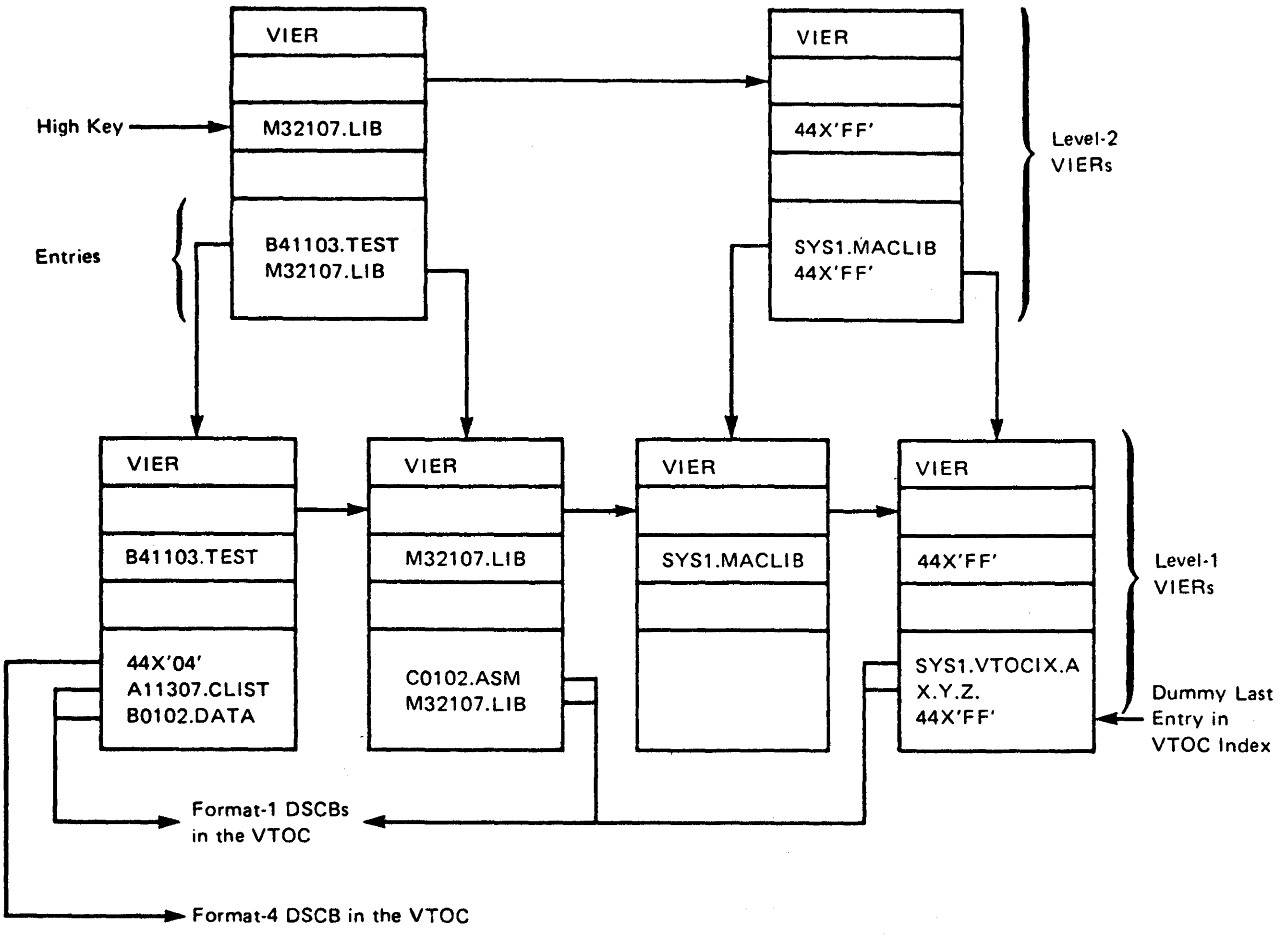


Figure 14. Structure of Linked VIERs

data set name) lower than or equal to the VIER's high key? If neither, the test is again applied with the VIER having a greater high key pointed to by the horizontal pointer. This procedure continues until a VIER is found having a high key that is greater than or equal to the key of the data set. Comparisons are then made with the entries in the VIER's sections. Eventually, an entry is found with a key greater than or equal to the data set key. This entry points to a VIER at the next-lower level.

The search proceeds to successively lower levels until an entry in a level-two VIER is found whose key is greater than or equal to the key of the data set. This entry points to a level-one VIER that, in turn, contains an entry with a key that is equal to the data set key and that points to the format-1 DSCB for the desired data set.

Special Cases in the Search for a DSCB

If there is only one level in the VTOC index, the entries in the VIERs all point to format-1 DSCBs, so only one level need be searched.

If an update to the VTOC index requires a new VIER and the update is interrupted (for example, because of an I/O error or a system failure), the entry in the level-n VIER may contain a key that is greater than the high key in the lower-level VIER pointed to by that entry. In this case, two VIERs at level n-1 may have to be searched. This situation is corrected when DADSM next processes the volume.

THE VTOC PACK SPACE MAP (VPSM)

The VPSM accounts for space on a disk pack. It shows what space on the volume has been allocated and what space remains free.

The map contains bit maps of the cylinders and tracks on the volume. A value of one indicates that the cylinder or track has been allocated; a value of zero, that it has not been allocated. The bit representing a cylinder is set to zero if no tracks on the cylinder have been allocated; it is set to one if any track has been allocated. Tracks assigned as alternate tracks are marked as allocated.

The VPSM replaces the chain of format-5 DSCBs, but one empty format-5 DSCB is left in the VTOC to allow for conversion back to a nonindexed VTOC, a process that requires reconstruction of a format-5 DSCB chain.

The format of an index map (including the VPSM) is shown in Figure 15.

00(00)	ID of This Map		
04(04)	RBA of This Map		
08(08)	Horizontal Pointer to Next VIR		
12(0C)	Sequence Number o: First Entry		
16(10)	VRFDA		VRFO
20(14)	FLG1	LUF1	LUOF
24(18)	Size of Large Unit Map		
28(1C)	SUF1	SUBIT	SUOF
32(20)	Size of Small Unit Map		
36(24)	Reserved		VIR
40(28)	RBA of First High-Level VIER		
	Large Unit Map (VTOC Pack Space Map Only)		
	Small Unit Map		
	VTOC Recording Facility Data (VTOC Index Map Only)		

Figure 15. An Index Map

THE VTOC INDEX MAP (VIXM)

The VIXM contains a bit map in which each bit represents one VTOC index record (VIR). The status of the bit indicates whether the VIR is allocated (1) or unallocated (0).

An area of the VIXM is reserved for VTOC recording facility (VRF) data. (This is the facility that allows detection of and recovery from certain errors in an indexed VTOC.)

A field in the first VIXM record points to the first high-level VIER. Another field in the first VIXM record (VIR in Figure 16) contains the number of VTOC index records which contain all the space maps.

THE VTOC MAP OF DSCBS (VMDS)

The VMDS contains a bit map in which each bit represents one DSCB in the VTOC. The status of the bit indicates whether the DSCB is allocated (1) or unallocated (0).

Name	Offset	Bytes	Description
VIMAP	00(00)	2048	VTOC map
VIMH	00(00)	44	VTOC map header
VIMID	00(00)	4	Map ID in EBCDIC ('VPSM', 'VIXM', or 'VMDS')
VIMRBA	04(04)	4	RBA of this map
VIMHZPTR	08(08)	4	Horizontal RBA pointer to next VIR of this map
VIMORG	12(0C)	4	Sequence number of the first entry in the map
VIMVRFDA	16(10)	2	Offset to current VRF data (if VIMVRFSW=1) or offset where VRF data may be written (if VIMVRFSW=0), (first VIXM only)
VIMVRFO	18(12)	2	Offset to VRF area (first VIXM VIR only)
VIMFLG1	20(14)	1	Flag byte
VIMVRFSW		1...	VRF data exists if 1
		.xxx xxxx	Reserved
VIMLUF1	21(15)	1	Large unit flag byte
VIMLUOF	22(16)	2	Offset into VIR of large unit map (zero if none)
VIMLUSZ	24(18)	4	Size in bits of large unit map
VIMSUF1	28(1C)	1	Small unit flag byte
VIMSUBIT	29(1D)	1	Number of small unit bits per large unit (zero if none)
VIMSUOF	30(1E)	2	Offset into VIR of small unit map
VIMSUSZ	32(20)	4	Size in bits of small unit map
	36(24)	3	Reserved
VIMVIR	39(27)	1	Number of map records (VIXM only)
VIMFHLV	40(28)	4	RBA of first high-level VIER (VIXM only)
VIMLUMAP	44(2C)	kk	Large unit map (kk is VIMLUSZ/8, rounded up)
VIMSUMAP	mm	nn	Small unit map (mm is VIMSUOF, nn is VIMSUSZ/8, rounded up)
VIMVRF	pp	qq	VRF area (pp is VIMVRFO, qq is remainder of first VIXM)

Figure 16. The Format of a VTOC Map

STRUCTURE OF AN INDEXED VTOC

An indexed VTOC is identical to a nonindexed VTOC, except that for an indexed VTOC only a single format-5 DSCB exists and is empty, and certain format-4 DSCB data (the number of format-0 DSCBs and the CCHHR of the highest format-1 DSCB) is not maintained by DADSM. The DOS bit (bit 0 in field DS4VTOCI), set to one in the format-4 DSCB, indicates that these fields (as well as the format-5 DSCB) cannot be relied on. The index bit (bit 7 in field DS4VTOCI) is set in the format-4 DSCB; it indicates that a VTOC index exists.

SCRATCH/RENAME/ALLOCATE RESTRICTIONS

A VTOC index data set may not be scratched if the VTOC index is active. Neither may a VTOC index data set be renamed if the VTOC index is active, unless it is being renamed to another name beginning with 'SYS1.VTOCIX.'. A data set may not be renamed to a name beginning with 'SYS1.VTOCIX.' if there is already such a data set on the volume. Only one data set whose name begins with 'SYS1.VTOCIX.' may be allocated on a volume.

CREATING THE VTOC AND VTOC INDEX

To prepare a volume for use (to initialize it), the Device Support Facilities utility is used. One of the things this utility does is to build the VTOC. After initialization, this VTOC will contain a format-4 DSCB and a format-5 DSCB. For a nonindexed VTOC, the format-5 DSCB contains an extent entry for all the free space on the volume; the initial number of extents in the format-5 DSCB is one or two, depending on where the VTOC is located on the volume. If the VTOC is located somewhere other than at the beginning or end of the volume, two extent entries are needed to describe the free space that precedes and follows it. For an indexed VTOC, the format-5 DSCB contains a zero.

A VTOC index can be created when a volume is initialized by using the Device Support Facilities command INIT and specifying the INDEX key word.

A nonindexed VTOC can be converted to an indexed VTOC by using the command BUILDIX and specifying the IXVTOC keyword. The reverse is also possible by using the BUILDIX command and specifying the OSVTOC keyword.

For more detailed information, refer to Device Support Facilities User's Guide and Reference.

PROTECTING A VTOC AND VTOC INDEX

RESOURCE ACCESS CONTROL FACILITY (RACF)

You can protect the VTOC and VTOC index by using the Resource Access Control Facility (RACF). This is done by defining the volume serial entity under the RACF class DASDVOL. A user must be authorized to the DASDVOL/volume serial entity at the following levels:

- At the UPDATE level, to open the VTOC for output processing
- At the UPDATE level, to open for output processing any data set whose name begins with 'SYS1.VTOCIX.'
- At the ALTER level, to allocate, rename, or scratch any data set whose name begins with 'SYS1.VTOCIX.'
- At the ALTER level, to rename a data set to any name that begins with 'SYS1.VTOCIX.'

Neither the VTOC nor the VTOC index is protected from being opened for input processing by the DASDVOL/volume serial entity.

Note that neither the VTOC nor the VTOC index can be protected through the RACF class DATASET.

AUTHORIZED PROGRAM FACILITY (APF) REQUIREMENTS

A program must be authorized by the authorized program facility (APF) to perform any of the following functions:

- Opening a VTOC for output processing
- Opening for output processing a data set whose name begins with 'SYS1.VTOCIX.'
- Allocating, renaming, or scratching any data set whose name begins with 'SYS1.VTOCIX.'
- Renaming a data set to any name that begins with 'SYS1.VTOCIX.'

PASSWORD PROTECTION

The VTOC index data set may be password protected. The protection is the same as for any password-protected data set. Password checking is bypassed if the volume on which the VTOC index resides is protected by RACF through the DASDVOL class.

COPY/RESTORE/INITIALIZE REQUIREMENTS

OPERATIONS ON VOLUMES CONTAINING AN UNINDEXED VTOC

- Restoring a Volume from a Dump Tape. There are no operational requirements if you change the volume serial number or do a partial restore that does not modify the VTOC. If you do a restore and change the VTOC size without changing the volume serial number, the volume must be varied offline after it is restored. You should not do a restore on a volume with an indexed VTOC.
- Copying a Volume. There are no operational requirements if you change the volume serial number or do not modify the VTOC of the receiving volume. If you do a copy and change the VTOC size without changing the volume serial number, the volume must be varied offline after it is copied. You should not do a copy from a volume with an indexed VTOC.

OPERATIONS ON VOLUMES CONTAINING AN INDEXED VTOC

You should use Device Support Facilities to convert a VTOC to a nonindexed format to update the volume. If you do not, take note of the following information:

- Initializing a Volume. If you do not change the volume serial number, the volume should be varied offline before starting the job.
- Restoring a Volume from a Dump Tape. There are no operational requirements if you change the volume serial number or do a partial restore that does not modify the VTOC or VTOC index. If you do a restore and modify the VTOC or VTOC index without changing the volume serial number, the volume should be varied offline after it is restored.
- Copying a Volume. There are no operational requirements if you change the volume serial number of the receiving volume or do a partial dump without modifying the VTOC or VTOC index. If you modify the VTOC or VTOC index without changing the volume serial number, the receiving volume should be varied offline after it is copied.
- Shared DASD Considerations. In shared DASD environments, whenever the VTOC index is modified or relocated, or the volume is changed from indexed VTOC to OS VTOC, or from OS

VTOC to indexed VTOC, the device should be varied offline to the sharing system or systems.

USING THE OBTAIN, SCRATCH, AND RENAME MACROS

This section tells how to use the OBTAIN, SCRATCH, and RENAME macro instructions. These macros are most commonly used by the operating system and the data set utility programs (IEHMOVE, IEBCOPY, and IEHPROGM), but you may use them in your own routines. The functions you can perform with these macros are:

- Reading a data set control block from the VTOC—OBTAIN
- Deleting a data set—SCRATCH
- Changing the name of a data set—RENAME

You can read a data set control block (DSCB) into virtual storage by using the OBTAIN and CAMLST macro instructions. There are two ways to specify the DSCB that you want to read: by using the name of the data set associated with the DSCB, or by using the absolute track address of the DSCB. You must provide a 140-byte data area in virtual storage, into which the DSCB will be read. When you specify the name of the data set, an identifier (format-1 or format-4) DSCB is read into virtual storage. To read a DSCB other than a format-1 or a format-4 DSCB, you must specify an absolute track address (see "Example" on page 48). (DSCB formats and field descriptions are contained in Debugging Handbook.)

You can delete a non-VSAM data set by using the SCRATCH and CAMLST macro instructions. This causes the DSCBs for the data set to be deleted.

You can change a data set name by using the RENAME and CAMLST macro instructions. This causes the data set name in the format-1 DSCB for the data set to be replaced with the new name.

Accompanying the descriptions of the macro instructions are coding examples, programming notes, and exception return code descriptions.

Note: OBTAIN, SCRATCH, and RENAME macro instructions cannot be used with a SYSIN or SYSOUT data set.

READING A DSCB BY NAME (OBTAIN AND CAMLST SEARCH): If you specify a data set name using OBTAIN and the CAMLST SEARCH option, the 96-byte data portion of the identifier (format-1) DSCB and the absolute track address of the DSCB are read into virtual storage. The absolute track address is a 5-byte field in the form CCHHR. The absolute track address field will contain zeros for VSAM and VIO data sets.

The format is:

[symbol]	**OBTAIN**	listname-addrx
listname	**CAMLST**	**SEARCH** ,dsname-relexp ,vol-relexp ,wkarea-relexp

listname-addrx
points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

SEARCH
this operand must be coded as shown.

dsname-relexp
: specifies the virtual storage location of a fully qualified data set name. The area that contains the name must be 44 bytes long.

    Note: A DSNAME of 44 bytes of X'04' (X'040404...04') can be used to read a format-4 DSCB.

vol-relexp
: specifies the virtual storage location of the 6-byte volume serial number of the volume on which the DSCB is located.

wkarea-relexp
: specifies the virtual storage location of a 140-byte work area that you must define.

Example: In the following example, the identifier (format-1) DSCB for data set A.B.C is read into virtual storage using the SEARCH option. The serial number of the volume containing the DSCB is 770655.

```
          OBTAIN    DSCBABC             READ DSCB FOR DATA
*                                       SET A.B.C INTO DATA
*                                       AREA NAMED WORKAREA

       Check Return Codes

DSCBABC   CAMLST    SEARCH,DSABC,VOLNUM,WORKAREA
DSABC     DC        CL44'A.B.C'         DATA SET NAME
VOLNUM    DC        CL6'770655'         VOLUME SERIAL NUMBER
WORKAREA  DS        140C                140-BYTE WORK AREA
```

The OBTAIN macro instruction points to the CAMLST macro instruction. SEARCH, the first operand of CAMLST, specifies that a DSCB be read into virtual storage, using the data set name you have supplied at the address indicated in the second operand. DSABC, the second operand, specifies the virtual storage location of a 44-byte area into which you have placed the fully qualified name of the data set whose format-1 DSCB is to be read. VOLNUM, the third operand, specifies the virtual storage location of a 6-byte area into which you have placed the serial number of the volume containing the required DSCB. WORKAREA, the fourth operand, specifies the virtual storage location of a 140-byte work area into which the DSCB is to be returned.

Control will be returned to your program at the next executable instruction following the OBTAIN macro instruction. If the DSCB has been successfully read into your work area, register 15 will contain zeros. Otherwise, register 15 will contain one of the following return codes:

Code	**Meaning**
4(04)	The required volume was not mounted.
8(08)	The format-1 DSCB was not found in the VTOC of the specified volume.
12(0C)	A permanent I/O error was encountered, or an invalid format-1 DSCB was found when processing the specified volume, or an unexpected error return code was received from CVAF (Common VTOC Access Facility).
16(10)	Invalid work area pointer.

After execution of these macro instructions, the first 96 bytes of the work area contain the data portion of the identifier (format-1 or format-4) DSCB; the next 5 bytes contain the absolute track address (CCHHR) of the DSCB. These 5 bytes will contain zeros for VSAM or VIO data sets.

READING A DSCB BY ACTUAL DEVICE ADDRESS (OBTAIN AND CAMLST SEEK): You can read any DSCB from a VTOC using OBTAIN and the CAMLST SEEK option. You specify the SEEK option by coding SEEK as the first operand of the CAMLST macro and by providing the absolute device address of the DSCB you want to read, unless the DSCB is for a VIO data set. Only the SEARCH option can be used to read the DSCB of a VIO data set.

The format is:

[symbol] listname	**OBTAIN** **CAMLST**	listname-addrx **SEEK** ,cchhr-relexp ,vol-relexp ,wkarea-relexp

listname-addrx
points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

SEEK
this operand must be coded as shown.

cchhr-relexp
specifies the virtual storage location of the 5-byte absolute device address (CCHHR) of a DSCB.

vol-relexp
specifies the virtual storage location of the 6-byte volume serial number of the volume on which the DSCB is located.

wkarea-relexp
specifies the virtual storage location of a 140-byte work area that you must define.

Example: In the following example, the DSCB at actual-device address X'00 00 00 01 07' is returned in the virtual storage location READAREA, using the SEEK option. The DSCB resides on the volume with the volume serial number 108745.

```
         OBTAIN   ACTADDR           READ DSCB FROM
*                                   LOCATION SHOWN IN CCHHR
*                                   INTO STORAGE AT LOCATION
*                                   NAMED READAREA

       Check Return Codes

ACTADDR  CAMLST   SEEK,CCHHR,VOLSER,READAREA
CCHHR    DC       XL5'0000000107' ABSOLUTE TRACK ADDRESS
VOLSER   DC       CL6'108745'     VOLUME SERIAL NUMBER
READAREA DS       140C            140-BYTE WORK AREA
```

The OBTAIN macro points to the CAMLST macro. SEEK, the first operand of CAMLST, specifies that a DSCB be read into virtual storage. CCHHR, the second operand, specifies the storage location that contains the 5-byte actual-device address of the DSCB. VOLSER, the third operand, specifies the storage location that contains the volume serial number of the volume on which the DSCB resides. The fourth operand, READAREA, specifies the storage location to which the 140-byte DSCB is to be returned.

Control will be returned to your program at the next executable instruction following the OBTAIN macro instruction. If the DSCB has been successfully read into your work area, register 15 will contain zeros. Otherwise, register 15 will contain one of the following return codes:

Code	Meaning
4(04)	The required volume was not mounted.
8(08)	The format-1 DSCB was not found in the VTOC of the specified volume.
12(0C)	A permanent I/O error was encountered or an unexpected error return code was received from CVAF.
16(10)	Invalid work area pointer.
20(14)	The SEEK option was specified and the absolute track address (CCHH) is not within the boundaries of the VTOC.

DELETING A DATA SET (SCRATCH AND CAMLST SCRATCH): You delete a non-VSAM data set by using the SCRATCH and CAMLST macro instructions. This causes all data set control blocks (DSCBs) for the data set to be deleted, and all space occupied by the data set to be made available for reallocation. If you want to scratch a data set being processed using virtual input/output (VIO), the data set must have been allocated for use by your job. Scratching VIO data sets not allocated to your job is not allowed.

If the data set to be deleted is sharing one or more cylinders with one or more data sets (a split-cylinder data set), the space will not be made available for reallocation until all data sets on the shared cylinders are deleted.

A data set cannot be deleted if the expiration date in the identifier (format-1) DSCB has not passed, unless you choose to ignore the expiration date. You specify that the expiration date is to be ignored by using the OVRD option in the CAMLST macro instruction.

For information on RACF-defined data sets, see Resource Access Control Facility (RACF): General Information Manual. You may only scratch a RACF-defined data set (that is, the DSCB indicates RACF-defined) if you have alter access authority to either the data set/volume serial in the DATASET class, or to the volume serial in the DASDVOL class (if the volume is RACF-defined).

If a data set to be deleted is stored on more than one volume, either a device must be available on which to mount the volumes, or at least one volume must be mounted. In addition, all other required volumes must be serially mountable.

When deleting a data set, you must build a volume list in virtual storage. This volume list consists of an entry for each volume on which the data set resides. The first two bytes of the list indicate the number of entries in the list. Each 12-byte entry consists of a 4-byte device code, a 6-byte volume serial number, and a 2-byte scratch status code which should be initialized to zero. Device codes are presented in Debugging Handbook in the description of UCBTYP.

If the space to be deleted is a VSAM data space, you must use the DELETE command provided by access method services. For complete information about the DELETE command, see Access Method Services Reference.

Volumes are processed in the order that they appear in the volume list. The volume at the beginning of the list is processed first. If a volume is not mounted, a message is issued to the operator requesting a volume be mounted. (A volume mount message will not be issued for a mass storage system (MSS) virtual volume; however, a status code will be returned to your program.) This is only done if register 0 has been loaded with the UCB associated with the device on which unmounted volumes are to be mounted. (The device must be allocated to your job.) If you do not load register 0 with a UCB address, its contents must be zero, and at least one of the volumes in the volume list must be mounted before the SCRATCH macro instruction is issued.

If the requested volume cannot be mounted, the operator issues a reply indicating that the request cannot be fulfilled. A status code is then set in the last byte of the volume pointer (the second byte of the scratch status code) for the unavailable volume, and the next volume indicated in the volume list is processed.

The format is:

[symbol] listname	SCRATCH CAMLST	listname-addrx SCRATCH ,dsname-relexp ,,vol list-relexp [,,OVRD]

listname-addrx
   points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

SCRATCH
   this operand must be coded as shown.

dsname-relexp
   specifies the virtual storage location of a fully qualified data set name. The area that contains the name must be 44 bytes long. The name must be defined by a C-type define constant (DC) instruction.

vol list-relexp
   specifies the virtual storage location of an area that contains a volume list. The area must begin on a halfword boundary.

OVRD
   when coded as shown, specifies that the expiration date in the DSCB should be ignored.

Example: In the following example, data set A.B.C is deleted from two volumes. The expiration date in the identifier (format-1) DSCB is ignored.

```
         SR       0,0               SET REG 0 TO ZERO
         SCRATCH  DELABC            DELETE DATA SET A.B.C
*                                   FROM TWO VOLUMES,
*                                   IGNORING EXPIRATION
*                                   DATE IN THE DSCB

      Check Return Codes and SCRATCH Status Codes

DELABC   CAMLST   SCRATCH,DSABC,,VOLIST,,OVRD
DSABC    DC       CL44'A.B.C'       DATA SET NAME
VOLIST   DC       H'2'              NUMBER OF VOLUMES
         DC       X'30C0200D'       3330 DISK DEVICE CODE
         DC       CL6'000017'       VOLUME SERIAL NO.
         DC       H'0'              SCRATCH STATUS CODE
         DC       X'30C0200D'       3330 DISK DEVICE CODE
         DC       CL6'000018'       VOLUME SERIAL NO.
         DC       H'0'              SCRATCH STATUS CODE
```

The SCRATCH macro instruction points to the CAMLST macro instruction. SCRATCH, the first operand of CAMLST, specifies that a data set be deleted. DSABC, the second operand, specifies the virtual storage location of a 44-byte area into

which you have placed the fully qualified name of the data set to be deleted. VOLIST, the fourth operand, specifies the virtual storage location of the volume list you have built. OVRD, the sixth operand, specifies that the expiration date in the DSCB of the data set to be deleted be ignored.

When you attempt to delete a password-protected data set which is not also RACF-protected, the operating system issues a message (IEC301A) to ask the operator at the console or terminal operator of a remote console to enter the password. The data set will be scratched only if the password supplied is associated with a WRITE protection mode indicator. The protection mode indicator is described under "Chapter 5. Password Protecting Your Data Sets" on page 113.

Control is returned to your program at the next executable instruction following the SCRATCH macro instruction. If the data set has been successfully deleted, register 15 will contain zeros and the scratch status code in the volume list entry for each volume will be set to zero. Otherwise, register 15 will contain one of the return codes that follow. To determine whether the data set has been successfully deleted from each volume on which it resides, you must examine the scratch status code, the last byte of each entry in the volume list.

Code **Meaning**

4(04) No volumes containing any part of the data set were mounted, nor did register 0 contain the address of a unit that was available for mounting a volume of the data set. The data set may be a VIO data set that was not allocated during your job. (This return code is accompanied by a scratch status code of 5 in each entry of the volume list.)

8(08) An unusual condition was encountered on one or more volumes.

12(0C) The volume list passed was invalid. The scratch status code, the last byte of each volume list entry, will not have been modified during scratch processing.

After the SCRATCH macro instruction is executed, the last byte of each 12-byte entry in the volume list indicates the following conditions in binary codes:

Scratch Status Code **Meaning**

0 All DSCBs for the data set have been deleted from the VTOC on the volume pointed to.

1 The VTOC of this volume does not contain the format-1 DSCB for the data set to be deleted.

2 The macro instruction failed when the correct password was not supplied in the two attempts allowed, or an attempt was made to scratch a VSAM data space or data set cataloged in an ICF catalog.

3 The data set was not deleted from this volume because either the OVRD option was not specified or the retention cycle has not expired.

4 A permanent I/O error was encountered, or an invalid format-1 DSCB was found when processing this volume, or an unexpected error return code was received from CVAF.

5 It could not be verified that this volume was mounted, and no device was available on which this volume could be mounted.

Scratch Status Code **Meaning**

6 The operator was unable to mount this volume. For MSS, a volume mount failure occurred. For a JES3-managed virtual volume, JES3 would not allow the volume to be mounted.

7 The specified data set could not be scratched because it was being used.

8 The DSCB indicates the data set is defined to RACF but either the accessor is not authorized to the data set or to the volume, or the data set is a VSAM data space, or the data set is cataloged in an ICF catalog, or the data set is not defined to RACF.

9 The data set is defined to RACF but its definition could not be deleted by RACF.

RENAMING A DATA SET (RENAME AND CAMLST RENAME): You rename a data set that is not cataloged in an ICF or VSAM catalog by using the RENAME and CAMLST macro instructions. This causes the data set name in all format-1 DSCBs for the data set to be replaced by the new name that you supply. (VIO data sets cannot be renamed.)

If a data set to be renamed is stored on more than one volume, either a device must be available on which to mount the volumes, or at least one volume must be mounted. In addition, all other volumes of the data set must be serially mountable.

For information on RACF-defined data sets, see Resource Access Control Facility (RACF): General Information Manual. Only an accessor with alter access authority may rename a RACF-defined data set.

When renaming a data set, you must build a volume list in virtual storage. This volume list consists of an entry for each volume on which the data set resides. The first two bytes of the list indicate the number of entries in the list. Each 12-byte volume list entry consists of a 4-byte device code, a 6-byte volume serial number, and a 2-byte rename status code which should be initialized to zero. Device codes are presented in Debugging Handbook. Volumes are processed in the order in which they appear in the volume list. The first volume on the list is processed first. If a volume is not mounted, a message is issued to the operator requesting that the volume be mounted. (A volume mount message will not be issued for an MSS volume; however, a status code will be returned to your program.) This is only done if you indicate the direct access device on which unmounted volumes are to be mounted by loading register 0 with the address of the UCB associated with the device to be used. (The device must be allocated to your job.) If you do not load register 0 with a UCB address, its contents must be zero, and at least one of the volumes in the volume list must be mounted before the RENAME macro instruction is executed.

If the operator cannot mount a volume in the volume list, a reply is issued that the request cannot be fulfilled. A status code is then set in the last byte of the volume list entry (the second byte of the rename status code) for the unavailable volume, and the next volume indicated in the volume list is processed or requested.

The format is:

[symbol] listname	RENAME CAMLST	listname-addrx RENAME ,dsname-relexp ,new name-relexp ,vol list-relexp

listname-addrx
: points to the parameter list (labeled listname) set up by the CAMLST macro instruction.

RENAME
: this operand must be coded as shown.

dsname-relexp
: specifies the virtual storage location of a fully qualified data set name to be replaced. The area that contains the name must be 44 bytes long. The name must be defined by a C-type define constant (DC) instruction.

new name-relexp
: specifies the virtual storage location of a fully qualified data set name that is to be used as the new name. The area that contains the name must be 44 bytes long. The name must be defined by a C-type Define Constant (DC) instruction.

vol list-relexp
: specifies the virtual storage location of an area that contains a volume list. The area must begin on a halfword boundary.

Example: In the following example, data set A.B.C is renamed D.E.F. The data set resides on two volumes.

```
         SR       0,0              SET REG 0 TO ZERO
         RENAME   DSABC            CHANGE DATA SET
                                   NAME A.B.C TO D.E.F

      Check Return Codes and RENAME Status Codes

DSABC    CAMLST   RENAME,OLDNAME,NEWNAME,VOLIST
OLDNAME  DC       CL44'A.B.C'      OLD DATA SET NAME
NEWNAME  DC       CL44'D.E.F'      NEW DATA SET NAME
VOLIST   DC       H'2'             TWO VOLUMES
         DC       X'30C0200D'      3330 DISK DEVICE CODE
         DC       CL6'000017'      VOLUME SERIAL NO.
         DC       H'0'             RENAME STATUS CODE
         DC       X'30C0200D'      3330 DISK DEVICE CODE
         DC       CL6'000018'      VOLUME SERIAL NO.
         DC       H'0'             RENAME STATUS CODE
```

The RENAME macro instruction points to the CAMLST macro instruction. RENAME, the first operand of CAMLST, specifies that a data set be renamed. OLDNAME, the second operand, specifies the virtual storage location of a 44-byte area into which you have placed the fully qualified name of the data set to be renamed. NEWNAME, the third operand, specifies the virtual storage location of a 44-byte area into which you have placed the new name of the data set. VOLIST, the fourth operand, specifies the virtual storage location of the volume list you have built.

Control is returned to your program at the next executable instruction following the RENAME macro instruction. If the data set has been successfully renamed, register 15 will contain zeros, and the rename status code in the volume list entry for each volume will be set to zero. Otherwise, register 15 will contain one of the return codes below. To determine whether the data set has been successfully renamed on each volume on which it resides, you must examine the rename status code, the last byte of each entry in the volume list.

Code	**Meaning**
4(04)	No volumes containing any part of the data set were mounted, nor did register 0 contain the address of a unit that was available for mounting a volume of the data set to be renamed. The data set may be a VIO data set, which can't be renamed. (This return code is accompanied by a rename status code of 5 in each entry of the volume list.)
8(08)	An unusual condition was encountered on one or more volumes.
12(0C)	The volume list passed was invalid. The rename status code, the last byte of each volume list entry, will not have been modified during rename processing.

After the RENAME macro instruction is executed, the last byte of each 12-byte entry in the volume list indicates one of the following conditions in binary code:

Rename Status Code	**Meaning**
0	The format-1 DSCB for the data set has been renamed in the VTOC on the volume pointed to.
1	The VTOC of this volume does not contain the format-1 DSCB for the data set to be renamed.
2	The macro instruction failed when the correct password was not supplied in the two attempts allowed, or the user tried to rename a VSAM data space or VSAM data set cataloged in an ICF catalog.
3	A data set with the new name already exists on this volume.
4	A permanent I/O error was encountered, or an invalid format-1 DSCB was found when trying to rename the data set on this volume, or an unexpected error return code was received from CVAF.
5	It could not be verified that the volume was mounted, and no device was available on which the volume could be mounted.
6	The operator was unable to mount this volume. For MSS, a volume mount failure occurred. For a JES3-managed virtual volume, JES3 would not allow the volume to be mounted.
7	The specified data set could not be renamed on this volume because it was being used.
8	The data set is defined to RACF but either the accessor is not alter authorized to the data set or the data set is defined to RACF on multiple volumes.

When you attempt to rename a password-protected data set, the operating system issues a message (IEC301A) to ask the operator or remote console operator to verify the password. The data set will be renamed only if the password supplied is associated with a WRITE protection mode indicator. The protection mode indicator is described under "Chapter 5. Password Protecting Your Data Sets" on page 111.

USING VTOC ACCESS MACROS

VTOC access macros enable you to:

- Determine whether a UCB points to an indexed VTOC (the CVAFTST macro)
- Directly access DSCBs and VTOC index records (the CVAFDIR macro)
- Read DSCBs in physical-sequential order, beginning with the DSCB you specify (the CVAFSEQ macro)
- Read DSCBs in data-set-name order using the VTOC index (the CVAFSEQ macro)
- Obtain free space information from each of the three index maps (the CVAFDSM macro)

If your program is unauthorized, you must open the VTOC to supply a DEB address, created by SAM or EXCP, to the CVAFDIR, CVAFDSM, or CVAFSEQ macros; the status of the VTOC will then be determined by CVAF and indicated in the CVPL by the CV1IVT bit.

In the sections that follow, VTOC access macros are described in general terms. Their syntax is explained in "Appendix A. VTOC Access Macros" on page 184.

OVERVIEW OF THE CVAFTST MACRO

The CVAFTST macro determines whether the system supports an indexed VTOC, and, if it does, whether the VTOC on the unit whose UCB is supplied is indexed or nonindexed.

You will get a return code of 12 if CVAFTST cannot determine whether an indexed or nonindexed VTOC is on the unit's volume. You should not receive a return code of 12 from CVAFTST if you have opened a data set (including the VTOC) on the volume.

You need no authorization to issue the CVAFTST macro.

The syntax of CVAFTST is explained in "Appendix A. VTOC Access Macros" on page 184. Return codes are explained in "Appendix C. Return Codes from VTOC Access Macros" on page 221.

OVERVIEW OF THE CVAFDIR MACRO

For an indexed or nonindexed VTOC, the CVAFDIR macro may be used to:

- Read or write a DSCB by specifying the name of the data set it represents
- Read or write a DSCB by specifying its address

In addition, for an indexed VTOC, the macro may be used to:

- Read or write VTOC index records
- Read and retain in virtual storage the first high-level VIER, and VIERs used during an index search.
- Read and retain in virtual storage the space map VIRs
- Free VIRs retained in virtual storage

The syntax of CVAFDIR is explained in "Appendix A. VTOC Access Macros" on page 184. A description of how to use it is under "How to Use the CVAFDIR Macro" on page 59.

OVERVIEW OF THE CVAFSEQ MACRO

The CVAFSEQ macro may be used to:

- Read an indexed VTOC sequentially, in data-set-name (DSN) order
- Read an indexed VTOC or a nonindexed VTOC in physical-sequential order

A description of how to use it is under "How to Use the CVAFSEQ Macro" on page 62.

The syntax of CVAFSEQ is explained in "Appendix A. VTOC Access Macros" on page 184.

OVERVIEW OF THE CVAFDSM MACRO

The CVAFDSM macro may be used for an indexed VTOC to:

- Obtain one or more extents that describe unallocated space on the volume
- Obtain a count of free DSCBs on the VTOC
- Obtain a count of free VTOC index records in the VTOC index.

The syntax of CVAFDSM is explained in "Appendix A. VTOC Access Macros" on page 184. A description of how to use it is under "How to Use the CVAFDSM Macro" on page 63.

BUFFER LISTS

A buffer list consists of one or more chained control blocks, each with a header and buffer list entries. The header indicates whether the buffer list is for DSCBs or VTOC index records. The entries point to and describe the buffers.

Buffer lists can be created in two ways:

- Directly, when you fill in the arguments and buffer addresses of DSCBs or VIRs to be read or written
- Indirectly, when you code the IXRCDS=KEEP and/or MAPRCDS=YES keywords

Buffer List Header

The header of the buffer list indicates whether the buffer list describes buffers for DSCBs or VTOC index records. The DSCB bit must be set to one and the VIR bit must be set to zero in order for CVAF to process a request to read or write a DSCB. The protect key and subpool fields in the buffer list header are used by CVAF only if ACCESS=RLSE is coded.

The buffer list header contains a count of the number of entries in the buffer list.

The forward chain address is used to chain buffer lists together. DSCB buffer lists must not be chained to VIR buffer lists and VIR buffer lists must not be chained to DSCB buffer lists.

The format of the buffer list header is shown in Figure 17 on page 57.

Name	Offset	Bytes	Description
BFLHDR	0(00)	8	Buffer list header
BFLHNOE	0(00)	1	Number of entries
BFLHFL	1(01)	1	Flag byte and key
BFLHKEY		xxxx	Protect key of buffer list and buffers
BFLHVIR		.... 1...	Buffer list entries describe VIRs
BFLHDSCB		.... .1..	Buffer list entries describe DSCBs
BFLHRSV6		.... ..x.	Reserved
BFLHRSV7		x	Reserved
BFLHRSV	2(02)	1	Reserved
BFLHSP	3(03)	1	Identifies the subpool of buffer list and buffers
BFLHFCHN	4(04)	4	Forward chain address of next buffer list

Figure 17. Format of a Buffer List Header

Buffer List Entry

A buffer list contains one or more entries. Each entry provides the buffer address, the length of the DSCB or VIR, the argument, and an indication whether the argument is an RBA, a TTR, or a CCHHR.

The fields and bit uses are listed below.

- For a VIR buffer, the TTR and CCHHR bits must be 0 and the RBA bit must be 1.
- For a DSCB buffer, the RBA bit must be 0, and only one of the TTR or CCHHR bits may be set to 1.
- The BFLEAUPD bit is an output indicator from CVAF that the BFLEARG field of a VIR buffer list was updated.
- The BFLEMOD bit indicates that a VIR buffer was modified and must be written; if no BFLEMOD bits are on in any of the entries for a CVAFDIR ACCESS=WRITE, all buffers are written.
- The BFLESKIP bit is used to cause an entry to be ignored.
- The BFLEIOER bit is an output indicator from CVAF to indicate an I/O error occurred during reading or writing of the DSCB or VIR.
- The BFLELTH field is the length of the buffer; for a DSCB buffer, the length must be 96 or 140; for a VIR buffer, the length must be the length of the buffer divided by 256.
- The BFLEARG field is the argument of the DSCB or VIR; the three possible formats of the 5-byte field are:
  - CCHHR=5 byte CCHHR
  - TTR=0TTR0
  - RBA=One byte of 0 followed by a 4-byte RBA

The format of the buffer list entry is shown in Figure 18 on page 58.

Name	Offset	Bytes	Description
BFLE	0(00)	12	Buffer list entry
BFLEFL	0(00)	1	Flag byte
BFLERBA		1...	Argument is RBA
BFLECHR		.1..	Argument is CCHHR
BFLETTR		..1.	Argument is TTR
BFLEAUPD		...1	CVAF updated argument field
BFLEMOD		.... 1...	Data in buffer has been modified
BFLESKIP		.... .1..	Skip this entry
BFLEIOER		.... ..1.	I/O error
BFLERSV7		x	Reserved
BFLERSV	1(01)	1	Reserved
BFLELTH	2(02)	1	Length of VIR buffer divided by 256 or length of DSCB buffer
BFLEARG	3(03)	5	Argument of VIR or DSCB
BFLEATTR	4(04)	3	TTR of DSCB
BFLEARBA	4(04)	4	RBA of VIR
BFLEBUF	8(08)	4	Buffer address

Figure 18. Format of a Buffer List Entry

THE CVAF PARAMETER LIST (CVPL)

A CVPL is generated by using the CVAFDIR, CVAFDSM, or CVAFSEQ macro with MF=L or MF=I specified or with MF not specified (MF=I is the default).

The CVPL passes information to CVAF. CVAF, in turn, returns information in the CVPL. The CV1IVT bit indicates whether an indexed or nonindexed VTOC is being accessed. The CVSTAT field contains feedback when an error occurs. The address of the map records buffer list is returned in the CVMRCDS field. The address of the VIER buffer list is returned in the CVIRCDS field. The CVAF I/O area address is returned in the CVIOAR field.

The CVPL generated by the MF=L or MF=I form of the CVAFDIR, CVAFDSM, or CVAFSEQ macro may be used (through the MF=E keyword) to execute a different macro from the one that generated the CVPL.

The format of the CVPL is shown in Figure 19 on page 59.

IDENTIFYING THE VTOC

The VTOC must be identified to CVAF by supplying either the address of a UCB (with the UCB keyword) or the address of a DEB opened to the VTOC (with the DEB keyword).

An unauthorized caller must supply the address of a SAM or EXCP DEB open to the VTOC. The DEB can be obtained by opening a DCB using the RDJFCB and OPEN TYPE=J macros. The DCBs DDNAME is that of a DD statement allocated to the unit whose VTOC is to be accessed. After issuing the RDJFCB macro, the JFCBDSNM field is overlaid with the data set name of the format-4 DSCB: 44X'04'. The DCB is opened for INPUT using OPEN TYPE=J. The DEB address is in DCB field, DCBDEBA. The OPEN macro is described under "OPEN—Initialize Data Control Block for Processing the JFCB" on page 147 and the RDJFCB macro is described under "RDJFCB—Read a Job File Control Block" on page 148.

Name	Offset	Bytes	Description
CVPL			
CVLBL	00(00)	4	EBCDIC "CVPL"
CVLTH	04(04)	2	Length of parameter list
CVFCTN	06(06)	1	Function Byte
CVDIRD			X'01'-CVAFDIR ACCESS=READ
CVDIWR			X'02'-CVAFDIR ACCESS=WRITE
CVDIRLS			X'03'-CVAFDIR ACCESS=RLSE
CVSEQGT			X'04'-CVAFSEQ ACCESS=GT
CVSEQGTE			X'05'-CVAFDIR ACCESS=GTEQ
CVDMMAP			X'0A'-CVAFDSM ACCESS=MAPDATA
CVSTAT	07(07)	1	Status Information
CVFL1	08(08)	1	First Flag Byte
CV1IVT		1...	Indexed VTOC Accessed
CV1IOAR		.1..	IOAREA=KEEP
CV1PGM		..1.	BRANCH=(YES,PGM)
CV1MRCDS		...1	MAPRCDS=YES
CV1IRCDS		.... 1...	IRCDS=KEEP
CV1MAPIX		.... .1..	MAP=INDEX
CV1MAPVT		.... ..1.	MAP=VTOC
CV1MAPVL		1	MAP=VOLUME
CVFL2	09(09)	1	Second Flag Byte
CV2HIVIE		1...	HIVIER=YES
CV2VRF		.1..	VRF Information Exists
CV2CNT		..1.	COUNT=YES
CV2RCVR		...1	RECOVER=YES
CV2SRCH		.... 1...	SEARCH=YES
CV2DSNLY		.... .1..	DSNONLY=YES
CV2VER		.... ..1.	VERIFY=YES
CV2RSV7		x	Reserved
CVRSVB	10(0A)	2	Reserved
CVUCB	12(0C)	4	UCB address
CVDSN	16(10)	4	Data set name address
CVBUFL	20(14)	4	Buffer list address
CVIRCDS	24(18)	4	Index VIRs buffer list address
CVMRCDS	28(1C)	4	Map VIRs buffer list address
CVIOAR	32(20)	4	I/O area address
CVDEB	36(24)	4	DEB address
CVARG	40(28)	4	Argument address
CVSPACE	44(2C)	4	SPACE parameter list address
CVEXTS	48(30)	4	Extent table address
CVBUFL2	52(34)	4	New VRF VIXM buffer list address
CVVRFDA	56(38)	4	VRF data address
CVCTAR	60(3C)	4	Count area address

Figure 19. Format of the CVAF Parameter List

If a CVAF macro call has specified IOAREA=KEEP, then a subsequent CVAF call using a different CVPL may omit the UCB and DEB keywords, and supply the IOAREA address from the other CVPL. You can use the IOAREA keyword to do this.

The above does not apply to the CVAFTST macro. Only a UCB may be supplied to identify the VTOC, and no authorization is required.

HOW TO USE THE CVAFDIR MACRO

CVAFDIR may be used to read or write a DSCB. For indexed VTOCs, CVAFDIR may be used to read or write VTOC index records.

After a CVAFDIR call, the CVAF parameter list bit, CV1IVT, may be tested to determine if the VTOC is indexed or nonindexed.

Specifying a Data Set Name to Read or Write a DSCB

To read or write a DSCB by specifying only a data set name, ACCESS=READ or ACCESS=WRITE must be coded.

The address of the data set name is supplied in the DSN keyword; the buffer list address is supplied in the BUFLIST keyword.

The buffer list must have at least one buffer list entry with the skip bit off and a pointer to a 96- or 140-byte buffer. Buffer lists may be chained together, but only the first eligible entry will be used.

For an indexed VTOC, the index will be searched for the data set name and, if it is found, the DSCB argument obtained will be put in the buffer list entry and used to read or write the DSCB. If the data set name is not found in the index, a key search of the VTOC will be performed.

For a nonindexed VTOC, a channel program will be used to do a key search of the VTOC to locate the data set name and read or write the DSCBs. If the data set name is found, the DSCB argument will be put in the buffer list entry.

The DSCB argument returned in the buffer list entry will be in the format determined by the buffer list entry bits BFLECHR or BFLETTR.

If the data set name is not found in the VTOC, register 15 will contain a return code of 4 and CVSTAT will contain an error code of 1.

Specifying the DSCB Location

To read or write a DSCB by specifying the DSCB's location, either ACCESS=READ or ACCESS=WRITE must be coded. The DSN keyword must be supplied but will not be used for a 140-byte DSCB. A buffer list address must be supplied in the BUFLIST keyword. The buffer list must have at least one buffer list entry with the skip bit off and pointing to a 96- or 140-byte buffer. Buffer lists may be chained together, but only the first eligible entry will be used. This procedure is the same for both indexed and nonindexed VTOCs.

If the buffer is for a 96-byte read or write, a channel program will be used to verify that the key in the DSCB is the same as the 44-byte data set name provided before reading or writing the DSCB. If the buffer is for a 140-byte read or write, a channel program will be used to read or write the DSCB at the location provided in the buffer list entry. The data set name will not be used, and the DSCB key will not be read.

If VERIFY=YES is coded and the write is for a 140-byte DSCB, the channel program used for the write will verify that the DSCB is a format-0 DSCB prior to the write.

Reading or Writing VTOC Index Records

VIRs may be read or written explicitly using the BUFLIST keyword or may be read implicitly using the IXRCDS and MAPRCDS keywords. A buffer list address may be supplied in the BUFLIST keyword to read or write one or more VIRs. The buffer list header must have the VIR bit set to one and the DSCB bit set to zero. Each entry in the buffer list (and subsequent buffer lists if more than one is chained) is inspected. If the skip bit is set to zero, the RBA bit is set to one (and the CCHHR and TTR bits are set to zero), and the buffer address is nonzero, the entry will be processed. The RBA in the argument field of the buffer list entry is used to read or write a VIR using the buffer address. Read and write requests will be in the order of entries in the buffer list(s).

For a write request, the modification bit in the buffer list entries is inspected. If the bit is not set in any entry, all are written. The modification bit is set to zero for entries whose VIR is written.

Map records and the first high-level VTOC index entry record may be read by supplying the keywords MAPRCDS=YES and/or IXRCDS=KEEP, and not supplying an address in the CVAF parameter list CVMRCDS/CVIRCDS fields.

Reading Map Records and VIERS

To read and retain in virtual storage the VTOC index map records and first high-level VIER, either ACCESS=READ or ACCESS=WRITE must be coded. Neither the DSN field nor the BUFLIST field is required.

MAPRCDS=YES must be coded to read and retain map records. The CVAF parameter list field CVMRCDS must be zero. CVAF will obtain a buffer list with the number of entries and buffers required to read all the map VIRs. The buffer list address will be put in the CVMRCDS field by CVAF.

IXRCDS=KEEP is coded in order to read and retain the first high-level VIER and (if an index search is required) all VIERs read. If the CVAF parameter list field CVIRCDS is zero, CVAF will obtain a buffer list with entries and buffers and read the first high-level VIER. The number of entries and number of buffers are determined by CVAF. If CVIRCDS is not zero, only VIERs required for an index search will be read.

The integrity of the maps and VIER read can only be ensured if you are enqueued on the VTOC and, in the case of shared DASD, reserved to the unit.

Map and VIER buffers obtained by CVAF, and retained, must be released by a subsequent CVAF call.

Releasing Buffers and Buffer Lists Obtained by CVAF

There are three ways to release buffers and buffer lists obtained by CVAF.

- Code MAPRCDS=NO or MAPRCDS=(NO,addr) for any specification of ACCESS, to free the MAP records buffer list.

- Code IXRCDS=NOKEEP or IXRCDS=(NOKEEP,addr) for any specification of ACCESS, to free the index records buffer list.

- Code ACCESS=RLSE and supply a buffer list address through the BUFLIST keyword for a subsequent CVAF call.

CVAF will free all eligible buffers, and buffer lists if they become empty. Eligible buffers are those pointed to by buffer list entries with the skip bit off. A buffer list will be freed if no buffer list entry has the skip bit on. If buffer lists are chained together, all buffer lists will be checked and freed if appropriate.

You must ensure that you do not request CVAF to release the same buffer list twice by supplying its address in more than one place.

HOW TO USE THE CVAFSEQ MACRO

Each CVAF call will return one of the following:

- One format-1 or format-4 DSCB in indexed (data-set-name) order

- One or more DSCBs in physical-sequential order (but only one DSCB can be requested by an unauthorized caller)

- The next data set name in the index

The DSCBs are read into buffers supplied through the BUFLIST keyword.

The argument of each DSCB read is also supplied in the buffer list. DSCBs of 96 bytes must be requested in the buffer list for indexed access; 140 bytes is required for physical-sequential access.

If indexed order is chosen, the VTOC index is used to return each format-1 or format-4 DSCB whose name is in the index. An option (DSNONLY=YES) allows only the data set names in the VTOC index, and not the DSCBs, to be obtained. In this case, the CCHHR of the DSCB is returned in the argument area supplied through the ARG keyword. The DSN area supplied is updated at each CVAFSEQ call to contain the data set name of each DSCB read.

Initiating Indexed Access (DSN Order)

To initiate indexed access (DSN order), either supply in the area coded through the DSN keyword 44 bytes of binary zeros (to indicate the first data set name in the index) or supply the data set name you wish to serve as the starting place for the index search.

The name returned in the DSN area will be the one equal to or greater than the DSN supplied, depending on the specification of the ACCESS keyword. The DSN field is updated by CVAF.

The ACCESS keyword determines whether the search is for a DSN greater than or equal to that supplied.

If DSNONLY=NO is coded, the DSCB and argument are returned to you using the buffer list provided through the BUFLIST keyword. The first entry in the buffer list with a skip bit of zero and a nonzero buffer address is used. The argument value is supplied if either the TTR or CCHHR bit is set in the buffer list entry. The default is CCHHR. The DSCB size in the buffer list entry must be 96 bytes for indexed access.

If DSNONLY=YES is coded, the CCHHR argument is supplied in the ARG area.

Note that the data set name of the format-4 DSCB is in the index and that its name (44 bytes of X'04') may be returned to you. The format-4 DSCB's name is likely to be the first data set name in the VTOC index.

Initiating Physical-Sequential Access

To initiate physical-sequential access, the DSN keyword must be omitted or DSN=0 must be coded. The argument field in the first buffer list entry must be initialized to zero or to the argument of the DSCB to begin the read. If the argument is zero, the argument used will be the start of the VTOC.

The DSCB size must be set to 140 in buffer list entries.

The specification of ACCESS will determine whether the DSCB whose argument is supplied or the DSCB following it is to be read.

For example, to read the first DSCB (the format-4 DSCB) in the VTOC, the BFLEARG in the first buffer list entry may be set to zero, and ACCESS=GTEQ coded in the CVAFSEQ macro. If ACCESS=GT is subsequently coded, the second DSCB (the first format-5 DSCB) is read.

If you are authorized, as many DSCBs as there are entries in the buffer list will be read with a single CVAF call. Only one DSCB will be read if you aren't authorized.

Only one buffer list is used; a second buffer list chained to the first will not be inspected. All entries in the buffer list will be used for authorized callers. The skip bit will not be inspected. Each entry must have a buffer address, the length field set to 140, and the TTR or CCHHR bit set (if neither bit is set, the CCHHR bit will be set on). Only the first entry will be used for unauthorized callers. The argument field of each buffer list entry will be updated by CVAF with the argument of the DSCB. The argument value is returned in either TTR or CCHHR format, depending on whether the TTR or CCHHR bit is set in the buffer list entry. The default is CCHHR.

Only the argument in the first entry is used to begin the search. Arguments in subsequent entries are not inspected. If a nonzero argument value is supplied in the first entry, there must be a DSCB with that argument.

End-of-data is indicated with a return code of 4 in register 15 and CVSTAT set to X'20'. Each buffer list entry following the last DSCB read has its argument field set to zero (this may be the first entry if no DSCBs are read).

Note that all DSCBs, including format-0 DSCBs, are read. You cannot be certain that you have read all format-1 through -6 DSCBs until the entire VTOC has been read. For a nonindexed VTOC, the CCHHR of the last format-1 DSCB is contained in the format-4 DSCB field DS4HPCHR; format-2 through -6 DSCBs may reside beyond that location. For an indexed VTOC, the VMDS contains information about which DSCBs are format-0 DSCBs.

HOW TO USE THE CVAFDSM MACRO

ACCESS=MAPDATA is used to obtain information contained in the space maps.

To count the number of unallocated VIRs in the VTOC index space map (VIXM), COUNT=YES and MAP=INDEX are coded. The number of unallocated VIRs is returned in the 4-byte area supplied through the CTAREA keyword.

To count the number of format-0 DSCBs, COUNT=YES and MAP=VTOC are coded. The number of format-0 DSCBs in the VTOC map of DSCBs VMDS is returned in the 4-byte area supplied through the CTAREA keyword.

To obtain one or more free space extents from the VTOC pack space map (VPSM), COUNT=NO and MAP=VOLUME are coded. The extents are returned in the area supplied through the EXTENTS keyword. Each extent is returned in a 5-byte XXYYZ format, the same as for a format-5 DSCB extent, where XX is the relative track address (RTA) of the first track of the extent, YY is the number of whole cylinders in the extent, and Z is the number of additional tracks in the extent. The RTA supplied to CVAF in the first (or only) extent will serve as a starting place for the VPSM search; the extent returned will be the next free extent with a higher starting RTA than the one supplied.

If all the unallocated extents in the VPSM are supplied before filling in all the extents supplied, the remaining extents are set to zero. Register 15 is set to 4 on return, with the CVSTAT field in the CVPL set to X'20' to indicate the end of data.

VTOC SERIALIZATION

It is your responsibility to serialize access to the VTOC and the VTOC index when you use VTOC access macros. The ENQ or RESERVE macro instruction with the SYSTEMS parameter is used for this serialization. The qname (major name) is SYSVTOC; the rname (minor name) is the 6-byte volume serial number of the volume. Only authorized programs may ENQ RESERVE using the SYSVTOC qname.

The SYSVTOC qname does not serialize access to the format-1 DSCB for a data set. You must allocate the data set with disposition OLD, MOD, or NEW (not SHR). This causes the proper ENQ, which ensures no other job will update that data set's format-1 DSCB.

Updates to the VTOC index performed without proper serialization will compromise the integrity of the VTOC or VTOC index.

REGISTER USAGE

Register 1 is used to contain the address of the CVAF parameter list (CVPL). Register 15 is used to contain the return code when processing has completed for a function.

VTOC ERROR DIAGNOSIS AND RECOVERY

ACTIONS TAKEN WHEN AN ERROR OCCURS

These actions are taken if an error occurs:

- If an index structure error is detected, DADSM or CVAF will cause the VTOC index to be disabled. The indexed VTOC bit will be zeroed in the format-4 DSCB. A software error record will be written to SYS1.LOGREC. A system dump is taken. The VTOC will be converted to a nonindexed format at the next DADSM allocate or extend call.
- If a program check, machine check, or other error occurs while using a VTOC access macro, a SYS1.LOGREC message is written and a system dump is taken.
- An error code is put in the CVSTAT field of the CVPL. The values and explanations of these error codes are listed in "Appendix D. VTOC Error Message and Associated Codes" on page 223.

RECOVERING FROM SYSTEM OR USER ERRORS

Neither the VTOC nor the VTOC index need be recovered from a user error caused by an unauthorized user, since an unauthorized user cannot modify a VTOC.

A system error will affect a VTOC and VTOC index, probably by interrupting DADSM while it is updating, thus leaving the VTOC and/or the VTOC index in a partially updated state. Both the VTOC and the VTOC index are designed to cause DADSM to recover from such an interruption.

For a nonindexed VTOC (or a VTOC with an index that has been disabled), a subsequent call to DADSM ALLOCATE or EXTEND will cause VTOC convert routines to reestablish the free space (format-5 DSCBs).

For an indexed VTOC, a subsequent call to any DADSM function will cause the recovery of the previous interrupt (either by backing out or completing the interrupted function).

GTF TRACE

A trace facility exists to trace all CVAF calls for VTOC index output I/O, all VTOC output I/O, and all VTOC index and space map modifications. See Common VTOC Access Facility Diagnosis Reference for information on this facility.

LISTING A VTOC AND VTOC INDEX

A VTOC and VTOC index can be listed using the IEHLIST utility program. Dump, formatted, or abridged listings can be obtained by using the LISTVTOC command of IEHLIST.

CHAPTER 3. EXECUTING YOUR OWN CHANNEL PROGRAMS (EXCP)

The execute-channel-program (EXCP) macro instruction provides you with complete control of the device characteristics and the organizing of data. This chapter contains a general description of the function and application of the EXCP macro instruction, accompanied by descriptions of specific control blocks and macro instructions used with EXCP. Factors that affect the operation of EXCP, such as device variations and program modification, are also discussed.

Before reading this chapter, you should be familiar with system functions and with the structure of control blocks, as well as with the operational characteristics of the I/O devices required by your channel programs. Operational characteristics of specific I/O devices are contained in IBM publications for each device.

You also need to understand the information in these publications:

- Data Management Services contains the standard procedures for I/O processing under the operating system.

- OS/VS-DOS/VSE-VM/370 Assembler Language contains the information necessary to code programs in the assembler language.

- Data Management Macro Instructions describes the system macro instructions that can be used in programs coded in the assembler language.

- Debugging Handbook, Volumes 2 and 3, contains format and field descriptions of the system control blocks referred to in this chapter.

The execute-channel-program (EXCP) macro instruction causes a supervisor-call interruption to pass control to the EXCP processor. (I/O process is the name we will use for the EXCP processor and the I/O supervisor. For our purposes, it's unnecessary to understand how input/output processing is divided between the two.) EXCP also provides the I/O supervisor with control information regarding a channel program to be executed. When an IBM access method is being used, an access method routine is responsible for issuing EXCP. If you are not using an IBM access method, you must issue EXCP in your program. (The EXCP macro instruction cannot be used to process SYSIN or SYSOUT data sets.)

You issue EXCP primarily for I/O programming situations to which the standard access methods do not apply. If you are writing your own access method, you must include EXCP for I/O operations. EXCP must be used for processing nonstandard labels, including reading and writing labels and positioning magnetic tape volumes.

To issue EXCP, you must provide a channel program (a list of channel command words) and several control blocks in your program area. The I/O process then schedules I/O requests for the device you have specified, executes the specified I/O commands, handles I/O interruptions, directs error recovery procedures, and posts the results of the I/O requests.

EXECUTING CHANNEL PROGRAMS IN SYSTEM AND PROBLEM PROGRAMS

This section briefly explains the procedures performed by the system and the programmer when EXCP is issued by the routines of IBM access methods. The additional procedures that you must perform when issuing EXCP yourself are then described by direct comparison.

SYSTEM USE OF EXCP

When using an IBM access method to perform I/O operations, the programmer is relieved of coding channel programs and constructing the control blocks necessary for the execution of channel programs. To permit I/O operations to be handled by an access method, the programmer need only issue the following macro instructions:

- A DCB macro instruction, which produces a data control block (DCB) for the data set to be retrieved or stored
- An OPEN macro instruction that initializes the data control block and produces a data extent block (DEB) for the data set
- A macro instruction (for example, GET or WRITE) that requests I/O operations

Access method routines will then:

1. Create a channel program that contains channel commands for the I/O operations on the appropriate device
2. Construct an input/output block (IOB) that contains information about the channel program
3. Construct an event control block (ECB) that is later posted with a completion code each time the channel program terminates
4. Issue an EXCP macro instruction to pass the address of the IOB to the routines that initiate and supervise the I/O operations

The I/O process consists of:

5. Constructing a request queue element (RQE) for scheduling the request
6. If the requestor is in a V=V address space, fixing the buffers so that they cannot be paged out and translating the requestor's virtual channel program into a real channel program
7. Issuing a start I/O (SIO) instruction to cause the channel to execute the real channel program
8. Processing I/O interruptions and scheduling error recovery procedures when necessary
9. Posting a completion code in the event control block after the channel program has been executed

Note: If the requestor is an authorized program in a V=R address space, a real channel program is provided, so item 6 is not performed.

The programmer is not concerned with these procedures and does not know the status of I/O operations until they are completed. Device-dependent operations are limited to those provided by the macro instructions of the particular access method selected.

USE OF EXCP IN PROBLEM PROGRAMS

To issue the EXCP macro instruction directly, you must follow the procedures that the access methods would perform, as summarized in items 1 through 4 of the preceding discussion. You must, in addition to constructing and opening the data control block with the DCB and OPEN macro instructions, construct a channel program, an input/output block, and an event control block before you can issue EXCP. The I/O process generally handles items 5 through 9.

After issuing EXCP, you should issue a WAIT macro instruction, specifying the address of the event control block, to determine whether the channel program has terminated. If volume switching is necessary, you must issue an EOV macro instruction. When all processing of the data set has been completed, you must issue a CLOSE macro instruction to restore the data control block.

EXCP OPERATIONS IN A V=R ADDRESS SPACE

User-constructed channel programs for I/O operations of an authorized program in a V=R address space are not translated. Because the address space is V=R, any CCWs created by the user have correct real data addresses. (Translation would only re-create the user's channel program, so the CCWs are used directly.)

Modification of an active channel program by data read in or by processor instructions is legitimate in a V=R address space, but not in a V=V address space.

EXCP REQUIREMENTS

This section describes the channel program that you must provide in order to issue EXCP. The control blocks that you must either construct directly, or cause to be constructed by use of macro instructions, are also described.

CHANNEL PROGRAM

The channel program supplied by you and executed through EXCP is composed of CCWs on doubleword boundaries. Each channel command word specifies a command to be executed and, for commands initiating data transfer, the area to or from which the data is to be transferred.

Channel command word operation codes used with specific I/O devices can be found in IBM publications for those devices. All channel command word operation codes described in these publications can be used. In addition, both data chaining and command chaining may be used.

To specify either data chaining or command chaining, you must set appropriate bits in the channel command word, and indicate the type of chaining in the input/output block. Both data and command chaining should not be specified in the same channel command word; if they are, data chaining takes precedence.

EXCP does not support channel programs that modify themselves, regardless of the method of modification: data chaining, command chaining, or a program to do the modification. The intended modification in virtual storage has no effect on the running real-channel program (see "Modification of a Channel Program during Execution" on page 71).

CONTROL BLOCKS

When using EXCP, you must be familiar with the function and structure of the IOB, the ECB, the DCB, the DEB, and the IDAW. IOB and ECB fields are illustrated under "Control Block Fields" on page 95. DCB fields are illustrated under "Macro Specifications for Use with EXCP" on page 80. The handling of IDAWs is described under "SIO Appendage" on page 100. Brief descriptions of these control blocks follow.

Input/Output Block (IOB)

The input/output block is used for communication between the problem program and the system. It provides the addresses of other control blocks, and maintains information about the channel program, such as the type of chaining and the progress of I/O operations. You must define the input/output block and specify its address as the only parameter of the EXCP macro instruction.

Event Control Block (ECB)

The event control block provides you with a completion code that describes whether the channel program was completed with or without error. A WAIT macro instruction, which can be used to synchronize I/O operations with the problem program, must identify the event control block. You must define the event control block and specify its address in the input/output block.

Data Control Block (DCB)

The data control block provides the system with information about the characteristics and processing requirements of a data set to be read or written by the channel program. A data control block must be produced by a DCB macro instruction that includes parameters for EXCP. If appendages are not being used, a short DCB is constructed. Such a DCB does not support reduced error recovery. You specify the address of the data control block in the input/output block.

Data Extent Block (DEB)

The data extent block contains one or more extent entries for the associated data set, as well as other control information. An extent defines all or part of the physical boundaries on an I/O device occupied by, or reserved for, a particular data set. Each extent entry contains the address of a unit control block (UCB), which provides information about the type and location of an I/O device. More than one extent entry can contain the same UCB address. For all I/O devices supported by the operating system, the data extent block is produced during execution of the OPEN macro instruction for the data control block. The system places the address of the data extent block into the data control block.

CHANNEL PROGRAM EXECUTION

This section explains how the system uses your channel program and control blocks after you issue EXCP.

INITIATION OF THE CHANNEL PROGRAM

By issuing EXCP, you request the execution of the channel program specified in the input/output block. The I/O process validates the request by checking certain fields of the control blocks associated with this request. If the I/O process detects invalid information in a control block, it initiates abnormal termination procedures.

The EXCP processor gets:

- The address of the data control block from the input/output block
- The address of the data extent block from the data control block
- The address of the unit control block from the data extent block

It places the IOB, TCB, DEB, and UCB addresses and other information about the channel program into an area called a request queue element (RQE). (Unless you are providing appendage routines (described under "Appendages" on page 72) you should not be concerned with the contents of RQEs.)

If you have provided a start I/O (SIO) appendage, the EXCP processor now passes control to it. The return address from the SIO appendage determines whether the EXCP processor must:

- Execute the I/O operation normally, or
- Skip the I/O operation.

For a description of the SIO appendage and its linkage to the EXCP processor, see "Appendages" on page 72.

If you are issuing EXCP from a V=V address space, the channel program you construct contains virtual addresses. Because channel subsystems cannot use virtual addresses, the EXCP processor must:

- Translate your virtual channel program into one that uses only real addresses.
- Fix in real storage the pages used as I/O areas for the data transfer operations specified in your channel program.

The EXCP processor builds the translated (real) channel program in a portion of real storage.

For direct access devices, specify the seek address in the input/output block. The I/O supervisor constructs a command chain to issue the seek, set file mask specified in the data extent block, and pass control to your real channel program.

If your channel program begins with a locate-record command, the I/O process builds a define-extent command and passes control to your real channel program. (You cannot issue the initial seek, set file mask, or define extent. The file mask is set to prohibit seek-cylinder commands, or, if space is allocated by tracks, seek-head commands. If the data set is open for INPUT, write commands are also prohibited.)

For a magnetic tape device, the I/O supervisor constructs a command chain to set the mode specified in the data extent block and passes control to your real channel program. (You cannot set the mode yourself.)

If the I/O device is other than a direct access device or a magnetic tape device, the I/O supervisor then places the address of the starting CCW of the channel program into the channel address word (CAW) and issues a start I/O (SIO) instruction.

MODIFICATION OF A CHANNEL PROGRAM DURING EXECUTION

Any problem program that modifies an active channel program with CPU instructions or with data read in by an I/O operation must be run in a V=R address space. It cannot run in a V=V address space because of the channel program translation performed by the I/O supervisor. (In a V=V address space, an attempt to modify an active channel program affects only the virtual image of the channel program, not the real channel program being executed by the channel subsystem.)

A program of this type can be changed to run in a V=V address space by issuing another EXCP macro for the modified portion of the channel program.

COMPLETION OF EXECUTION

The system considers the channel program completed when it receives an indication of a channel-end condition in the channel status word. Unless a channel-end or abnormal-end appendage directs otherwise, the request queue element for the channel program is made available, and a completion code is placed into the event control block. The completion code indicates whether errors are associated with channel end. If device end occurs simultaneously with channel end, errors associated with device end (that is, unit exception or unit check) are also accounted for.

If device end occurs after channel end, and an error is associated with device end, the completion code in the event control block does not indicate the error. However, the status of the unit and channel is saved by the I/O supervisor for the device, and the UCB is marked as intercepted. The input/output block for the next request directed to the I/O device is also marked as intercepted. The error is assumed to be permanent, and the completion code in the event control block for the intercepted request indicates interception. The DCBIFLGS field of the data control block is also flagged to indicate a permanent error. Note that if a write-tape-mark or erase-long-gap CCW is the last or only CCW in your channel program, the I/O process will not attempt recovery procedures for device end errors. In these circumstances, command chaining a NOP CCW to your write-tape-mark or erase-long-gap CCW ensures initiation of device-end error recovery procedures.

To be prepared for device-end errors, you should be familiar with device characteristics that can cause such errors. After one of your channel programs has terminated, you should not release buffer space until you have determined that your next request for the device has not been intercepted. You may reissue an intercepted request.

INTERRUPTION HANDLING AND ERROR RECOVERY PROCEDURES

An I/O interruption allows the processor to respond to signals from an I/O device which indicate either termination of a phase of I/O operations or external action on the device. A complete explanation of I/O interruptions is contained in IBM System/370 Principles of Operation. For descriptions of interruption by specific devices, refer to IBM publications for each device.

If error conditions are associated with an interruption, the I/O supervisor schedules the appropriate device-dependent error routine. The channel subsystem is then restarted with another request that is not related to the channel program in error. (The following paragraphs discuss "related" channel programs.) If the error recovery procedures fail to correct the error, the system places ones in the first two bit positions of the DCBIFLGS field of the data control block. You are informed of the error by an error code in the event control block.

If a channel program depends on the successful completion of a previous channel program—as when one channel program retrieves data to be used in building another—the previous channel program is called a "related" request. Such a request must be identified to the EXCP processor. To find out how, see "Input/Output Block Fields" on page 95.

If a permanent error occurs in the channel program of a related request, the EXCP processor removes the request queue elements for all dependent channel programs from their queue and makes them available.

The related request queue (RRQ) reflects the order in which request queue elements are removed from their queue.

For all requests dependent on the channel program in error, the system places completion codes into the event control blocks. The DCBIFLGS field of the data control block is also flagged. Any requests for a data control block with error flags are posted complete without execution. To reissue requests dependent on the channel program in error, you must reset the first two bits of the DCBIFLGS field of the data control block to zeros. You then reissue EXCP for each channel program desired.

With the 3800, a cancel key or a system-restart-required paper jam causes both a lost data indicator to be set in DCBIFLGS and a lost page count and channel page identifier to be stored in the UCB extension. (See Debugging Handbook and IBM 3800 Printing Subsystem Programmer's Guide.)

APPENDAGES

An appendage is a programmer-written routine that provides additional control over I/O operations. By using appendages, you can examine the status of I/O operations and determine the actions to be taken for various conditions. An appendage may receive control when one of the following occurs:

- EXCP SVC
- Program controlled interrupt
- End of extent
- Channel end
- Abnormal end

Appendages get control in supervisor state, receiving the following pointers from the EXCP processor:

- Register 1: Points to the request queue element for the channel program.
- Register 2: Points to the input/output block (IOB).
- Register 3: Points to the data extent block (DEB).
- Register 4: Points to the data control block (DCB).
- Register 6: Points to the seek address if control is given to an end-of-extent appendage.
- Register 7: Points to the unit control block (UCB).
- Register 13: Points to a 16-word area you can use to save input registers or data.

- Register 14: Points to the location in the EXCP processor to which control is to be returned after execution of an appendage. When returning control to the EXCP processor, you may use displacements from the return address in register 14. Allowable displacements are summarized in Figure 20 and described later for each appendage.

- Register 15: Points to the entry point of the appendage.

The processing done by appendages is subject to these requirements and restrictions:

- Register 9, if used, must be set to binary zeros before control is returned to the system. All other registers, except those indicated in the descriptions of each appendage, must be saved and restored if they are used. Figure 20 summarizes register conventions.

- No SVC instructions or instructions that change the status of the system (for example, WTO, LPSW, or any privileged instructions) can be issued.

- Loops that test for the completion of I/O operations must not be used.

- Storage used by the I/O supervisor or EXCP processor must not be altered.

The types of appendages are described in the following sections, with explanations of when they are created, how they return control to the system, and which registers they may use without saving and restoring their contents.

Appendage	Entry Point	Returns		Available Work Registers[1]
EOE	Reg 15	Reg 14 + 0	Return	Reg. 10, 11, 12, and 13
		Reg 14 + 4	Skip	
		Reg 14 + 8	Try Again	
SIO	Reg 15	Reg 14 + 0	Normal	Reg. 10, 11, and 13
		Reg 14 + 4	Skip	
PCI	Reg 15	Reg 14 + 0	Normal	Reg. 10, 11, 12, and 13
CHE	Reg 15	Reg 14 + 0	Normal	Reg. 10, 11, 12, and 13
		Reg 14 + 4	Skip	
		Reg 14 + 8	Re-EXCP	
		Reg 14 + 12	By-Pass	
ABE	Reg 15	Reg 14 + 0	Normal	Reg. 10, 11, 12, and 13
		Reg 14 + 4	Skip	
		Reg 14 + 8	Re-EXCP	
		Reg 14 + 12	By-Pass	

[1] Certain register conventions for passing parameters from appendages to the EXCP processor must be followed. These conventions are described in the individual appendage descriptions.

Figure 20. Entry Points, Returns, and Available Work Registers for Appendages

START-I/O (SIO) APPENDAGE

Unless an error recovery procedure is in control, the EXCP processor passes control to the SIO appendage just before the EXCP processor translates your channel program.

Optional return vectors give the I/O requestor the following choices:

Reg. 14 + 0
Normal return. Normal channel program translation and initiation of I/O.

Reg. 14 + 4
Skip the I/O operation. The channel program is not posted complete, but the request queue element is made available. You may post the channel program as follows:

1. Save necessary registers.

2. Put the address of the post routine (found at CVTOPT01 in the communications vector table) in register 15.

3. Place the ECB address from the IOB in register 11.

4. Set the completion code in register 10. These are the four bytes of an ECB.

5. Go to the post routine pointed to by the CVT, using BALR 14,15.

PROGRAM-CONTROLLED INTERRUPTION (PCI) APPENDAGE

This appendage is entered at least once if the channel finds one or more PCI bits on in a channel program, and may be entered as many times as the channel finds PCI bits on. Before the appendage is entered, the contents of the channel status word are placed in the "channel status word" field of the input/output block.

A PCI appendage will be reentered if an error recovery procedure is retrying a channel program in which a PCI bit is on. The IOB error flag is set when the error recovery procedure is in control (IOBFLAG1 = X'20'). (For special PCI conditions encountered with command retry, see "Channel Programming Notes" on page 79.)

To post the channel program from a PCI appendage, the procedure described for the start-I/O appendage is used if the step is running ADDRSPC=VIRT or an authorized program is running V=R. If the step is running ADDRSPC=REAL and an authorized program issued the EXCP request, or SVC 114(EXCPVR) was issued, the PCI appendage uses real storage addresses and the following procedure is used to post the channel program from the PCI appendage.

1. Put the completion code in register 10 and place X'80' in the high-order byte to indicate the key is in register 0 (step 5).

2. Put X'80' in the high-order byte of register 11 and the address of the ECB in the low-order bytes.

3. Put X'80' in the high-order byte of register 12 and the address of a BR 14 instruction in the low-order bytes. This BR 14 must be in storage addressable from any address space (for example, CVTBRET). Note that registers 9 and 14 only are restored when you use this option.

4. Put the address of the ASCB in register 13.

   The next two paragraphs describe how to obtain the ASCB address and are followed by sample instructions to illustrate the procedure.

   Get the SRB address associated with the I/O operation from the RQE field, RQESRB (the RQE address was in register 1 when the appendage was given control). Get the IOSB address from SRBPARM. From that IOSB, get the identifier field, IOSASID. Multiply IOSASID by 4.

Get the pointer to the ASVT (address space vector table) found at CVTASVT. The address of the ASCB can be found in the ASVT, using the field ASVTENTY-4 indexed by the value calculated in the above paragraph.

```
USING    RQE,1
L        Y,RQESRB
USING    SRBSECT,Y
LH       Y,SRBPARM
USING    IOSB,Y
LH       Y,IOSASID
SLA      Y,2
L        X,16
USING    CVT,X
L        X,CVTASVT
USING    ASVT,X
L        13,ASVTENTY-4(Y)
```

Note:
X and Y are work registers.

5. Put the requestor's key in register 0.

6. Put the address of the post routine (found at CVTOPT01 in the communications vector table) in register 15.

7. Go to the post routine using BALR 14,15. Upon return, only registers 9 and 14 are valid. For more information on the POST routine, see System Programming Library: Supervisor Services and Macro Instructions.

This procedure can be used even if the PCI appendage uses virtual storage addresses, but performance may be slightly slower.

To return control to the EXCP processor for normal interruption processing, use the return address in register 14.

END-OF-EXTENT (EOE) APPENDAGE

This appendage is entered when the seek address specified in the input/output block is outside the allocated extent limits indicated in the data extent block.

If you use the return address in register 14 to return control to the system, the abnormal-end appendage is entered. An end-of-extent error code (X'42') is placed in the "ECB code" field of the input/output block for subsequent posting in the ECB.

You may use the following optional return addresses:

- Contents of register 14 plus 4: The channel program is posted complete; its request element is returned to the available queue.

- Contents of register 14 plus 8: The request is tried again.

You may use registers 10 through 13 in an end-of-extent appendage without saving and restoring their contents.

Note: If an end-of-cylinder or file-protect condition occurs, the EXCP processor updates the seek address to the next higher cylinder or track address, and re-executes the request. If the new seek address is within the data set's extent, the request is executed; if the new seek address is not within the data set's extent, the end-of-extent appendage is entered. If you wish to try the request in the next extent, you must move the new seek address to the location pointed to by register 6.

If a file protect is caused by a full seek (command code=07) embedded within a channel program, the request is flagged as a permanent error, and the abnormal end appendage is entered.

CHANNEL-END (CHE) APPENDAGE

This appendage is entered when a channel end (CHE), unit exception (UEX) with or without channel end, or channel end with wrong length record (WLR) occurs without any other abnormal-end conditions.

If you use the return address in register 14 to return control to the EXCP processor, the channel program is posted complete, and its request element is made available. In the case of unit exception or wrong length record, the error recovery procedure is performed before the channel program is posted complete, and the IOBEX flag (X'04') in IOBFLAG1 is set on. The CSW status may be obtained from the IOBCSW field.

If the appendage takes care of the wrong length record and/or unit exception, it may turn off the IOBEX (X'04') flag in IOBFLAG1 and return normally. The event will then be posted complete (completion code X'7F' under normal conditions, taken from the high-order byte of the IOBECBCC field). If the appendage returns normally without resetting the IOBEX flag to zero, the request will be routed to the associated device error recovery procedure (ERP), and then the abnormal-end appendage will be entered with the completion code in IOBECBCC set to X'41' if the ERP could not correct the error. (See Step 1 of "Abnormal-End (ABE) Appendage.")

You may use the following optional return addresses:

- Contents of register 14 plus 4: The channel program is not posted complete, but its request element is made available. You may post the channel program by using the calling sequence described under the start-I/O appendage. This is especially useful if you wish to post an ECB other than the ECB in the input/output block.

- Contents of register 14 plus 8: The channel program is not posted complete, and its request element is placed back on the request queue so that the I/O operation can be retried. For correct re-execution of the channel program, you must reinitialize the IOBFLAG1, IOBFLAG2, and IOBFLAG3 fields of the input/output block and set the "Error Counts" field to zero. As an added precaution, the IOBSENS0, IOBSENS1, and IOBCSW fields should be cleared.

- Contents of register 14 plus 12: The channel program is not posted complete, and its request element is not made available. (This return must be used if, and only if, the appendage has passed the RQE to the exit effector for use in scheduling an asynchronous routine. For information on the exit effector, refer to System Programming Library: Supervisor.)

You may use registers 10 through 13 in a channel-end appendage without saving and restoring their contents.

ABNORMAL-END (ABE) APPENDAGE

This appendage may be entered on abnormal conditions, such as: unit check, unit exception, wrong length indication, program check, protection check, channel data check, channel control check, interface control check, chaining check, out-of-extent error, and intercept condition (that is, device end error). It may also be entered when an EXCP is issued for a request queue element that has already been purged.

1. When this appendage is entered because of a unit exception and/or wrong length record indication, IOBECBCC is set to X'41'. For further information on these conditions, see "Channel-End (CHE) Appendage" on page 76.

2. When the appendage is entered because of an out-of-extent error, the IOBECBCC is set to X'42'.

3. When this appendage is entered with IOBECBCC set to X'4B', it is because of:

   a. The tape error recovery procedure (ERP) encountering an unexpected load point, or

   b. The tape error recovery procedure (ERP) finding zeros in the command address field of the CSW.

4. When the appendage is first entered because of an intercept condition, the IOBECBCC is set to X'7E'. If it is then determined that the error condition is permanent, the appendage will be entered a second time with the IOBECBCC set to X'44'. The intercept condition signals that an error was detected at device end after channel end on the previous request.

5. When the appendage is entered because of an EXCP being issued to an already purged request queue element, this request will enter the abnormal end appendage with the IOBECBCC set to X'48'. This applies only to related requests.

6. If the appendage is entered with IOBECBCC set to X'7F', it may be because of a unit check, program check, protection check, channel data check, channel control check, interface control check, or chaining check. If the IOBECBCC is X'7F', it is the first detection of an error in the associated channel program. If the IOBEX flag (bit 5 of the IOBFLAG1) is on, the IOBECBCC field will contain a 41, 42, 48, 4B, or 4F in hexadecimal, indicating a permanent I/O error.

To determine if an error is permanent, you should check the IOBECBCC field of the IOB. To determine the type of error, check the channel status word field and the sense information in the IOB. However, when the IOBECBCC is X'42', X'48', or X'4F', these fields are not applicable. For X'44', the CSW is applicable, but the sense is valid only if the unit check bit is set.

If you use the return address in register 14 to return control to the system, the channel program is posted complete, and its request element is made available. You may use the following optional return addresses:

- Contents of register 14 plus 4: The channel program is not posted complete, but its request element is made available. You may post the channel program by using the calling sequence described under the start-I/O appendage.

- Contents of register 14 plus 8: The channel program is not posted complete, and its request element is placed back on the request queue so that the request can be retried. For correct reexecution of the channel program, you must reinitialize the IOBFLAG1, IOBFLAG2, and IOBFLAG3 fields of the input/output block and set the IOBERRCT field to zero. As an added precaution, the IOBSENS0, IOBSENS1, and IOBCSW fields should be cleared.

- Contents of register 14 plus 12: The channel program is not posted complete, and its request element is not made available. (This return must be used if, and only if, the appendage has passed the RQE to the exit effector for use in scheduling an asynchronous routine.)

You may use registers 10 through 13 in an abnormal-end appendage without saving and restoring their contents.

MAKING YOUR APPENDAGES PART OF THE SYSTEM

Before your appendages can be executed, they must become members of either the SYS1.LPALIB or SYS1.SVCLIB data set. There are two ways to put appendages into SYS1.LPALIB or SYS1.SVCLIB: they can be included at system generation using the DATASET macro instruction (a full explanation appears in System Generation Reference), or they can be link-edited into SYS1.LPALIB or SYS1.SVCLIB after the system has been generated. Each appendage must have an 8-character member name, the first six characters being IGG019, the last two being anything in the range from WA to Z9. Note, however, if your program runs in a V=R address space and uses a PCI appendage, the PCI appendage and any appendage that the PCI appendage refers to must be placed in either SYS1.SVCLIB or the fixed link pack area (LPA). For information on providing a list of programs to be fixed in storage, see System Programming Library: Initialization and Tuning.

THE AUTHORIZED APPENDAGE LIST (IEAAPP00)

If an "unauthorized" program opens a DCB to be used with an EXCP macro instruction, the names of any appendages associated with the DCB must be listed in the IEAAPP00 member of SYS1.PARMLIB. (An "unauthorized" program is one that runs in a protection key greater than 7 and has not been marked as authorized by the Authorized Program Facility.)

If your appendages were put in SYS1.LPALIB or SYS1.SVCLIB at system generation, their names are automatically put in IEAAPP00. If your appendages were added to SYS1.LPALIB or SYS1.SVCLIB after system generation, you can add IEAAPP00 to SYS1.PARMLIB and put the names of the appendages in it in one job step with the IEBUPDTE utility.

Here is an example of JCL statements and IEBUPDTE input that will add IEAAPP00 to SYS1.PARMLIB and put the names of one EOE appendage, two SIO appendages, two CHE appendages, and one ABE appendage in IEAAPP00:

```
//              JOB       ...
//              EXEC      PGM=IEBUPDTE
//SYSPRINT      DD        SYSOUT=A
//SYSUT2        DD        DSN=SYS1.PARMLIB,DISP=OLD
//SYSIN         DD        *
./              ADD       NAME=IEAAPP00,LIST=ALL
EOEAPP WA,
SIOAPP X1,X2,
CHEAPP Z3,Z4,
ABEAPP Z2
/*
```

Note the following about the IEBUPDTE input:

- The type of appendage is identified by six characters that begin in column 1. EOEAPP identifies an EOE appendage, SIOAPP an SIO appendage, CHEAPP a CHE appendage, and ABEAPP an ABE appendage. (The PCI appendage identifier, PCIAPP, is not shown, because the example adds no PCI appendage name to IEAAPP00.)

- Only the last two characters in an appendage's name are specified, beginning in column 8.

- Each statement that identifies one or more appendage names ends in a comma, except the last statement.

You can also use IEBUPDTE to add appendage names later or delete appendage names. Here is an example of JCL statements and IEBUPDTE input that adds the names of a PCI and an ABE appendage to the IEAAPP00 appendage list that was created in the preceding example, and deletes the name of an SIO appendage from that list:

```
//                JOB
//                EXEC     PGM=IEBUPDTE
//SYSPRINT        DD       SYSOUT=A
//SYSUT2          DD       DSN=SYS1.PARMLIB,DISP=OLD
//SYSIN           DD       *
./                REPL     NAME=IEAPP00,LIST=ALL
PCIAPP Y1,
EOEAPP WA,
SIOAPP X1,X2,
CHEAPP Z3,Z4,
ABEAPP Z2,Z4
/*
```

Note the following about the IEBUPDTE input:

- The command to IEBUPDTE in this case is REPL (replace).
- All the appendage names that are to remain in IEAAPP00 are repeated.
- IGG019Z4 is both a CHE and an ABE appendage.

CHANNEL PROGRAMMING NOTES

Command retry is a function of the channel supporting the 2305, 3330/3333, 3340/3344, 3350, 3375, and 3380 direct access devices. When the channel subsystem receives a retry request, it repeats the execution of the CCW requiring no additional input/output interrupts. For example, a control unit may initiate a retry procedure to recover from a transient error.

A command retry during the execution of a channel program may cause any of the following conditions to be detected by the initiating program:

- Modifying CCWs: A CCW used in a channel program must not be modified before the CCW operation has been successfully completed. Without the command retry function, a command was fetched only once from storage by a channel. Therefore, a program could determine through condition codes or program controlled interruptions (PCI) that a CCW had been fetched and accepted by the channel. This permitted the CCW to be modified before reexecution. With the command retry function, this cannot be done, because the channel will fetch the CCW from storage again on a command retry sequence. In the case of data chaining, the channel will retry commands starting with the first CCW in the data chain.
- Program Controlled Interrupts: A CCW containing a PCI flag may cause multiple program controlled interrupts to occur. This happens if the PCI-flagged CCW was retried during a command retry procedure, and a PCI could be generated each time the CCW is reexecuted.
- Residual Count: If a channel program is prematurely terminated during the retry of a command, the residual count in the channel status word (CSW) will not necessarily indicate how much storage was used. For example, if the control unit detects a "wrong length record" error condition, an erroneous residual count is stored in the CSW until the command retry is successful. When the retry is successful, the residual in the CSW reflects the correct length of the data transfer.

- Command Address: When data chaining with command retry, the CSW may not indicate how many CCWs have been executed at the time of a PCI. For example:

CCW#	Channel Program
1	Read, data chain
2	Read, data chain
3	Read, data chain, PCI
4	Read, command chain

  In this example, assume that the control unit signals command retry on Read #3 and the CPU accepts the PCI after the channel resets the command address to Read #1 because of command retry. The CSW stored for the PCI will contain the command address of Read #1, when actually the channel has progressed to Read #3.

- Testing Buffer Contents on Data Read: Any program that tests a buffer to determine when a CCW has been executed and continues to execute based on this data may get incorrect results if an error is detected and the CCW is retried.

MACRO SPECIFICATIONS FOR USE WITH EXCP

If you are using the EXCP macro instruction, you must also use DCB, OPEN, CLOSE, and, in some cases, the EOV macro instruction. The parameters of these macro instructions and the EXCP macro instructions are explained here. A diagram of the data control block is included with the description of the DCB macro instruction.

DCB—DEFINE DATA CONTROL BLOCK FOR EXCP

The EXCP form of the DCB macro instruction produces a data control block that can be used with the EXCP macro instruction. You must issue a DCB macro instruction for each data set to be processed by your channel programs. Notation conventions and format illustrations of the DCB macro instruction are given in Data Management Macro Instructions. DCB parameters that apply to EXCP may be divided into four categories, depending on the following portions of the data control block that are generated when they are specified:

- Foundation block. This portion is required and is always 12 bytes in length. You must specify two of the parameters in this category.

- EXCP interface. This portion is optional. If you specify any parameter in this category, 20 bytes are generated.

- Foundation block extension and common interface. This portion is optional and is always 20 bytes in length If this portion is generated, the device-dependent portion is also generated.

- Device dependent. This portion is optional and is generated only if the foundation block extension and common interface portion is generated. Its size ranges from 4 to 20 bytes, depending on specifications in the DEVD parameter. However, if you do not specify the DEVD parameter (and the foundation extension and common interface portion is generated), the maximum 20 bytes for this portion are generated.

Some of the procedures performed by the system when the data control block is opened and closed (such as writing file marks for output data sets on direct access volumes) require information from optional data control block fields. You should make sure that the data control block is large enough to provide all information necessary for the procedures you want the system to handle.

Figure 21 on page 82 shows the relative position of each portion of an opened data control block. The fields corresponding to each parameter of the DCB macro instruction are also designated, with the exception of DDNAME, which is not included in a data control block that has been opened. The fields identified in parentheses represent system information that is not associated with parameters of the DCB macro instruction.

Sources of information for data control block fields other than the DCB macro instruction are data definition (DD) statements, data set labels, and data control block modification routines. You may use any of these sources to specify DCB parameters. However, if a particular portion of the data control block is not generated by the DCB macro instruction, the system does not accept information intended for that portion from any alternative source.

You may provide symbolic names for the fields in one or more EXCP DCBs by coding a DCBD macro to generate a dummy control section (DSECT). To map the common interface, foundation block extension, and foundation block, you code DSORG=XE. To map the foundation block and EXCP interface, you code DSORG=XA. You may code DSORG=(XA,XE) to map both. For further information, see Data Management Macro Instructions.

Foundation Block Parameters

DDNAME=symbol
The name of the data definition (DD) statement that describes the data set to be processed. This parameter must be given.

MACRF=(E)
The EXCP macro instruction is to be used in processing the data set. This operand must be coded.

REPOS={Y|N}
Magnetic tape volumes: This parameter indicates to the dynamic device reconfiguration (DDR) routine whether the user is keeping an accurate block count. If the user is keeping an accurate block count, the DDR routine can attempt to swap the volume. (You must maintain the block count in the DCBBLKCT field.)

Y—The user is keeping an accurate block count and the DDR routine can attempt to swap the volume.

N—The block count is unreliable and the DDR routine cannot and will not attempt to swap the volume.

If the operand is omitted, N is assumed.

EXCP Interface Parameters

EOEA=symbol
2-byte identification of an EOE appendage that you have entered into SYS1.LPALIB or SYS1.SVCLIB.

PCIA=symbol
2-byte identification of a PCI appendage that you have entered into SYS1.LPALIB or SYS1.SVCLIB.

SIOA=symbol
2-byte identification of a SIO appendage that you have entered into SYS1.LPALIB or SYS1.SVCLIB.

CENDA=symbol
2-byte identification of a CHE appendage that you have entered into SYS1.LPALIB or SYS1.SVCLIB.

0	The device-dependent portion of the data control block varies in length and format according to specifications in the DSORG and DEVD parameters. Illustrations of this portion for each device type are included in the description of the DEVD parameter.		Device Dependent
20	BUFNO	BUFCB	Common Interface
24	BUFL	DSORG	
28	IOBAD		
32	BFTEK, BFALN	EODAD	Foundation Block Extension
36	RECFM	EXLST	
40	(TIOT)	MACRF	Foundation Block
44	(IFLGS)	(DEB Address)	
48	(OFLGS)	Reserved	
52	OPTCD	Reserved	EXCP Interface
56	Reserved		
60	EOEA	PCIA	
64	SIOA	CENDA	
68	XENDA	Reserved	

Figure 21. Data Control Block Format for EXCP (After OPEN)

XENDA=symbol
2-byte identification of an ABE appendage that you have entered into SYS1.LPALIB or SYS1.SVCLIB.

OPTCD=Z
indicates that, for magnetic tape (input only), a reduced error recovery procedure (5 reads only) will occur when a data check is encountered. It should be specified only when the tape is known to contain errors and the application does not require that all records be processed. Its proper use would include error frequency analysis in the SYNAD routine. Specification of this parameter will also cause generation of a foundation block extension. This parameter is ignored unless it was selected at system generation.

IMSK=value

Any specification indicates that the system will not use IBM-supplied error routines.

Foundation Block Extension and Common Interface Parameters

EXLST=address

the address of an exit list that you have written for exception conditions. The format of this exit list is given in Data Management Services.

EODAD=address

the address of your end-of-data-set routine for input data sets. If this routine is not available when it is required, the task is abnormally terminated.

DSORG={PS|PO|DA|IS}

the data set organization (one of the following codes). Each code indicates that the format of the device-dependent portion of the data control block is to be similar to that generated for a particular access method:

Code	DCB Format for
PS	QSAM or BSAM
PO	BPAM
DA	BDAM
IS	QISAM or BISAM

For direct access devices, if you specify PS or PO, you must maintain the following fields of the device-dependent portion of the data control block so that the system can write a file mark for output data sets:

- The track balance (DCBTRBAL) field, which contains a 2-byte binary number that indicates the remaining number of bytes on the current track. This number can be obtained from the system track algorithm routine.
- The full disk address (DCBFDAD) field, which indicates the location of the current record. The address is in the form MBBCCHHR.

These fields are written into the format-1 DSCB and are used by Open routines for staging MSS data sets. Staging is done only up through the last cylinder specified by these fields if the data set is reopened for OUTPUT, INOUT, OUTIN, OUTINX, or EXTEND.

If you specify PO for a direct access device, the DCBDIRCT field will not be updated. Therefore, you should be careful when using EXCP with the STOW macro.

IOBAD=address

the address of an input/output block (IOB). If a pointer to the current IOB is not required, you may use this field for any purpose.

The following parameters are not used by the EXCP routines. They provide additional information that the system will store for later use by access methods that read or update the data set.

RECFM=code

the record format of the data set. Record format codes are given in Data Management Macro Instructions. When writing a data set to be read later, RECFM, LRECL, and BLKSIZE should be specified to identify the data set attributes. LRECL and BLKSIZE can only be specified in a DD statement, because these fields do not exist in a DCB used by EXCP.

BFTEK={S|E}
the buffer technique, either simple or exchange.

BFALN={F|D}
the word boundary alignment of each buffer, either fullword or doubleword.

BUFL=length
the length in bytes of each buffer; the maximum length is 32,767.

BUFNO=number
the number of buffers assigned to the associated data set; the maximum number is 255.

BUFCB=address
the address of a buffer pool control block, that is, the 8-byte field preceding the buffers in a buffer pool.

Device-Dependent Parameters

DEVD=code
the device on which the data set may reside. The codes are listed in order of descending space requirements for the data control block:

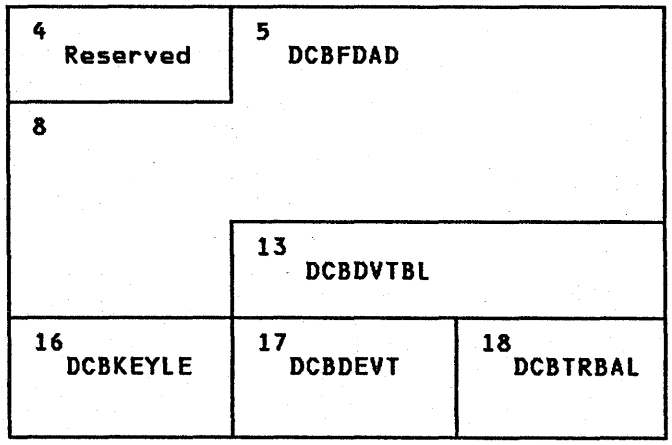
Code	**Device**
DA	Direct access
TA	Magnetic tape
PT	Paper tape
PR	Printer
PC	Card punch
RD	Card reader

Note: For MSS virtual volumes, DA should be used.

If you do not wish to select a specific device until job setup time, you should specify the device type requiring the largest area; that is, DEVD=DA.

The following diagrams illustrate the device-dependent portion of the data control block for each combination of device type specified in the DEVD parameter and data set organization specified in the DSORG parameter. Fields that correspond to device-dependent parameters in addition to DEVD are indicated by the parameter name. For special services, you may have to maintain the fields shown in parentheses. The special services are explained in the note that follows the diagram.

Device-dependent portion of data control block when DEVD=DA and DSORG=PS:



For output data sets, the system uses the contents of the full disk address (DCBFDAD) field plus one to write a file mark when the data control block is closed, provided the track balance (DCBTRBAL) field indicates that space is available. If DCBTRBAL is less than 8, the file mark is written on the next sequential track. You must maintain the contents of these two fields yourself if the system is to write a file mark. OPEN will initialize DCBDVTBL and DCBDEVT.

Device-dependent portion of data control block when DEVD=DA and DSORG=DA:

16 DCBKEYLE	18 Reserved

Device-dependent portion of data control block when DEVD=TA and DSORG=PS:

12 DCBBLKCT			
16 DCBTRTCH	17 Reserved	18 DCBDEN	19 Reserved

The system uses the contents of the block count (DCBBLKCT) field to write the block count in trailer labels when the data control block is closed or when the EOV macro instruction is issued. You must maintain the contents of this field yourself if the system is to have the correct block count. (**Note**: The I/O supervisor increments this field by the contents of the IOBINCAM field of the IOB at the completion of each I/O request.)

When using EXCP to process a tape data set open at a checkpoint, you must be careful to maintain the correct count; otherwise, the system may position the data set incorrectly when restart occurs. If REPOS=Y, the count must be maintained by you for repositioning during dynamic device reconfiguration.

Device-dependent portion of data control block when DEVD=PT and DSORG=PS:

16 DCBCODE	18 Reserved

Device-dependent portion of data control block when DEVD=PR and DSORG=PS:

16 DCBPRTSP	18 Reserved

Device-dependent portion of data control block when DEVD=PC or RD and DSORG=PS:

16 DCBMODE,DCBSTACK	18 Reserved

The following DCB operands pertain to specific devices and may be specified only when the DEVD parameter is specified.

KEYLEN=length
for direct access devices, the length in bytes of the key of a physical record, with a maximum value of 255. When a block is read or written, the number of bytes transmitted is the key length plus the record length.

CODE=value
for paper tape, the code in which records are punched:

Value	**Code**
I	IBM BCD
F	Friden
B	Burroughs
C	National Cash Register
A	ASCII
T	Teletype (trademark of Teletype Corporation)
N	No conversion (format-F records only)

If this parameter is omitted, N is assumed.

DEN=value
for magnetic tape, the tape recording density in bits per inch:

Value:	**Density:** **7-track tape device**	**9-track tape device**
1	556	—
2	800	800(NRZI)
3	—	1600(PE)
4	—	6250(GCR)

NRZI—Non-return-to-zero change to ones recording
PE—phase encoded recording
GCR—group coded recording

If this parameter is omitted, the highest density available on the device is assumed.

TRTCH=value
for 7-track magnetic tape, the tape recording technique:

Value	**Tape Recording Technique**
C	Data conversion feature is available.
E	Even parity is used. (If omitted, odd parity is assumed.)
T	BCDIC to EBCDIC translation is required.

MODE=value
for a card reader or punch, the mode of operation. Either C (column binary mode) or E (EBCDIC code) may be specified.

STACK=value
for a card punch or card reader, the stacker bin to receive cards, either 1 or 2.

PRTSP=value
for a printer, the line spacing, either 0, 1, 2, or 3.

DSORG Parameter of the DCBD Macro

In addition to the operands described in Data Management Macro Instructions for the DSORG parameter of the DCBD macro, you may specify the following operands.

DSORG=

XA specifies a DCB with the EXCP interface section (including appendage names)

XE specifies a DCB with the foundation block extension

OPEN—INITIALIZE DATA CONTROL BLOCK

The OPEN macro instruction initializes one or more data control blocks so that their associated data sets can be processed. You must issue OPEN for all data control blocks that are to be used by your channel programs. (A dummy data set may not be opened for EXCP.) Some of the procedures performed when OPEN is executed are:

- Reading in the JFCB (job file control block), unless the TYPE=J option of the macro instruction was coded
- Construction of the data extent block (DEB)
- Transfer of information from the JFCB and data set labels to the DCB
- Verification or creation of standard labels
- Tape positioning
- Loading of your appendage routines

The parameters of the OPEN macro instruction are:

[symbol]	OPEN	(dcb address ,[(options)],...)

dcb address—A-type address or (2-12)

the address of the data control block to be initialized. (More than one data control block may be specified.)

option1

the intended method of I/O processing of the data set. You may specify this parameter as either INPUT, RDBACK, OUTPUT, or EXTEND. For magnetic tape, label processing for each of these when OPEN is executed is as follows:

INPUT	Header labels are verified.
RDBACK	Trailer labels are verified.
OUTPUT	Header labels are created.
EXTEND	Header labels are created.

If this parameter is omitted, INPUT is assumed.

option2

the volume disposition that is to be provided when volume switching occurs. The operand values and meanings are as follows:

REREAD Reposition the volume to process the data set again.

LEAVE No additional positioning is performed at end-of-volume processing.

DISP Specifies that a tape volume is to be disposed of in the manner implied by the DD statement associated with the data set. Direct access volume positioning and disposition are not affected by this parameter of the OPEN macro instruction. There are several dispositions that can be specified in the DISP parameter of the DD statement:

DISP=PASS, DELETE, KEEP, CATLG, or UNCATLG. Only DISP=PASS has significance at the time an end-of-volume condition is encountered. The end-of-volume condition may result from the issuance of an FEOV macro instruction or may be the result of reaching the end of a volume.

If DISP=PASS was coded in the DD statement, the tape will be spaced forward to the logical end of the data set on the current volume.

If a DISP option other than DISP=PASS is coded on the DD statement, the action taken when an end-of-volume condition occurs depends (1) on how many tape units are allocated to the data set and (2) on how many volumes are specified for the data set in the DD statement. This is determined by the UNIT= and VOLUME= operands of the DD statement associated with the data set. If the number of volumes is greater than the number of units allocated, the current volume will be rewound and unloaded. If the number of volumes is less than or equal to the number of units, the current volume is merely rewound.

If you intend to process a multivolume direct data set, you must cause open routines to build a data extent block for each volume and issue mount messages for them. This can be done by reading in the JFCB with a RDJFCB macro instruction and opening each volume of the data set. The following code illustrates the procedure:

```
         RDJFCB      DCB1                 READS IN THE JFCB
         SR          R3,R3                CLEARS REG 3; IT WILL
*                                         HOLD COUNT OF VOLS TO
*                                         BE OPENED
         IC          R3,JFCBNVOL          PUTS # OF VOLS
*                                         IN REG 3
         LA          R4,DCB1              R4 POINTS TO DCB FOR
*                                         VOL TO BE OPENED
         LA          R5,1                 PUTS SEQUENCE # OF
*                                         FIRST VOL TO BE
*                                         OPENED IN REG 5
LOOP     EQU         *
         STH         R5,JFCBVLSQ          PUTS SEQ # OF VOL
*                                         TO BE OPENED WHERE
*                                         OPEN RTNS LOOK
         OPEN ((R4),OUTPUT),TYPE=J        OPENS ONE VOL
*    NOTE THAT THE TYPE=J OPTION OF THE MACRO MUST BE USED
         LA          R4,DCB2-DCB1(R4)     INCREMENT REG 4 TO
*                                         POINT TO THE DCB FOR
*                                         THE NEXT VOL TO BE
*                                         OPENED
         LA          R5,1(R5)             INCREMENT TO SEQ # OF
*                                         NEXT VOL TO BE OPENED
         BCT         R3,LOOP              LOOP UNTIL ALL VOLS
*                                         OPEN
         .
         .
         .
JFCB     DS          CL176               JFCB READ IN HERE
         ORG         JFCB+70
JFCBVLSQ DS          H                   SEQ # OF VOL TO BE
*                                        OPENED
         ORG         JFCB+117
JFCBNVOL DS          FL1                 # OF VOLS IN DATA SET
         ORG
* MAPPING MACRO IEFJFCBN MAY ALSO BE USED
DCB1 DCB DDNAME=SYSUT1,MACRF=(E),EXLST=EXITS,DSORG=PS
DCB2 DCB DDNAME=SYSUT1,MACRF=(E),EXLST=EXITS,DSORG=PS
DCB3 DCB DDNAME=SYSUT1,MACRF=(E),EXLST=EXITS,DSORG=PS
DCB4 DCB DDNAME=SYSUT1,MACRF=(E),EXLST=EXITS,DSORG=PS
DCB5 DCB DDNAME=SYSUT1,MACRF=(E),EXLST=EXITS,DSORG=PS
* THIS PROCEDURE WORKS FOR 5 VOLS OR LESS; THE JFCB
* EXTENSION, WHICH IDENTIFIES ADDITIONAL VOLS, CAN'T
* BE READ IN
EXITS    DS          0F
         DC          X'87',AL3(JFCB) 87 IDENTIFIES THIS AS
*                                    THE EXIT LIST ENTRY
*                                    THAT SHOWS WHERE JFCB
*                                    WILL BE READ IN
```

Use of the RDJFCB macro instruction and the OPEN macro instruction with the TYPE=J option is explained in detail under "Reading and Modifying a Job File Control Block" on page 144.

EXCP—EXECUTE CHANNEL PROGRAM

The EXCP macro instruction requests the initiation of the I/O operations of a channel program. You must issue EXCP whenever you want to execute one of your channel programs. The format of the EXCP macro instruction is:

[symbol]	EXCP	iob-address

iob-address—A-type address, (2-12), or (1)
the address of the input/output block of the channel program to be executed.

ATLAS—ASSIGNING AN ALTERNATE TRACK AND COPYING DATA FROM THE DEFECTIVE TRACK

A program that uses the EXCP macro instruction for input and output and is APF authorized may use the ATLAS macro instruction, during the execution of the program, to obtain an alternate track and to copy a defective track onto the alternate track. With the use of ATLAS, the program can recover from permanent (hard) errors encountered in the execution of the following types of I/O commands:

- Search ID.
- Write. (The error condition must be confirmed during the execution of the channel program by a CCW that checks the data written.)
- Read count. Errors in the CCHHR part of the count area can be recovered from unless the record is the home address or record zero. Errors in the KDD part of the count area cannot be recovered from unless the user has identified the defective record.

Note: ATLAS may be used for all direct access devices with the exception of MSS volumes (3330V).

Your DCB must include the DCBRECFM field, and the field must show whether the data set is in the track overflow format. If it is, recovery from errors in last records on tracks depends on your identifying the track overflow record segments.

Recovery takes the form of obtaining a good alternate track and copying the defective track onto the good alternate one. Unless a reexecution of the channel program by ATLAS can correct the defect, the user should examine, and if necessary replace, defective records in a subsequent job if the data set is to be processed again.

The format is:

<table>
<tr><td>[symbol]</td><td>ATLAS</td><td>PARMADR={address}
[,CHANPRG={R def.|NR}]
[,CNTPTR={P|F}]
[,WRITS={YES|NO}]</td></tr>
</table>

PARMADR
Address of a parameter address list of the following format:

0	Address of IOB for the channel program that encountered the error
4	Address of count area field

The count area field contains the CCHHRKDD of a defective record or the CCHH of a track that is to be copied.

address—A-type address, (2-12), or (1)

CHANPRG={R|NR}
specifies whether the channel program that encountered the error can be executed again.

R Channel program may be executed again by ATLAS. Before permitting reexecution of the channel program by ATLAS, you must reset the error indications of the previous execution fields in the DCBIFLGS. (See the example of the use of ATLAS below.)

NR Channel program may not be executed again.

If this parameter is omitted, R is assumed.

CNTPTR
specifies whether the count area field contains a full count area (CCHHRKDD) or a partial count area (CCHH).

P Part of the count area (the CCHH address of the track to be copied).

F Full count area (CCHHRKDD count of the record that was found defective).

If this parameter is omitted, P is assumed.

WRITS
track overflow segment identification.

If your data set is in the track overflow format, this identification determines recovery from errors in last records on tracks.

YES If this is the last record on the track, it is a segment other than the last of a track overflow record.

NO If this is the last record on the track, it is the last or only segment of a track overflow record.

If this parameter is omitted, it is assumed that it cannot be established whether a last record is a segment of an overflow record.

Using ATLAS

If a channel program encounters a unit check condition (shown in the CSW) in its execution, the EXCP Processor program will place the sense bytes in the IOB. ATLAS can be used to recover from sense conditions shown by the following bit settings:

IOBSENS0	X'08'	Data check
IOBSENS1	X'80'	Permanent
IOBSENS1	X'02'	Missing address marker (see the following for combinations of this bit setting which ATLAS cannot handle).

However, defects in the home address record or the record zero record cannot be recovered from through the use of ATLAS. These conditions are shown by:

IOBSENS1 X'02' and IOBSENS0 X'01'—home address defect.

IOBSENS1 X'0A'—record zero defect, or, home address cannot be located.

Also, before using ATLAS, you must reset error indications as follows:

```
NI  DCBIFLGS,X'3F'    Reset the DCBIFLGS error indications.
```

The ATLAS program will attempt to find a good alternate track and will attempt to copy the defective track onto the good track, including all error conditions in either key or data areas. The error conditions may be rectified by reexecuting the channel program or through the use of the IEHATLAS utility program in a subsequent step.

Example: The following illustrates the use of the ATLAS macro instruction.

```
         EXCP     MYIOB
         WAIT     ECB=MYECB
         TM       MYECB,X'7F'          TEST FOR I/O ERROR
         BO       NEXT                 NO, SUCCESSFUL, GO TO
*                                      ANOTHER ROUTINE
         TM       IOBCSW+3,X'02'       UNIT CHECK
         BZ       OTHER                NO, DO OTHER ERROR
*                                      PROCESSING
         TM       IOBSENS0,X'08'       DATA CHECK
         BNO      OTHER                NO, CAN'T HANDLE
         TM       IOBSENS1,X'80'       PERMANENT
         BNO      OTHER                NO, CAN'T HANDLE
         NI       DCBIFLGS,X'3F'       RESET ERROR
*                                      INDICATORS
         ATLAS    PARMADR=THERE,CHANPRG=R
```

Operation of the ATLAS Program

The ATLAS program (SVC 86):

- Establishes the availability and address of the next alternate track from the format-4 DSCB of the VTOC.
- Brings all count fields from the defective track into storage to establish the description of the track.
- Initializes the alternate track. (Writes the home address and record zero.)
- Brings the key and data areas of each record into storage, one at a time, and combines them with their new count area to write the complete record onto the alternate track.
- When the copying is finished, chains the alternate to the defective track and updates the VTOC.

Control is returned to your program at the next executable instruction following the ATLAS macro instruction. The success of the ATLAS macro instruction can be determined by examining the contents of register 15, which will contain one of the return codes described below. If register 15 contains decimal 0, 36, 40, or 44, the contents of register 0 may be significant.

Code **Meaning**

0(00) Successful completion. Key and data areas have been copied from the defective track onto a good alternate one. The only error encountered was in the record identified by the user's CCHHRKDD value.

If the channel program is reexecutable, it has been successfully reexecuted.

4(04) This device type does not have alternate tracks that can be assigned by programming.

8(08) All alternate tracks for the device have been assigned.

12(0C) A request for storage (GETMAIN macro instruction) could not be satisfied.

16(10) All attempts to initialize and transfer data to an alternate track failed. The number of attempts made is equal to 10% of the assigned alternates for the device.

20(14) The type of error shown by the sense byte cannot be handled through the use of the ATLAS macro instruction. The condition is other than a data check (in the count or data areas) or a missing address marker.

24(18) The format-4 DSCB of the VTOC cannot be read; therefore alternate track information is not available to ATLAS.

28(1C) The record specified by the user was the format-4 DSCB and it could not be read.

32(20) An error found in count area of last record on the track cannot be handled because last-record-on-track identification is not supplied.

36(24) An error was encountered when reading or writing the home address record or record zero. No error recovery has taken place. If register 0 contains X'01 00 00 00', the defect is in record zero.

40(28) Successful completion. Key and data areas have been copied from the defective track onto a good alternate one. However, the alternate track may have records with defective key or data areas. Register 0 identifies the first three found defective as follows:

```
  n R R R
```

n—The number of record numbers that follow (0, 1, 2, or 3).

R—The hexadecimal number of the record found defective but copied anyway.

If the channel program is reexecutable, it has been successfully reexecuted.

44(2C) Errors encountered and no alternate track has been assigned. The return parameter register (register 0) will contain the R of a maximum of three error records.

Error conditions that return this code are:

1. ATLAS received an error indication for a record with a data length in the count field of zero. Recovery was not possible because a distinction cannot be made between an EOF record and an invalid data length.

2. An error occurred while reading the count field of a record and the KDD (key length-data length) was found to be defective.

3. More than three records on the specified track contained errors in their count fields.

48(30) No errors found on the track, no alternate assigned. ATLAS will not assign an alternate unless a track has at least one defective record.

52(34) I/O error in reexecuting user's channel program. A good alternate is chained to the defective track and data has been transferred. The user's control blocks will give indication of the error condition causing failure in reexecution of the channel program.

56(38) The DCB reflects a track overflow data set, but the UCB device type shows that the device does not support track overflow.

60(3C) The CCHH of the user-specified count area is not within the extents of the data set.

64(40) The device is an MSS virtual device, which is not supported.

EOV—END OF VOLUME

The EOV macro instruction identifies end-of-volume and end-of-data-set conditions. For an end-of-volume condition, EOV causes switching of volumes and verification or creation of standard labels. For an end-of-data-set condition, EOV causes your end-of-data set routine to be entered. Before processing trailer labels on a tape input data set, you must decrement the DCBBLKCT field. You issue EOV if switching of magnetic tape or direct access volumes is necessary, or if secondary allocation is to be performed for a direct access data set opened for output.

For magnetic tape, you must issue EOV when either a tapemark is read or a reflective spot is written over. In these cases, bit settings in the 1-byte DCBOFLGS field of the data control block determine the action to be taken when EOV is executed. Before issuing EOV for magnetic tape, you must make sure that appropriate bits are set in DCBOFLGS. Bit positions 2, 3, 6, and 7 of DCBOFLGS are used only by the system; you are concerned with bit positions 0, 1, 4, and 5. The use of these DCBOFLGS bit positions is as follows:

Bit 0
set to 1 indicates that a write command was executed and that a tapemark is to be written.

Bit 1
indicates that a backward read was the last I/O operation.

Bit 4
indicates that data sets of unlike attributes are to be concatenated.

Bit 5
indicates that a tapemark has been read.

If bits 0 and 5 of DCBOFLGS are both off when EOV is executed, the tape is spaced past a tapemark, and standard labels, if present, are verified on both the old and new volumes. The direction of spacing depends on bit 1. If bit 1 is off, the tape is spaced forward; if bit 1 is on, the tape is backspaced.

If bit 0 is on, but bit 5 is off, when EOV is executed, a tapemark is written immediately following the last data record of the data set. Standard labels, if specified, are created on the old and the new volume.

After issuing EOV for sequentially organized output data sets on direct access volumes, you can determine whether additional space was obtained on the same or a different volume. You do this by examining the data extent block (DEB) and the unit control block (UCB). If neither the address of the UCB, as shown in the DEB, nor the volume serial number, as shown in the UCB, have changed, additional space was obtained on the same volume. Otherwise, space was obtained on a different volume.

The only parameter of the EOV macro instruction is:

[symbol]	**EOV**	dcb address

dcb address—A-type address, (2-12), or (1)
the address of the data control block that is opened for the data set. If this parameter is specified as (1), register 1 must contain this address.

Note: To learn how the system disposes of a tape volume when an EOV macro is issued, see the description of the DISP parameter under "OPEN—Initialize Data Control Block" on page 87.

CLOSE—RESTORE DATA CONTROL BLOCK

The CLOSE macro instruction restores one or more data control blocks so that processing of their associated data sets can be terminated. You must issue CLOSE for all data control blocks that were used by your channel programs. Some of the procedures performed when CLOSE is executed are:

- Release of data extent block (DEB)

- Removal of information transferred to data control block fields when OPEN was executed

- Verification or creation of standard labels

- Volume disposition

- Release of programmer-written appendage routines

When CLOSE is issued for data sets on magnetic tape volumes, labels are processed according to bit settings in the DCBOFLGS field of the data control block. Before issuing CLOSE for magnetic tape, you must set the appropriate bits in DCBOFLGS. The DCBOFLGS bit positions that you are concerned with are listed in the EOV macro instruction description.

For information about the forms of the CLOSE macro and their parameters, refer to *Data Management Macro Instructions*.

CONTROL BLOCK FIELDS

The fields of the input/output block, event control block, and data extent block are illustrated and explained here; the data control block fields are described with the parameters of the DCB macro instruction under "EXCP Requirements" on page 68.

INPUT/OUTPUT BLOCK FIELDS

The input/output block (IOB) is not automatically constructed by a macro instruction; it must be defined as a series of constants and must be on a fullword boundary. For unit-record and tape devices, the IOB is 32 bytes in length. For direct access, teleprocessing, and graphic devices, 8 additional bytes must be provided. You may want to use the system mapping macro IEZIOB, which expands into a DSECT, to help in constructing an IOB.

In Figure 22 the diagonally-ruled areas indicate fields in which you must specify information. The other fields are used by the system and must be defined as all zeros. You may not place information into these fields, but you may examine them.

IOBFLAG1 (1 byte)
You must set bit positions 0, 1, and 6. One-bits in positions 0 and 1 indicate data chaining and command chaining, respectively. (If both data chaining and command chaining are specified, the system does not use error recovery routines except for the 2671, 1052, 2150, and the direct access devices.) A one-bit in position 6 indicates that the channel program is not a 'related' request; that is, the channel program is not related to any other channel program. If you intend to issue an EXCP macro with a BSAM, QSAM, or BPAM data control block, you may want to turn on bit 7 to prevent access-method appendages from processing the I/O request.

IOBFLAG2 (1 byte)
If you set bit 6 in the IOBFLAG1 field to zero, then bits 2 and 3 in this field must be set to:

- 00, if any channel program or appendage associated with a related request might modify this IOB or channel program.

- 01, if the conditions requiring a 00 setting don't apply, but the CHE or ABE appendage might retry this channel program if it completes normally or with the unit-exception or wrong-length-record bits on in the CSW.

- 10 in all other cases.

The three combinations of bits 2 and 3 represent the three kinds of related requests, known as type 1 (00), type 2 (01), and type 3 (10). The type you use determines how much the EXCP Processor can overlap the processing of related requests. Type 3 allows the greatest overlap, normally making it possible to quickly reuse a device after a channel-end interruption. (Related requests that were executed on a pre-MVS system are executed as type-1 requests if not modified.)

IOBSENS0 and IOBSENS1 (2 bytes)
are placed into the input/output block by the EXCP Processor when a unit check occurs. On occasion, the system is unable to obtain any sense bytes because of unit checks when sense commands are issued. In this case the system simulates sense bytes by moving X'10FE' to IOBSENS0 and IOBSENS1.

IOBECBCC (1 byte)
the first byte of the completion code for the channel program. The system places this code in the high-order

<table>
<tr><td>0(0) IOBFLAG1</td><td>IOBFLAG2</td><td>IOBSENS0</td><td>IOBSENS1</td><td rowspan="8">All Devices</td></tr>
<tr><td>4(4) IOBECBCC</td><td colspan="3">IOBECBPT</td></tr>
<tr><td>8(8) IOBFLAG3</td><td colspan="3" rowspan="2">IOBCSW</td></tr>
<tr><td>12(C)</td></tr>
<tr><td>16(10) IOBSIOCC</td><td colspan="3">IOBSTART</td></tr>
<tr><td>20(14) Reserved</td><td colspan="3">IOBDCBPT</td></tr>
<tr><td>24(18) IOBRESTR</td><td colspan="3">IOBRESTR+1</td></tr>
<tr><td colspan="2">28(1C) IOBINCAM</td><td colspan="2">IOBERRCT</td></tr>
<tr><td>32(20) IOBSEEK (first byte, M)</td><td colspan="4">Direct Access, Teleprocessing, and Graphic Devices</td></tr>
<tr><td colspan="4">33(21) IOBSEEK (second through eighth bytes, BBCCHHR) 39(27)</td><td>Direct Access Storage Devices (DASD)</td></tr>
</table>

Figure 22. Input/Output Block Format

byte of the event control block when the channel program is posted complete. The completion codes and their meanings are listed under "Event Control Block Fields" on page 97.

IOBECBPT (3 bytes)
the address of the 4-byte event control block you have provided.

IOBFLAG3 (1 byte)
is used only by the system.

IOBCSW (7 bytes)
the low-order seven bytes of the channel status word, which are placed into this field each time a channel-end or PCI interruption occurs.

IOBSIOCC (1 byte)
in bits 0 and 1, the instruction-length code; in bits 2 and 3, the start I/O (SIO) condition code for the instruction the system issues to start the channel program; and in bits 4 through 7, the program mask.

IOBSTART (3 bytes)
the starting address of the channel program to be executed.

Reserved (1 byte)
used only by the system.

IOBDCBPT (3 bytes)
the address of the data control block of the data set to be read or written by the channel program.

IOBRESTR (1 byte)
used by the system for volume repositioning in error recovery procedures.

IOBRESTR+1 (3 bytes)
used by the system, if a related channel program is permanently in error, to chain together IOBs that represent dependent channel programs. To learn more about the conditions under which the chain is built, refer to "Interruption Handling and Error Recovery Procedures" on page 71.

IOBINCAM (2 bytes)
for magnetic tape, the amount by which the block count (DCBBLKCT) field in the device-dependent portion of the data control block is to be incremented. You may alter these bytes at any time. For forward operations, these bytes should contain a binary positive integer (usually +1); for backward operations, they should contain a binary negative integer. When these bytes are not used, all zeros must be specified.

Reserved (2 bytes)
used only by the system.

IOBSEEK (first byte, M)
for direct access devices, the extent entry in the data extent block that is associated with the channel program (0 indicates the first entry; 1 indicates the second, and so forth). For teleprocessing and graphic devices, it contains the UCB index.

IOBSEEK (last 7 bytes, BBCCHHR)
for direct access devices, the seek address for your channel program.

EVENT CONTROL BLOCK FIELDS

You must define an event control block (ECB) as a 4-byte area on a fullword boundary. When the channel program has been completed, the input/output supervisor places a completion code containing status information into the ECB (Figure 23 on page 98). Before examining this information, you must test for the setting of the "complete bit." If the complete bit is not on, and your problem program cannot perform other useful operations, you should issue a WAIT macro instruction that specifies the event control block. Under no circumstances should you construct a program loop that tests for the complete bit.

DATA EXTENT BLOCK FIELDS

The data extent block (DEB) is constructed by the system when an OPEN macro instruction is issued for the data control block. You may not modify the fields of the DEB, but you may examine them. The DEB format and field descriptions are contained in Debugging Handbook.

WAIT bit=0	COMPLETE bit=1	Remainder of completion code

```
bit
0                  1                  2                                       31
```

Wait bit
A one-bit in this position indicates that the WAIT macro instruction has been issued, but the channel program has not been completed.

Complete bit
A one-bit in this position indicates that the channel program has been completed; if it has not been completed, a zero-bit is in this position.

Completion code
This code, which includes the wait and complete bits, may be one of the following 4-byte hexadecimal expressions:

Code	**Meaning**
7F000000	The channel program has terminated without error.
41000000	The channel program has terminated with a permanent error.
42000000	The channel program has terminated because a direct access extent address has been violated.
44000000	The channel program has been intercepted because of a permanent error associated with a device end for the previous request. You may reissue the EXCP macro instruction to restart the channel program.
48000000	The request queue element for a channel program has been made available after it has been purged.
4B000000	One of the following errors occurred during error recovery processing for a tape device. • The CSW command address in the IOB is zeros. • An unexpected load point was encountered.
4F000000	Error recovery routines have been entered because of direct access error but are unable to read the home address or record 0.

Figure 23. Event Control Block after Posting of Completion Code (EXCP)

EXECUTING FIXED CHANNEL PROGRAMS IN REAL STORAGE (EXCPVR)

The EXCPVR macro instruction provides you with the same functions as the EXCP macro instruction (that is, a device-dependent means of performing input/output operations). In addition, it allows your program to improve the efficiency of the I/O operations in a paging environment by translating its own virtual channel programs to real channel programs. Authorized programs are allowed to execute in a V=V area and provide the EXCP processor with real channel programs. This eliminates the translation of channel programs by the EXCP processor. The program issuing the EXCPVR must remain in authorized state until the completion of the channel programs.

Problem programs are authorized to use the EXCPVR macro instruction under the authorized program facility (APF). A description of how to authorize a program can be found in System Programming Library: Supervisor Services and Macro Instructions.

[symbol]	EXCPVR	iob-address

iob-address—A-type address, (2-12), or (1)
the address of the input/output block of the channel program to be executed.

To use EXCPVR, you must do all the things you would do to execute an EXCP request; in addition you must:

1. Code PGFX=YES in the DCB associated with the EXCPVR requests and provide a page-fix (PGFX) appendage by specifying SIOA=symbol in the DCB.

2. Fix the data area that contains your channel program, the data areas that are referred to by your channel program, your PCI appendage (if your program can generate program controlled interrupts), and any area referred to by your PCI appendage. To cause EXCP to fix these data areas, you build a list that contains the addresses of these virtual areas. You should build the list in your PGFX appendage.

3. Determine whether the data areas in virtual storage specified in the address fields of your CCWs cross page boundaries. If they do, you must build an indirect data address list (IDAL) and put the address of the IDAL in the affected CCW.

4. Translate the addresses in your CCWs from virtual to real addresses.

Items 3 and 4 must be done in your start-I/O (SIO) appendage. A description of the SIO appendage is presented under "Appendages" on page 72.

BUILDING THE LIST OF DATA AREAS TO BE FIXED

The EXCP processor expects programs using the EXCPVR macro instruction to pass a list of data areas to be fixed. This list is to be built in the PGFX appendage, as described below.

The data areas you must fix in real storage (if not already fixed in real storage) are:

1. The channel program. If the channel program is already in a fixed subpool, it does not have to be fixed.

2. The data areas from which your channel program will be writing and to which your channel program will be reading. If the data areas are already in a fixed subpool, they do not have to be fixed.

3. The PCI appendage, if used, and any areas referred to in the PCI appendage.

4. Any system or user control blocks (as well as, the DEB).

You need not fix areas that have already been fixed, such as the modules that reside in the fixed link pack area (LPA).

PAGE FIX (PGFX) AND START-I/O (SIO) APPENDAGE

This appendage comprises two essentially independent appendages. The complete appendage can be viewed as a reenterable subroutine having two entry points, one for the SIO appendage and one for the PGFX appendage.

The SIO entry point is located at offset 0 in the subroutine; any other location in the appendage may be branched to from this entry point. The entry point of the PGFX appendage is at offset

+4 in the SIO subroutine, which is set in register 15 as the entry point of the PGFX appendage.

Page Fix (PGFX) Appendage: The purpose of this appendage is to list all of the areas that must be fixed to prevent paging exceptions during the execution of the current I/O request. This appendage may be entered more than once. However, each time it is entered, it must create the same list of areas to be fixed. The appendage may use the 16-word save area pointed to by register 13. Registers 10, 11, and 13 may be used as work registers.

Page-Fix List Processing

Each page-fix entry placed in the list by the appendage must have the following doubleword format:

X'00'	Starting virtual address of area to be fixed	X'00'	Ending virtual address of area to be fixed + 1
<—1 byte—>	<——3 bytes——>	<—1 byte—>	<——3 bytes——>

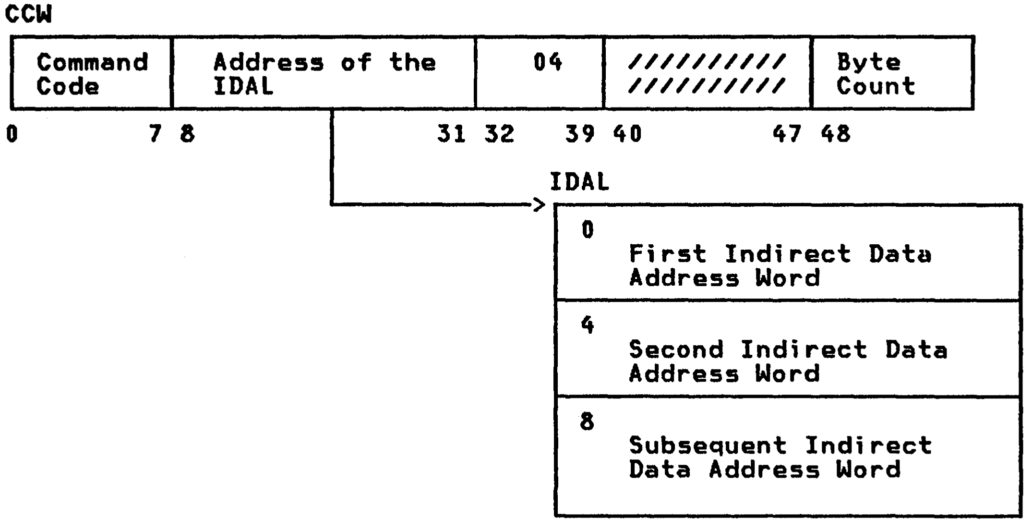
On return from your PGFX appendage to the EXCP processor (via the return address provided in register 14), register 10 must point to the first page-fix entry and register 11 must contain the number of page-fix entries in the work area. The EXCP processor then fixes the pages corresponding to the areas listed by the PGFX appendage. The pages remain fixed until the associated I/O request terminates.

If either the channel end appendage or the abnormal end appendage returns via the return address in register 14 plus 8, the PGFX appendage is not normally reentered. Instead, the SIO appendage is entered, and the page-fix list built by the PGFX appendage is still active. However, the PGFX appendage is entered after either the channel end appendage or the abnormal end appendage returns via the return address in register 14 plus 8 when a PURGE macro has been issued (for instance, when a memory swap has occurred). In this case, when I/O is restored, the PGFX appendage is entered.

Note: The page-fix list must be in page-fixed storage.

SIO APPENDAGE: If you are using EXCPVR to execute your channel program, you must translate the virtual addresses in the operands of your channel program to real addresses. This should be done in your SIO appendage. If indirect data addressing is required, the SIO appendage should also build the indirect data address lists (IDALs) and turn on the IDA indicators in the associated CCWs.

Translating Virtual Addresses and Building the IDAL: You must convert the virtual addresses in the channel program to real addresses. You must also check the areas whose addresses appear in bits 8-31 of your CCWs to determine whether the data areas cross 2K-byte boundaries. If they do, you must provide an entry in the IDAL for each 2K-byte boundary crossed. The channel subsystem uses the IDAL to identify the address at which it will continue reading or writing when a 2K-byte boundary is crossed during a read or write operation. The IDAL must contain real addresses when it is processed by the channel.


CCW
Command Code
Address of the IDAL
04
//////////
//////////
Byte Count
0
7
8
31
32
39
40
47
48
IDAL
0
First Indirect Data Address Word
4
Second Indirect Data Address Word
8
Subsequent Indirect Data Address Word

Notes:

1. You must put one entry in the IDAL for each 2K-byte page boundary your data area crosses.

2. If the CCW has an IDAL address rather than a data address, bit 37 must be set to signal this to the channel.

3. The maximum number of entries needed in the IDAL is determined from the count in the CCW as follows:

   Number of IDAL entries=((CCW byte-count - 1)/2048) + 1. (Round up division to next highest integer if remainder is not zero.)

The number of IDAL entries required ultimately depends on the number of 2K-byte boundaries crossed by the data. For example, if your data is 800 bytes long and does not cross a 2K-byte page boundary, no IDAL entries are required. If your data crosses a 4K-byte page boundary, then two IDAL entries are required. If your data is 5000 bytes long, at least two IDAL entries are required. If your data crosses two 4K-byte page boundaries, four IDAL entries are required.

The first indirect address is the real address of the first byte of the data area. The second and subsequent indirect addresses are the real addresses of the second and subsequent 2K-byte boundaries of the data area.

For example, if the data area real address is X'707FF' and the byte count is X'1802', the IDAL would contain the following real addresses (assuming the real addresses are contiguous, which may not always be the case):

```
707FF
70800
71000
```

If the data area real address is X'707FF' and the byte count is X'800', the IDAL would contain the following addresses:

```
707FF
70800
```

CHAPTER 4. USING XDAP TO READ AND WRITE TO DIRECT-ACCESS DEVICES

The execute direct access program (XDAP) macro instruction provides you with a means of reading, verifying, or updating blocks on direct access volumes without using an access method and without writing your own channel program. This chapter explains what the XDAP macro instruction does and how you can use it. The control block generated when XDAP is issued and the macro instructions used with XDAP are also discussed.

Since most of the specifications for XDAP are similar to those for the execute channel program (EXCP) macro instruction, you should be familiar with the "Executing Your Own Channel Programs (EXCP)" chapter of this publication, as well as with the information contained in Data Management Services which provides how-to information for using the access method routines of the system control program.

INTRODUCTION

Execute direct access program (XDAP) is a macro instruction that you may use to read, verify, or update a block on a direct access volume. If you are not using the standard IBM data access methods, you can, by issuing XDAP, generate the control information and channel program necessary for reading or updating the records of a data set. (XDAP cannot be used, however, to read, verify, or update a SYSIN or SYSOUT data set.)

You cannot use XDAP to add blocks to a data set, but you can use it to change the keys of existing blocks. Any block configuration and any data set organization can be read or updated.

Although the use of XDAP requires less storage than do the standard access methods, it does not provide many of the control program services that are included in the access methods. For example, when XDAP is issued, the system does not block or deblock records and does not verify block length.

To issue XDAP, you must provide the actual track address of the track containing the block to be processed. You must also provide either the block identification or the key of the block, and specify which of these is to be used to locate the block. If a block is located by identification, both the key and data portions of the block may be read or updated. If a block is located by key, only the data portion can be processed.

For additional control over I/O operations, you may write appendages, which must be entered into the LPA library. Descriptions of these routines and their coding specifications are included under "Executing Your Own Channel Programs (EXCP)."

XDAP REQUIREMENTS

When using the XDAP macro instruction, you must, somewhere in your program, code a DCB macro instruction, which produces a data control block (DCB) for the data set to be read or updated. You must also code an OPEN macro instruction, which initializes the data control block and produces a data extent block (DEB). The OPEN macro instruction must be executed before any XDAP macro instructions are executed.

When the XDAP macro instruction is assembled, a control block and executable code are generated. This control block may be logically divided into three sections:

- An event control block (ECB), which is supplied with a completion code each time the direct access channel program is terminated.

- An input/output block (IOB), which contains information about the direct access channel program.

- A direct access channel program, which consists of three or four channel command words (CCWs). The type of channel program generated depends on specifications in the parameters of the XDAP macro instruction. When executed, it locates a block by either its actual address or its key and reads, updates, or verifies the block.

When the channel program has terminated, a completion code is placed into the event control block. After issuing XDAP, you should therefore issue a WAIT macro instruction, specifying the address of the event control block, to regain control when the direct access program has terminated. If volume switching is necessary, you must issue an EOV macro instruction. When processing of the data set has been completed, you must issue a CLOSE macro instruction to restore the data control block.

MACRO SPECIFICATIONS FOR USE WITH XDAP

When you are using the XDAP macro instruction, you must also code DCB, OPEN, CLOSE, WAIT, and, in some cases, the EOV macro instructions. The parameters of the XDAP macro instruction are listed and described here. For the other required macro instructions, special requirements or options are explained, but you should refer to "Macro Specifications for Use with EXCP" on page 80 for listings of their parameters.

DCB—DEFINE DATA CONTROL BLOCK

You must issue a DCB macro instruction for each data set to be read, updated, or verified by the direct access channel program. Refer to "DCB—Define Data Control Block for EXCP" on page 80 to learn which macro instruction parameters to code.

OPEN—INITIALIZE DATA CONTROL BLOCK

The OPEN macro instruction initializes one or more data control blocks so that their associated data sets can be processed. You must issue OPEN for all data control blocks that are to be used by the direct access program. Some of the procedures performed when OPEN is executed are:

- Construction of data extent block (DEB).

- Transfer of information from DD statements and data set labels to the data control block.

- Verification or creation of standard labels.

- Loading of programmer-written appendage routines.

The two parameters of the OPEN macro instruction are the address(es) of the data control block(s) to be initialized, and the intended method of I/O processing of the data set. The method of processing may be specified as INPUT, OUTPUT, EXTEND; however, if nothing is specified, INPUT is assumed.

XDAP—EXECUTE DIRECT-ACCESS PROGRAM

The XDAP macro instruction produces the XDAP control block (that is, the ECB, IOB, and channel program) and executes the direct access channel program. The format of the XDAP macro instruction is:

```
[symbol]   XDAP      ecb-symbol
                     ,type
                     ,dcb-addr
                     ,area-addr
                     ,length-value
                     ,[(key-addr,keylength-value)]
                     ,blkref-addr
                     ,[sector-addr]
                     [,MF={E|L}]
```

ecb-symbol—symbol or (2-12)
the symbolic name to be assigned to the XDAP event control block. Registers can be used only with MF=E.

type—{RI|RK|WI|WK|VI|VK}
the type of I/O operation intended for the data set and the method by which blocks of the data set are to be located. One of the combinations shown must be coded in this field.

The codes and their meanings are:

R Read a block.

W Update a block.

V Verify that the device is able to read the contents of a block, but do not transfer data.

I Locate a block by identification. (The key portion, if present, and the data portion of the block are read, updated, or verified.)

K Locate a block by key. (Only the data portion of the block is read, updated, or verified.) If you code this value, you must code the 'key-addr,keylength-value' operands.

dcb-addr—A-type address or (2-12)
the address of the data control block for the data set. If this data control block is also being used by a sequential access method (BSAM, BPAM, QSAM), you must reassemble the XDAP macro instruction. Otherwise, sequential access method appendages will be called at the conclusion of the XDAP channel program.

area-addr—A-type address or (2-12)
the address of an input or output area for a block of the data set.

length-value—absexp or (2-12)
the number of bytes to be transferred to or from the input or output area. If blocks are to be located by identification and the data set contains keys, the value must include the length of the key. The maximum number of bytes transferred is 32,767.

key-addr—RX-type address or (2-12)
when blocks are to be located by key, the address of a virtual storage field that contains the key of the block to be read, updated, or verified.

keylength-value—absexp or (2-12)
when blocks are to be located by key, the length of the key. The maximum length is 255 bytes.

<u>blkref-addr—RX-type address or (2-12)</u>

the address of a field in virtual storage containing the actual track address of the track containing the block to be located. The actual address of a block is in the form MBBCCHHR, where M indicates which extent entry in the data extent block is associated with the direct access program; BB is not used but must be zero; CC indicates the cylinder address; HH indicates the actual track address; and R indicates the block identification. R is not used when blocks are to be located by key. (For more detailed information, see "Conversion of Relative Track Address to Actual Track Address" on page 107.)

<u>sector-addr—RX-type address or (2-12)</u>

the address of a 1-byte field containing a sector value. The sector-address parameter is used for rotational position sensing (RPS) devices only. The parameter is optional, but its use will improve channel performance. When the parameter is coded, a set-sector CCW (using the sector value indicated by the data address field) precedes the search-ID-equal command in the channel program. The sector-address parameter is ignored if the type parameter is coded as RK, WK, or VK. If a sector address is specified in the execute form of the macro, then a sector address, not necessarily the same, must be specified in the list form. The sector address in the executable form will be used.

Note: No validity check is made on either the address or the sector value when the XDAP macro is issued. However, a unit check/command reject interruption will occur during channel-program execution if the sector value is invalid for the device or if the sector-addr operand is used when accessing a device without RPS. (For more detailed information, see "Obtaining Sector Number of a Block on a Device with the RPS Feature" on page 109.)

MF=

you may use the L-form of the XDAP macro instruction for a macro expansion consisting of only a parameter list, or the E-form for a macro expansion consisting of only executable instructions.

MF=E

The first operand (ecb-symbol) is required and may be coded as a symbol or supplied in registers 2 through 12. The type, dcb-addr, area-addr, and length-value operands may be supplied in either the L- or E-form. The blkref-addr operand may be supplied in the E-form or moved into the IOBSEEK field of the IOB by you. The sector-addr is optional; it may be coded either in both the L- and E-form or in neither.

MF=L

The first two operands (ecb-symbol and type) are required and must be coded as symbols. If you choose to code length-value or keylength-value, they must be absolute expressions. Other operands, if coded, must be A-type addresses. (blkref-addr is ignored if coded.)

The dcb-addr, area-addr, blkref-addr, and sector-value operands may be coded as RX-type addresses or supplied in registers 2 through 12. The length-value and keylength-value operands can be specified as absolute expressions or decimal integers or supplied in registers 2 through 12.

EOV—END OF VOLUME

The EOV macro instruction identifies end-of-volume and end-of-data-set conditions. For an end-of-volume condition, EOV causes switching of volumes and verification or creation of standard labels. For an end-of-data-set condition, EOV causes your end-of-data-set routine to be entered. When using XDAP, you issue EOV if switching of direct access volumes is necessary, or if secondary allocation is to be performed for a direct access data set opened for output.

The only parameter of the EOV macro instruction is the address of the data control block of the data set.

CLOSE—RESTORE DATA CONTROL BLOCK

The CLOSE macro instruction restores one or more data control blocks so that processing of their associated data sets can be terminated. You must issue CLOSE for all data sets that were used by the direct access channel program. Some of the procedures performed when CLOSE is executed are:

- Release of data extent block (DEB)
- Removal of information transferred to data control block fields when OPEN was executed
- Verification or creation of standard labels
- Release of programmer-written appendage routines

The CLOSE macro instruction must identify the address of at least one data control block to be restored, and may specify other options. See Data Management Macro Instructions to learn what these options are and how they are specified.

CONTROL BLOCKS USED WITH XDAP

The three control blocks generated during execution of the XDAP macro instruction are described here.

EVENT CONTROL BLOCK

The event control block (ECB) begins on a fullword boundary and occupies the first 4 bytes of the XDAP control block. Each time the direct access channel program terminates, the I/O supervisor places a completion code containing status information into the event control block (Figure 24 on page 107). Before examining this information, you must wait for the completion of the channel program by issuing a WAIT macro instruction that specifies the address of the event control block.

INPUT/OUTPUT BLOCK

The input/output block (IOB) is 40 bytes in length and immediately follows the event control block. "Control Block Fields" on page 95 contains a diagram of the input/output block (Figure 24 on page 107). You may want to examine the IOBSENS0, IOBSENS1, and IOBCSW fields if the ECB is posted with X'41'.

WAIT bit	COMPLETE bit	Completion code	
bit 0	1	2	31

Wait bit
A one bit in this position indicates that the direct access channel program has not been completed.

Complete bit
A one bit in this position indicates that the channel program has been completed; if it has not been completed, a zero bit is in this position.

Completion code
This code, which includes the wait and complete bits, may be one of the following 4-byte hexadecimal expressions:

Code	**Meaning**
7F000000	Direct access program has terminated without error.
41000000	Direct access program has terminated with permanent error.
42000000	Direct access program has terminated because a direct access extent address has been violated.
4F000000	Error recovery routines have been entered because of direct access error but are unable to read home address or record 0.

Figure 24. Event Control Block after Posting of Completion Code (XDAP)

DIRECT ACCESS CHANNEL PROGRAM

The direct access channel program is 24 bytes in length (except when set sector is used for RPS devices) and immediately follows the input/output block. Depending on the type of I/O operation specified in the XDAP macro instruction, one of four channel programs may be generated. The three channel command words for each of the four possible channel programs are shown in Figure 25 on page 108.

When a sector address is specified with an RI, VI, or WI operation, the channel program is 32 bytes in length. Each of these channel programs in Figure 25 would be, in this case, preceded by a set sector command.

CONVERSION OF RELATIVE TRACK ADDRESS TO ACTUAL TRACK ADDRESS

To issue XDAP, you must provide the actual track address of the track containing the block to be processed. If you know only the relative track address, you can convert it to the actual address by using a resident system routine. The entry point to this conversion routine is labeled IECPCNVT. The address of the entry point (CVTPCNVT) is in the communication vector table (CVT). The address of the CVT is in location 16. (For the displacements and descriptions of the CVT fields, see Debugging Handbook.)

Type of I/O Operation	CCW	Command Code
Read by identification	1	Search ID Equal
	2	Transfer in Channel
Verify by identification[1]	3	Read Key and Data
Read by key	1	Search Key Equal
	2	Transfer in Channel
Verify by key[1]	3	Read Data
Write by identification	1	Search ID Equal
	2	Transfer in Channel
	3	Write Key and Data
Write by key	1	Search Key Equal
	2	Transfer in Channel
	3	Write Data

[1] For verifying operations, the third CCW is flagged to suppress the transfer of information to virtual storage.

Figure 25. The XDAP Channel Programs

The conversion routine does all its work in general registers. You must load registers 0, 1, 2, 14, and 15 with input to the routine. Register usage is as follows:

Register **Use**

0 Must be loaded with a 4-byte value of the form TTRN, where TT is the number of the track relative to the beginning of the data set, R is the identification of the block on that track, and N is the concatenation number of a BPAM data set. (0 indicates the first data set in the concatenation, an unconcatenated BPAM data set, or a non-BPAM data set.)

1 Must be loaded with the address of the data extent block (DEB) of the data set.

2 Must be loaded with the address of an 8-byte area that is to receive the actual address of the block to be processed. The converted address is of the form MBBCCHHR, where M indicates which extent entry in the data extent block is associated with the direct access program (0 indicates the first extent, 1 indicates the second, and so forth); BB is two bytes of zeros; CC is the cylinder address; HH is the actual track address; and R is the block number.

3-8 Are not used by the conversion routine.

9-13 Are used by the conversion routine and are not restored.

14 Must be loaded with the address to which control is to be returned after execution of the conversion routine.

15 Is used by the conversion routine as a base register and must be loaded with the address at which the conversion routine is to receive control.

When control is returned to your program, register 15 will contain one of the following return codes:

Code	Meaning
0(00)	Successful conversion.
4(04)	The relative block address converts to an actual track address outside the extents defined in the DEB.

CONVERSION OF ACTUAL TRACK ADDRESS TO RELATIVE TRACK ADDRESS

To get the relative track address when you know the actual track address, you can use the conversion routine labeled IECPRLTV. The address of the entry point (CVTPRLTV) is in the communication vector table (CVT). The address of the CVT is in location 16.

The conversion routine does all of its work in general registers. You must load registers 1, 2, 14, and 15 with input to the routine. Register usage is as follows:

Register	Use
0	Will be loaded with the resulting TTR0 to be passed back to the caller.
1	Must be loaded with the address of the data extent block (DEB) of the data set.
2	Must be loaded with the address of an 8-byte area containing the actual address to be converted to a TTR. The actual address is of the form MBBCCHHR.
3-8	Are not used by the conversion routine.
9-13	Are used by the conversion routine and are not restored.
14	Must be loaded with the address to which control is to be returned after execution of the conversion routine.
15	Is used by the conversion routine as a base register and must be loaded with the address at which the conversion routine is to receive control.

OBTAINING SECTOR NUMBER OF A BLOCK ON A DEVICE WITH THE RPS FEATURE

To obtain the performance improvement given by rotational position sensing, you should specify the sector-addr parameter in the XDAP macro. For programs that can be used with both RPS and non-RPS devices, the UCBRPS bit (bit 3 at an offset of 17 bytes into the UCB) should be tested to determine whether the device has rotational position sensing. If the UCBRPS bit is off, a channel program with a "set sector" command must not be issued to the device.

The sector-addr parameter on the XDAP macro specifies the address of a one-byte field in your region. You must store the sector number of the block to be located in this field. You can obtain the sector number of the block by using a resident conversion routine, IECOSCR1. The address of this routine is in field CVTOSCR1 of the CVT, and the address of the CVT is in location 16. The routine should be invoked via a BALR 14,15 instruction. If you are passing the track balance to the routine, you invoke the routine using a BAL 14,8(15).

For RPS devices, the conversion routine does all its work in general registers. You must load registers 0, 2, 14, and 15 with input to the routine. Register usage is as follows:

Register **Use**

0 For fixed, standard blocks or fixed, unblocked records not in a partitioned data set: Register 0 must be loaded with a 4-byte value in the form XXKR, where XX is a 2-byte field containing the physical block size, K is a 1-byte field containing the key length, and R is a 1-byte field containing the number of the record for which a sector value is desired. The high-order bit of register 0 must be turned off (set to 0) to indicate fixed-length records.

Passing the track balance: Register 0 must be loaded with the 4-byte value of the track balance of the record preceding the required record.

For all other cases: Register 0 must be loaded with a 4-byte value in the form BBIR, where BB is the total number of key and data bytes on the track up to, but not including, the target record; I is a 1-byte key indicator (1 for keyed records, 0 for records without keys); and R is a 1-byte field containing the number of the record for which a sector value is desired. The high-order bit of register 0 must be turned on (set to 1) to indicate variable-length records.

1 Not used by the sector-convert routine.

2 Must be loaded with a 4-byte field in which the first byte is the UCB device type code for the device (obtainable from UCB+19), and the remaining three bytes are the address of a 1-byte area that is to receive the sector value.

3-8,12,13 Not used.

9-11 Used by the convert routine and are not saved or restored.

14 Must be loaded with the address to which control is to be returned after execution of the sector conversion routine.

15 Used by the conversion routine as a base register and must be loaded with the address of the entry point to the conversion routine.

CHAPTER 5. PASSWORD PROTECTING YOUR DATA SETS

OS/VS password protection does not apply to VSAM data sets. Information about VSAM data set protection is in VSAM Reference and Access Method Services Reference. For information on RACF and its relationship to password protection, refer to Resource Access Control Facility (RACF): General Information Manual. To use the data set protection feature of the operating system, you must create and maintain a PASSWORD data set consisting of records that associate the names of the protected data sets with the password assigned to each data set. There are four ways to maintain the PASSWORD data set:

- You can write your own routines.
- You can use the PROTECT macro instruction.
- You can use the utility control statements of the IEHPROGM utility program.
- If you have TSO, you can use the TSO PROTECT command.

This chapter discusses only the first two of the four ways: It provides technical detail about the PASSWORD data set that is necessary for writing your own routines, and it describes how to use the PROTECT macro instruction. (The last two of the four ways are discussed in other publications, as indicated in the list of publications below.)

Before using the information in this chapter, you should be familiar with information in several related publications. The following publications are recommended:

- Data Management Services contains a general description of the data set protection feature.
- Message Library: System Messages contains a description of the operator messages and replies associated with the data set protection feature.
- JCL contains a description of the data definition (DD) statement parameter used to indicate that a data set is to be password protected.
- DADSM and Common VTOC Access Facility Diagnosis Guide and DADSM Diagnosis Reference contain a description of the PASSWORD data set record format.
- Utilities contains a description of how to maintain the PASSWORD data set using the utility control statements of the IEHPROGM utility program.
- TSO Command Language Reference describes the use of the TSO PROTECT command.

INTRODUCTION

In addition to the usual label protection that prevents opening of a data set without the correct data set name, the operating system provides data set security options that prevent unauthorized access to confidential data. Password protection prevents access to data sets, until a correct password is entered by the system operator, or, for TSO, by a remote terminal operator.

The following are the types of access allowed to password-protected data sets:

- PWREAD/PWWRITE—A password is required to read or write.
- PWREAD/NOWRITE—A password is required to read. Writing is not allowed.
- NOPWREAD/PWWRITE—Reading is allowed without a password. A password is required to write.

To prepare for use of the data set protection feature of the operating system, you place a sequential data set, named PASSWORD, on the system residence volume. This data set must contain at least one record for each data set placed under protection. In turn, each record contains a data set name, a password for that data set, a counter field, a protection mode indicator, and a field for recording any information you desire to log. On the system residence volume, these records are formatted as a "key area" (data set name and password) and a "data area" (counter field, protection mode indicator, and logging field). The data set is searched on the "key area."

Note: The area allocated to the data set should not have been previously used for a PASSWORD data set as this may cause unpredictable results when adding records to the data set.

You can write routines to create and maintain the PASSWORD data set. If you use the PROTECT macro instruction to maintain the PASSWORD data set, see "Using the PROTECT Macro Instruction to Maintain the PASSWORD Data Set" on page 115. If you use the IEHPROGM utility program to maintain the PASSWORD data set, see Utilities. These routines may be placed in your own library or the system's library (SYS1.LINKLIB). You may use a data management access method or EXCP programming to read from and write to the PASSWORD data set.

If a data set is to be placed under protection, it must have a protection indicator set in its label (format-1 DSCB or header 1 tape label). This is done by the operating system when the data set is created, by the IEHPROGM utility program, or, by the PROTECT macro when creating or adding the control password. The protection indicator is set in response to a value in the LABEL= operand of the DD statement associated with the data set being placed under protection. The publication JCL describes the LABEL operand.

Note: Data sets on magnetic tape are protected only when standard labels are used.

Password-protected data sets can only be accessed by programs that can supply the correct password. When the operating system receives a request to open a protected data set, it first checks to see if the data set has already been opened for this job step. If so, only the access mode will be checked to determine whether it is compatible with the protection mode under which it was previously opened. If the data set has not been previously opened by this job step, or if the access mode is not compatible with the protection mode under which it was previously opened, a message is issued that asks for the password; the message goes to the operator console. If the program requesting that the data set be opened is running under TSO in the foreground, the message goes to the TSO terminal operator. If you want the password supplied by another method in your installation, you can modify the READPSWD source module or code a new routine to replace READPSWD in SYS1.LPALIB.

PASSWORD DATA SET CHARACTERISTICS

The PASSWORD data set must reside on the same volume as your operating system. The space you allocate to the PASSWORD data set must be contiguous, that is, its DSCB must indicate only one extent. The amount of space you allocate depends on the number of data sets your installation wants to protect. Each entry in the PASSWORD data set requires 132 bytes of space. The organization of the PASSWORD data set is physical sequential, the record format is unblocked, fixed-length records (RECFM=F). Each record, which forms the data area, is 80 bytes long (LRECL=80,BLKSIZE=80), and is preceded by a 52-byte key (KEYLEN=52). The key area contains the fully qualified data set name of up to 44 bytes and a password of one to eight bytes, left justified with blanks added to fill the areas. The password assigned may be from one to eight alphameric characters in length. DADSM and Common VTOC Access Facility Diagnosis Guide and DADSM Diagnosis Reference describe the PASSWORD data set record format.

Note: For data sets on magnetic tape designed according to the specifications of the International Organization for Standardization (ISO) 1001-1979 or the equivalent American National Standards Institute (ANSI) X3.27-1978, do not include generation and version numbers as part of generation data set names. The generation and version numbers are not included as part of the names in the tape labels, and are ignored if included in the PASSWORD data set.

You can protect the PASSWORD data set itself by creating a password record for it when your program initially builds the data set. Thereafter, the PASSWORD data set cannot be opened (except by the operating system routines that scan the data set) unless the operator enters the password.

Note: If a problem occurs on a password-protected system data set, maintenance personnel must be provided with the password in order to access the data set and resolve the problem.

CREATING PROTECTED DATA SETS

A data definition (DD) statement parameter (LABEL=) may be used to indicate that a data set is to be password-protected. For data sets on DASD, an alternative method is to use the PROTECT macro instruction for a previously allocated data set. A data set may be created and the protection indicator set in its label without entering a password record for it in the PASSWORD data set.

Operating procedures at your installation must ensure that password records for all data sets currently password-protected are entered in the PASSWORD data set. Installations where independent computing systems share common DASD resources must ensure that PASSWORD data sets on all systems contain the appropriate password records for any protected data set on shared DASD.

Under certain circumstances, the order in which data sets are allocated and deallocated from multiple systems on shared DASD may result in loss of password-protection. For example, if an unprotected data set is allocated and opened by a user on System A and then scratched by a different user on System B, the first user is given a "window" to the unallocated (free) area. If any data set, protected or unprotected, is allocated in that space by a user on either system during the time the "window" is open, the new data set has no protection from the user with the "window."

While the allocation disposition is still NEW, a password-protected data set can be used without supplying a password. However, after the data set is deallocated, any subsequent attempt to open will result in termination of the program unless the password record is available and the correct password is supplied. Note that, if the protection mode is NOPWREAD and the request is to open the data set for input or read backward, no password will be required.

Tape Volumes Containing More Than One Password-Protected Data Set

To password protect a data set on a tape volume containing other data sets, you must password protect all the data sets on the volume. (Standard labels—SL, SUL, AL, or AUL—are required. See Magnetic Tape Labels and File Structure for definitions of these label types and the protection-mode indicators that can be used.)

If you issue an OPEN macro instruction to create a data set following an existing, password-protected data set, the password of the existing data set will be verified during open processing for the new data set. The password supplied must be associated with a PWWRITE protection-mode indicator.

PROTECTION FEATURE OPERATING CHARACTERISTICS

The topics that follow provide information concerning actions of the protection feature in relation to termination of processing, volume switching, data set concatenation, SCRATCH and RENAME functions, and counter maintenance.

Termination of Processing

Processing is terminated when:

1. The operator cannot supply the correct password for the protected data set being opened after two tries.

2. A password record does not exist in the PASSWORD data set for the protected data set being opened.

3. The protection-mode indicator in the password record, and the method of I/O processing specified in the Open routine do not agree, for example, OUTPUT specified against a read-only protection-mode indicator.

4. There is a mismatch in data set names for a data set involved in a volume switching operation. This is discussed in the next paragraph.

Volume Switching

The system ensures a continuation of password protection when volumes of a multivolume data set are switched. It accepts a newly-mounted tape volume, to be used for input, or a newly-mounted direct access volume, regardless of its use, if these conditions are met:

- The data set name in the password record for the data set is the same as the data set name in the JFCB. (This ensures that the problem program has not changed the data set name in the JFCB since the data set was opened.)

- The protection-mode indicator in the password record is compatible with the processing mode and a valid password has been supplied.

The system accepts a newly-mounted tape volume to be used for output under any of these conditions:

- The security indicator in the HDR1 label indicates password protection, the data set name in the password record is the same as the data set name in the JFCB, and the protection-mode indicator is compatible with the processing mode. (If the data set name in the JFCB has been changed, a new password is requested from the operator.)

- The security indicator in the HDR1 label does not indicate password protection. (A new label will be written with the security indicator indicating password protection.)
- Only a volume label exists. (A HDR1 label will be written with the security indicator indicating password protection.)

Data Set Concatenation

A password is requested for every protected data set that is involved in a concatenation of data sets, regardless of whether the other data sets involved are protected or not.

SCRATCH and RENAME Functions

To delete or rename a protected data set, it is necessary that the job step making the request be able to supply the password. The system first checks to see if the job step is currently authorized to write to the data set. If not, message IEC301A is issued to request the password. The password provided must be associated with a "WRITE" protection-mode indicator.

Counter Maintenance

The operating system increments the counter in the password record on each usage, but no overflow indication will be given (overflow after 65,535 openings). You must provide a counter maintenance routine to check and, if necessary, reset this counter.

USING THE PROTECT MACRO INSTRUCTION TO MAINTAIN THE PASSWORD DATA SET

To use the PROTECT macro instruction, your PASSWORD data set must be on the system residence volume. The PROTECT macro can be used to:

- Add an entry to the PASSWORD data set.
- Replace an entry in the PASSWORD data set.
- Delete an entry from the PASSWORD data set.
- Provide a list of information about an entry in the PASSWORD data set; this list will contain the security counter, access type, and the 77 bytes of security information in the "data area" of the entry.

In addition, the PROTECT macro updates the DSCB of a protected direct access data set to reflect its protection status; this feature eliminates the need for you to use job control language whenever you protect a data set.

PASSWORD DATA SET CHARACTERISTICS AND RECORD FORMAT WHEN YOU USE THE PROTECT MACRO INSTRUCTION

When you use the PROTECT macro, the record format and characteristics of the PASSWORD data set are no different from the record format and characteristics that apply when you use your own routines to maintain it.

Number of Records for Each Protected Data Set

When you use the PROTECT macro, the PASSWORD data set must contain at least one record for each protected data set. The password (the last 8 bytes of the "key area") that you assign when you protect the data set for the first time is called the control password. In addition, you may create as many secondary records for the same protected data set as you need. The passwords assigned to these additional records are called secondary passwords. This feature is helpful if you want several users to have access to the same protected data set, but you also want to control the manner in which they can use it. For example: One user could be assigned a password that allowed the data set to be read and written, and another user could be assigned a password that allowed the data set to be read only.

Note: The PROTECT macro will update the protection-mode indicator in the format-1 DSCB in the protected data set only when you issue it for adding, replacing, or deleting a control password.

Protection-Mode Indicator

You can set the protection-mode indicator in the password record to four different values:

- X'00' to indicate that the password is a secondary password and the protected data set is to be read only (PWREAD).

- X'80' to indicate that the password is the control password and the protected data set is to be read only (PWREAD).

- X'01' to indicate that the password is a secondary password and the protected data set is to be read and written (PWREAD/PWWRITE).

- X'81' to indicate that the password is the control password and the protected data set is to be read and written (PWREAD/PWWRITE).

Because of the sequence in which the protection status of a data set is checked, the following defaults will occur:

If control password is:	Secondary password must be:
1. PWREAD/PWWRITE or PWREAD/NOWRITE	PWREAD/PWWRITE or PWREAD/NOWRITE
2. NOPWREAD/PWWRITE	NOPWREAD/PWWRITE

If the control password is set to either of the settings in item 1 above, the secondary password will be set to PWREAD/PWWRITE if you try to set it to NOPWREAD/PWWRITE.

If the control password is changed from either of the settings in item 1 to the setting in item 2 above, the secondary password will be automatically reset to NOPWREAD/PWWRITE.

If the control password is changed from the setting in item 2 to either of the settings in item 1 above, the secondary password is set by the system to PWREAD/PWWRITE.

Because the DSCB of the protected data set is updated only when the control password is changed, you may request protection attributes for secondary passwords that conflict with the protection attributes of the control password.

PROTECT MACRO SPECIFICATION

The format is:

[symbol]	**PROTECT**	parameter list address

parameter list address—A-type address, (2-12), or (1)
indicates the location of the parameter list. The parameter list must be set up before the PROTECT macro is issued. The address of the parameter list may be passed in register 1, in any of the registers 2 through 12, or as an A-type address. The first byte of the parameter list must be used to identify the function (add, replace, delete, or list) you want to perform. See Figure 26 on page 117 through Figure 29 on page 119 for the parameter lists and codes used to identify the functions.

0 X'01'	1 00 00 00
4 Length of data set name	5 Pointer to data set name
8 00	9 00 00 00
12 00	13 Pointer to control password
16 Number of volumes	17 Pointer to volume list
20 Protection code	21 Pointer to new password
24 String length	25 Pointer to string

0 X'01'
Entry code indicating ADD function.

4 Length of data set name.

5 Pointer to data set name.

13 Pointer to control password.
The control password is the password assigned when the data set was placed under protection for the first time. The pointer can be 3 bytes of binary zeros if the new password is the control password.

16 Number of volumes.
If the data set is not cataloged and you want to have it flagged as protected, you have to specify the number of volumes in this field. A zero indicates that the catalog information should be used.

17 Pointer to volume list.
If the data set is not cataloged and you want to have it flagged as protected, you provide the address of a list of volume serial numbers in this field. Zeros indicate that the catalog information should be used.

20 Protection code.
A one-byte number indicating the type of protection: X'00' indicates default protection (for the ADD function; the default protection is the type of protection specified in the control password record of the data set); X'01' indicates that the data set is to be read and written; X'02' indicates that the data set is to be read only; and X'03' indicates that the data set can be read without a password, but a password is needed to write into it. The PROTECT macro will use the protection code value, specified in the parameter list, to set the protection-mode indicator in the password record.

21 Pointer to new password.
If the data set is being placed under protection for the first time, the new password becomes the control password. If you are adding a secondary entry, the new password is different from the control password.

24 String length.
The length of the character string (maximum 77 bytes) that you want to place in the optional information field of the password record. If you don't want to add information, set this field to zero.

25 Pointer to string.
The address of the character string that is going to be put in the optional information field. If you don't want to add additional information, set this field to zero.

Figure 26. Parameter List for ADD Function

0 X'02'	1 00 00 00
4 Length of data set name	5 Pointer to data set name
8 00	9 Pointer to current password
12 00	13 Pointer to control password
16 Number of volumes	17 Pointer to volume list
20 Protection code	21 Pointer to new password
24 String length	25 Pointer to string

0 X'02'.
Entry code indicating REPLACE function.

4 Length of data set name.

5 Pointer to data set name.

9 Pointer to current password.
The address of the password that is going to be replaced.

13 Pointer to control password.
The address of the password assigned to the data set when it was first placed under protection. The pointer can be set to 3 bytes of binary zeros if the current password is the control password.

16 Number of volumes.
If the data set is not cataloged and you want to have it flagged as protected, you have to specify the number of volumes in this field. A zero indicates that the catalog information should be used.

17 Pointer to volume list.
If the data set is not cataloged and you want to have it flagged as protected, you have to provide the address of a list of volume serial numbers in this field. If this field is zero, the catalog information will be used.

20 Protection code.
A one-byte number indicating the type of protection: X'00' indicates that the protection is default protection (for the REPLACE function the default protection is the protection specified in the current password record of the data set); X'01' indicates that the data set is to be read and written; X'02' indicates that the data set is to be read only; and X'03' indicates that the data set can be read without a password, but a password is needed to write into the data set.

21 Pointer to new password.
The address of the password that you want to replace the current password.

24 String length.
The length of the character string (maximum 77 bytes) that you want to place in the optional information field of the password record. Set this field to zero if you don't want to add additional information.

25 Pointer to string.
The address of the character string that is going to be put in the optional information field of the password record. Set the address to zero if you don't want to add additional information.

Figure 27. Parameter List for REPLACE Function

0	X'03'	1	00 00 00
4	Length of data set name	5	Pointer to data set name
8	00	9	Pointer to current password
12	00	13	Pointer to control password
16	Number of volumes	17	Pointer to volume list

0 X'03'.
    Entry code indicating DELETE function.

4   Length of data set name.

5   Pointer to data set name.

9 Pointer to current password.
    The address of the password that you want to delete. You can delete either a control entry or a secondary entry.

13 Pointer to control password.
    The address of the password assigned to the data set when it was placed under protection for the first time. The pointer can be 2 bytes of binary zeros if the current password is also the control password.

16 Number of volumes.
    If the data set is not cataloged and you want to have it flagged as protected, you have to specify the number of volumes in this field. A zero indicates that the catalog information should be used.

17 Pointer to volume list.
    If the data set is not cataloged and you want to have it flagged as protected, you have to provide the address of a list of volume serial numbers in this field. If this field is zero, the catalog information will be used.

Figure 28. Parameter List for DELETE Function

0	X'04'	1	Pointer to 80-byte buffer
4	Length of data set name	5	Pointer to data set name
8	00	9	Pointer to current password

0 X'04'.
    Entry code indicating LIST function.

1 Address of 80-byte buffer.
    The address of a buffer where the list of information can be returned to your program by the macro instruction.

4   Length of data set name.

5   Pointer to data set name.

9 Pointer to current password.
    The address of the password of the record that you want listed.

Figure 29. Parameter List for LIST Function

RETURN CODES FROM THE PROTECT MACRO

When the PROTECT macro finishes processing, register 15 contains one of the following return codes:

Code	Meaning
0(00)	The updating of the PASSWORD data set was successfully completed.
4(04)	The PASSWORD of the data set name was already in the password data set.
8(08)	The password of the data set name was not in the PASSWORD data set.
12(0C)	A control password is required or the one supplied is incorrect.
16(10)	The supplied parameter list was incomplete or incorrect.
20(14)	There was an I/O error in the PASSWORD data set.
24(18)[1]	The PASSWORD data set was full.
28(1C)	The validity check of the buffer address failed.
32(20)[2]	The LOCATE macro failed. LOCATE's return code is in register 1, and the number of indexes searched is in register 0.
36(24)[2]	The OBTAIN macro failed. OBTAIN's return code is in register 1.
40(28)[2]	The DSCB could not be updated.
44(2C)	The PASSWORD data set does not exist.
48(30)[2]	Tape data set cannot be protected.
52(32)[2]	Data set in use.

[1] For this return code, a message is written to the console indicating that the PASSWORD data set is full.

[2] For this return code, the PASSWORD data set has been updated, but the DSCB has not been flagged to indicate the protected status of the data set.

CHAPTER 6. EXIT ROUTINES

This chapter discusses how installation-written modules can:

- Take control before and after direct access device storage management (DADSM) processing
- Take control during Open for a DCB
- Determine whether a missing data set control block (such as for a data set that has been moved to another volume) can be restored to a volume
- Recover from errors that may occur during the opening, closing, or handling of an end-of-volume condition for a data set associated with the user's task

This chapter also describes how user programs can:

- Identify a specific tape volume to be requested in place of a nonspecific (scratch) tape volume
- Verify that an IBM-standard labeled tape selected by open or EOV should, in fact, be used, and whether certain security checks may be bypassed (this exit is for authorized programs only)

Note: For information on IBM-supplied exits for tapes with International Organization for Standardization (ISO) or American National Standard labels, refer to MVS/370: Magnetic Tape Labels and File Structure.

DADSM PREPROCESSING AND POSTPROCESSING EXIT ROUTINES

There are exit routines to enable an installation-written module to take control before and after DADSM processing. An exit parameter list is used to communicate with DADSM. The format of this parameter list is shown in Figure 30 on page 123.

THE EXIT MODULES

All of the DADSM functions (allocate, extend, scratch, partial release, and rename) have a common preprocessing exit routine and a common postprocessing exit routine that the installation exit routine can replace. These exit routines enable you to gain control before and after DADSM processing. The preprocessing exit routine module is IGGPRE00; the postprocessing exit routine module is IGGPOST0. Each is used by all the DADSM functions listed above. The modules reside in SYS1.LPALIB. You can use System Modification Program (SMP) to replace the IBM-supplied exit routine modules with an installation exit routine you write.

THE EXIT ENVIRONMENT

The exit routines are given control in supervisor state and protect key zero with no locks held. The exit routines must be reentrant. System enqueues will have been issued either by DASDM or by the programs that invoke DADSM, to serialize system functions. These enqueues may prevent other system services from being invoked. In particular, dynamic allocation, OPEN, CLOSE, EOV, LOCATE, and other DADSM functions may not be issued because of an enqueue on the SYSZTIOT resource. If the exit routines require access to an installation data set, the control blocks required to access that data set (DCB, DEB) should be built during system initialization (IPL/NIP). RACF macros may be invoked from the exit routines.

WHEN IGGPRE00 GETS CONTROL

The preprocessing exit routine, IGGPRE00, is given control before the first VTOC update and after initial validity checking. Input to IGGPRE00 is a parameter list, mapped by macro IECIEXPL, that contains addresses of input data and a function code that identifies the DADSM function. IGGPRE00 is given control once for each volume in the volume list supplied to scratch and rename. A field in the parameter list, IEXRSVWD, may be used to pass data from the preprocessing exit routine to the postprocessing exit routine.

A zero return code from IGGPRE00 indicates the DADSM function may proceed.

REJECTING A DADSM REQUEST

A preprocessing exit routine may reject a DADSM request, in which case an I/O error return code is generated for all functions except allocate and extend. A return code of 4 or 8 from IGGPRE00 to allocate will cause allocate to return X'B4' or X'B0', respectively, to its caller in Register 15. Scheduler allocation will treat a X'B4' as a conditional rejection of the allocate request only for the volume being processed. If the allocate request is not for a specific volume, another volume may be chosen and the allocate function retried. Scheduler allocation will treat a X'B0' return code from allocate as an unconditional rejection of the allocate request. If the allocate request is rejected, the preprocessing exit routine can put a reason code in the parameter list field, IEXREASN, and the code will be returned by allocate to its caller, together with the X'B0' or X'B4' return code in Register 15. The reason code will appear in the JCL error message, if the allocate request is not retried. A nonzero return code from IGGPRE00 to extend will cause extend to return an error return code of X'FFFF FFEC' to its caller. If the caller is End-of-Volume, an E37-0C abend will be issued.

PASSING A MODEL FORMAT-1 DSCB

The preprocessing exit for allocate and extend on a new volume may return, in the parameter list field IEXFMT1, the address of the data portion of a model format-1 DSCB, starting with field DS1FMTID. The DSCB will be moved to the allocate or extend work area before building the format-1 DSCB. The only fields that may be nonzero in the area are the DS1REFD (the data-last-referenced field) and fields currently unused. All other fields will be initialized by allocate or extend. IEXFMT1 may not be supplied by IGGPRE00 for a VIO allocate request (indicated by flag, IEXVIO, set to one), or if a partial DSCB instead of a JFCB has been supplied to allocate (indicated by flag, IEXMF1, set to one). In the latter case, IEXFMT1 is passed to IGGPRE00 initialized to the address of the DS1FMTID field of the partial format-1 DSCB (supplied to allocate by its caller) in the allocate work area, and DS1REFD may be initialized by IGGPRE00. If extend was successful, IEXFMT1 is zeroed out prior to taking the post-exit, IGGPOST0.

WHEN IGGPOST0 GETS CONTROL

The postprocessing exit module, IGGPOST0, is given control after a DADSM function has been completed or attempted. IGGPOST0 is given control if IGGPRE00 was given control, whether the DADSM function was successful or not. IGGPOST0 is not given control if IGGPRE00 was not given control, or if the DADSM function terminated abnormally. IGGPRE00 may establish a recovery routine, if required, to clean up system resources. The DADSM recovery routine does not give IGGPOST0 control.

Input to IGGPOSTO is the same parameter list passed to IGGPRE00. No return codes from IGGPOSTO are defined.

Name	Offset	Bytes	Description
IEXID	00(00)	4	EBCDIC 'IEPL'
IEXLENG	04(04)	1	Length of parameter list
IEXFUNC	05(05)	1	DADSM function code:
IEXALL			X'01'-Allocate
IEXEXT			X'02'-Extend
IEXSCR			X'03'-Scratch
IEXPR			X'04'-Partial Release
IEXREN			X'05'-Rename
IEXEXTCD	06(06)	1	Extend code X'01' Extend data set on current volume X'02' Extend an OS catalog on current volume X'04' Extend data set on new volume X'81' Extend VSAM data space on current volume
IEXFLAG	07(07)	1	Flag byte
IEXENQ		1...	VTOC is enqueued upon entry
IEXVIO		.1..	VIO data set
IEXMF1		..1.	IEXFMT1 points to DX1FMTID of a partial format-1 DSCB (partial DSCB passed as input to Allocate, and not JFCB is not available).
*		...x xxxx	Reserved
IEXREASN	08(08)	2	Installation reject reason code
*		2	Reserved
IEXUCB	12(0C)	4	Address of UCB
IEXPTR1	16(10)	4	Address of the following: - JFCB (Allocate, Extend, Partial Release) - Scratch/Rename input parameter list (in user storage)
IEXPTR2	20(14)	4	Address of the following: - DSAB list (ISAM Allocate) - DEB (Extend on old volume) - DCB (Partial Release) - Current volume list entry (Scratch/Rename)
IEXDSN	24(18)	4	Address of the data set name
IEXFMT1	28(1C)	4	Address of the 96-byte data portion of format-1 DSCB (pre- and post-exit for partial release; post-exit for scratch). May be supplied by pre-exit of allocate, and extend on new volume, to serve as a model if IEXMF1 and IEXVIO are zero; postexit for allocate
IEXFMT2	32(20)	4	Address of format-2 DSCB (ISAM Allocate post exit)
IEXRSV00	36(24)	4	Reserved
IEXEXTBL	40(28)	4	Address of extent table (pre- and post-exit for scratch and partial release; post-exit for allocate and extend)
IEXDCC	44(2C)	4	DADSM completion code (post exit)
IEXRSVWD	48(30)	4	Reserved word for use by installation exit

Figure 30. Format of the DADSM Preprocessing and Postprocessing Exit Parameter List

SYSTEM CONTROL BLOCKS

The DADSM installation exit parameter list contains the address of system control blocks. The mapping macros of those control blocks are listed below together with the name of the system library in which they reside. One of the macros, ICVARXNT, is only supplied with the optional material.

Macro	Control Block	Location
IECSDSL1	DSCB	SYS1.AMODGEN
IEFUCBOB	UCB	SYS1.AMODGEN
IEFJFCBN	JFCB	SYS1.AMODGEN
IHADSAB	DSAB	SYS1.MACLIB
IEZDEB	DEB	SYS1.MACLIB
IHADCB	DCB	SYS1.MACLIB
IEFTIOT1	TIOT	SYS1.AMODGEN
ICVARXNT	Extent Table	Optional Material
IECIEXPL	DADSM installation exit parameter list	SYS1.MACLIB

There is no mapping macro for the SCRATCH/RENAME parameter list or the associated volume list.

For extend and partial release, the address of the JFCB passed to the user exit points to a copy of the real JFCB. Updating the copied JFCB will not result in a corresponding change to the real JFCB.

During EXTEND of a VSAM data set, the exit is passed the address of a dummy DEB. This DEB does not contain any EXTENT information.

REGISTERS AT ENTRY TO EXITS

At entry to your exit routine, register contents are as follows:

Register	Contents
1	Address of the exit parameter list
13	Address of an 18-word save area
14	Return address to DADSM
15	Address of your exit routine

REGISTERS AT RETURN TO DADSM

When you return to DADSM, register contents must be as follows:

Register	Contents
0-14	Same as on entry to your exit routine
15	A return code from IGGPRE00

The return codes and their meanings are as follows:

Code	Meaning
0	Indicates that you want the DADSM request to be processed
4	Indicates that no DADSM request for the current volume is to be processed
8	Indicates that you do not want the DADSM request to be processed

DCB OPEN INSTALLATION EXIT

There is an exit that enables an installation-written module to take control during Open for a DCB. An exit parameter list is used by open processing to communicate with the exit module. The format of the parameter list is shown in Figure 31 on page 126.

THE EXIT MODULE

OPEN has an exit module that the installation can replace. This module is IFG0EX0B, which resides in load module IGC0001I. The load module resides in SYS1.LPALIB. You can use System Modification Program (SMP) to replace the IBM-supplied exit module with an installation exit you write.

THE EXIT ENVIRONMENT

IFG0EX0B is given control in supervisor state and protect key zero with no locks held. System enqueues will have been issued to serialize system functions. These enqueues may prevent other system services from being invoked. In particular, dynamic allocation, OPEN, CLOSE, EOV, and DADSM functions should not be invoked because of an enqueue on the SYSZTIOT resource. If the exit requires access to an installation data set, the control blocks required to access that data set (DCB, DEB) should be built during system initialization (IPL/NIP). RACF macros may be invoked from the exit.

OPEN PROCESSING BEFORE IFG0EX0B GETS CONTROL

The exit module, IFG0EX0B, is given control whenever OPEN processes a DCB. The exit is taken after the following functions have been performed for the DCB.

- DASD data sets
  - Volume mounted
  - Format-1, -2, and -3 DSCBs read
  - Forward merge from format-1 DSCB to JFCB
- Tape data sets
  - Volume mounted
  - Header labels verified
  - Forward merge from header labels to JFCB
- All data sets
  - Forward merge from JFCB to DCB
  - User DCB OPEN exit (if any) taken
  - RACF or password verification processing

OPEN PROCESSING AFTER IFG0EX0B GETS CONTROL

The following functions have not yet been performed at the time the exit is given control for the DCB.

- Reverse merge from DCB to JFCB (not all fields are merged)
- Reverse merge from JFCB to format-1 DSCB for DASD data sets (not all fields are merged)
- Header labels written (for output tape data set)

- Access-method-dependent processing (obtain buffers, getmain and build IOBs and DEB)

- Write JFCB

- Write format-1 DSCB

GETTING CONTROL FROM OPEN

The exit is given control for each DCB being opened, even when two or more DCBs are being opened, in parallel, with one invocation of OPEN.

The exit is given control from OPEN (SVC 19) and OPEN TYPE=J (SVC 22). The exit is given control from end-of-volume (EOV; SVC 55) and from force-end-of-volume (FEOV; SVC 31) when a concatenation of two sequential data sets with unlike attributes is being processed. In this case, EOV gives control to CLOSE, which gives control to OPEN. The exit is not given control from EOV when a concatenation of two sequential data sets with like attributes is being processed. In this case, EOV does not give control to CLOSE and OPEN. A request by the user program for concatenation with unlike attributes is shown in the DCB by flag DCBOFPPC (bit 4; mask X'08') in field DCBOFLGS being set to one.

DATA THAT OPEN PASSES TO THE EXIT

The parameter list mapped by macro IECOIEXL is supplied to the installation exit. It contains data and the addresses of control blocks that may be of interest to the exit.

The format of the parameter list is shown in Figure 31.

Name	Offset	Bytes	Description
OIEXL	00(00)	0	DCB Open installation exit parameter list
OIEXOOPT	00(00)	1	Open option (last 4 bits).
OIEXRSVD		1111	X'F0' first 4 bits reserved.
OIEXOOUT		 1111	15 output
OIEXOOIN		.... .111	7 outin
OIEXOUPD		.... .1..	4 update
OIEXOINO		11	3 inout
OIEXORDB		1	1 read backward
OIEXOINP		....	0 input
OIEXUKEY	01(01)	1	User protect key. Key of user DCB.
OIEXLTH	02(02)	2	Length of OIEXL
OIEXUDCB	04(04)	4	Address of user DCB in user protect key (OIEXUKEY)
OIEXPDCB	08(08)	4	Address of protected copy of DCB used by OPEN
OIEXJFCB	12(0C)	4	Address of JFCB
OIEXDSCB	16(10)	4	Address of data portion of format-1 DSCB
OIEXTIOT	20(14)	4	Address of TIOT entry
OIEXUCB	24(18)	4	Address of UCB

Figure 31. Format of OPEN Exit Parameter List

Note that two DCB addresses are supplied. OPEN maintains a protected copy of the user DCB. OPEN's copy of the DCB may be used to test DCB fields. If any modification is made to the DCB, both the user DCB and OPEN's protected copy must be updated. The protect key of the user DCB is supplied in the exit parameter list. This protect key must be used to either fetch from or store into the user DCB.

Care should be taken to determine the type of DCB and device passed to the exit before testing access-method or device-dependent fields in the DCB. The sample exit shown in Appendix E gives an example of isolating a QSAM DCB being opened to a DASD or tape device.

The JFCB address supplied to the exit points to a copy of the JFCB that is in OPEN's work area. There may be other JFCBs associated with the OPEN, if ISAM or concatenated partitioned data sets are being opened.

In the case of BDAM, ISAM, and concatenated partitioned data sets, the UCB, whose address is supplied to the exit, may not be the only UCB associated with the DCB being opened. The UCB should not be modified.

The TIOT address supplied is of a TIOT entry (TIOENTRY label in the IEFTIOT1 macro). In the cases of ISAM and concatenated partitioned data sets, other TIOT entries may be associated with the DCB being opened. If concatenation of unlike attributes is being processed, the TIOT entry may have a blank DDNAME field.

The format-1 DSCB passed to the exit is in the OPEN work area. The address is of the field, DS1FMTID. There may be format-2 and -3 DSCBs associated with the format-1 DSCB. There may be other format-1 through -3 DSCBs associated with the DCB being opened in the cases of ISAM, BDAM, and concatenated partitioned data sets. If the OPEN is to the VTOC, a format-4 DSCB address is passed to the exit; this can be determined by testing field DS1FMTID for a value of X'F4', or the data set name in the JFCBDSNM field of 44X'04'.

DEFAULTING BUFFER NUMBER FOR QSAM

If a value has not yet been supplied, the exit may be used to supply an installation-determined value for DCBBUFNO (number of buffers) for QSAM DCBs.

A sample exit program that does this is shown in Appendix E.

It may not be advisable to override a nonzero value of DCBBUFNO without knowing what dependency the user program has on that value. DCBBUFNO can not be overridden when a buffer pool control block address exists in the DCB field, DCBBUFCA; this indicates buffers have been acquired before OPEN. DCBBUFCA is set to one (and not zero) if no buffer pool control block address exists.

MODIFYING THE JFCB

Whenever the JFCB is modified, code 4 should be returned to OPEN. This will cause OPEN to rewrite the JFCB. The JFCB should not be modified if the user program has set JFCNWRIT (bit 4) in byte JFCBTSDM as it indicates the JFCB should not be written.

A sample exit program that modifies the JFCB is shown in Appendix E.

REQUESTING PARTIAL RELEASE

An example of modifying the JFCB in OPEN's work area is used to set the bits to 1 indicating partial release has been requested: JFCRLSE (bits 0 and 1; mask X'C0') in byte JFCBIND1. This should be done only for DASD physical sequential or partitioned data sets opened for OUTPUT or OUTIN and processed by (1) EXCP with a 5-word device-dependent section present in the DCB, (2) BSAM, or (3) QSAM.

Care should be taken in modifying the JFCB release bits. For example, a data set that is opened for output many times, writing varying amounts of data each time, may have to extend after each OPEN, resulting in many small extents and, perhaps, reaching the 16-extent limit. This could result in a B37 abend.

Care should also be taken in setting the JFCBSPAC bits to define the space quantity units when the partial release flag, JFCBRLSE, is also set on. A cylinder allocated extent may be released on a track boundary when JFCBSPAC does not indicate cylinder units or average block length units with ROUND specified. This will cause the cylinder boundary extent to become a track boundary extent, thereby losing the performance advantage of cylinder boundary extents. Zeroing the release indicator and increasing secondary allocation quantity when, for example, the data set has extended a large number of times, may prevent such a B37 abend. Setting the release indicator could result in more space being made available to other users sharing the volume.

UPDATING THE SECONDARY SPACE DATA

The JFCB may also be modified by updating the secondary space data. Byte JFCBCTRI contains the space request type coded in the DD statement, or merged from the format-1 DSCB. Field JFCBSQTY contains the amount of secondary space (in either tracks, cylinders, or average block units). Field JFCBPQTY contains the amount of primary space (in either tracks, cylinders, or average block units).

Setting the contiguous bit (JFCONTIG) to zero may prevent an out-of-space ABEND where there is enough space, but not enough contiguous space, to satisfy a request to extend the data set.

REGISTERS AT ENTRY TO IFG0EX0B

At entry to the exit, register contents are as follows:

Register	Contents
1	Address of the DCB OPEN installation exit parameter list
13	Address of an 18 word save area
14	Return address to OPEN
15	Address of the entry point to IFG0EX0B

REGISTERS AT RETURN TO OPEN

When you return to OPEN, register contents are as follows:

Register	Contents
0-14	Same as on entry to the exit
15	Set to 4 if the JFCB has been modified. Set to 0 if the JFCB has not been modified

OPEN/EOV INSTALLATION EXIT FOR FORMAT-1 DSCB NOT FOUND

The function of the Format-1 DSCB-not-found installation exit in OPEN and EOV is to determine if a missing DSCB (such as a data set which has been migrated to another volume) can be restored to the volume. If your exit module restores the DSCB, it indicates this when it returns control to the control program. The exit module, IFG0EX0A, is given control whenever OPEN or EOV fails to find a format-1 DSCB on a volume. There is an

IBM-supplied exit module, IFG0EX0A, in SYS1.LPALIB. If you wish to use your own exit module, you must replace IFG0EX0A. Your exit module must have an entry point name of IFG0EX0A. If you do not write your own exit module, processing continues normally as the IBM-supplied exit returns a zero return code.

The exit is taken even under conditions where abnormal termination ordinarily would not occur. Two examples of these conditions follow:

1. When you have specified DISP=MOD and error recovery processing is taking place because the last volume specified in the JFCB does not contain the DSCB, but an earlier volume does. For this case, if your return code from IFG0EX0A is zero or if your return code is 4 and the DSCB has not been restored, OPEN and EOV search the other volumes for the DSCB after the exit is taken.

2. Another condition occurs during EOV output when space has not yet been allocated on the new volume. Space is allocated after the exit is taken if your return code from IFG0EX0A is zero or if your return code is 4 and the DSCB has not been restored.

When a DSCB is not found, IFG0EX0A is given control as follows:

- In system protect key 5 (data management key)
- In supervisor state
- The system resource represented by the SYSZTIOT major name is enqueued for shared control (this ENQ prevents the exit from invoking system functions such as SCRATCH, RENAME, dynamic allocation, or LOCATE).

Standard register linkage conventions are used when IFG0EX0A is given control as follows:

Register	**Contents**
0	If 0, entry was from OPEN (single volume data set) If C, entry was from OPEN (multivolume data set) If F, entry was from EOV
1	Address of parameter list
2-12	Unpredictable
13	Address of 18-word save area
14	Return address
15	Address of entry point IFG0EX0A

The parameter list pointed to by register 1 consists of two fullwords. The first fullword contains the address of the UCB for the volume on which the DSCB was not found. The second fullword contains the address of the 44-byte data set name, left justified, and padded with blanks. Bit zero of the second fullword is set to one, indicating the last word in the parameter list. The data set name must not be modified by the exit. The parameter list, save area, and data set name are in protect key 5 virtual storage, which is not fetch protected. IFG0EX0A must be reenterable. All work areas obtained through GETMAIN must be released through FREEMAIN. The return from your module, IFG0EX0A, to OPEN or EOV must be made as follows:

- Using the return address passed to you in register 14
- Registers 2-12 restored
- In protect key 5

- In supervisor state
- With a return code of 0, 4, or 8 in register 15

The return code you set in register 15 has the following meanings:

Code **Meaning**

0(00) Processing continues normally. This return code is given if the exit does not restore the DSCB. Zero is the return code always given by the IBM-supplied exit module.

4(04) The volume is searched one more time by OPEN or EOV for the DSCB. This return code is given if IFG0EX0A restores the DSCB to the volume. If the DSCB is again not found, IFG0EX0A is not given control and processing continues normally.

8(08) The task is abnormally terminated without attempting to determine if DISP=MOD error recovery or allocation on the new volume should occur. This return code is given if IFG0EX0A encounters an error and you wish no further processing to occur.

You should have IFG0EX0A establish its own error recovery environment (such as through an ESTAE), intercept any indeterminate errors, and return to the control program with return code 8. Problem determination is the responsibility of your exit module. A write-to-programmer (WTO with routing code 11) or a TPUT (if a TSO region) may be used to issue an informative message.

During a parallel OPEN when two or more DCBs are being opened at the same time, and two of the DCBs are opening the same data set, the DSCB may be missing. If IFG0EX0A is called for the first of the two DCBs and restores the DSCB, the channel program attempting to read the DSCB for the second DCB may have been executed before the restoration of the DSCB was complete. IFG0EX0A is then called for the second DCB even though the DSCB has already been restored. Return from IFG0EX0A with a return code 4 is appropriate in this case.

IFG0EX0A is not given control when you are processing a VSAM data set with an ACB; however, it is given control when you are processing a VSAM data space with a DCB. IFG0EX0A is bypassed if the format-4 DSCB is not found on a volume, even if the OPEN is to the VTOC data set name (data set name of 44 bytes of X'04').

DATA MANAGEMENT ABEND INSTALLATION EXIT

The abend installation exit provides the ability to recover from abnormal conditions that may occur during the opening, closing, or handling of an end-of-volume condition for a non-VSAM data set associated with the user's task.

When an abnormal condition occurs, control passes to the DCB abend user exit routine, if one is provided, and processing continues as specified in the DCB abend user exit routine. (The DCB abend user exit routine gives you some options regarding the actions you want the system to take when a condition arises that may result in abnormal termination of your task. For additional information about the DCB abend user exit routine, see _Data Management Services_.) However, if the DCB abend user exit routine is not specified, or specifies to abnormally terminate the task immediately, the system passes control to the abend installation exit. If a DCB abend user exit routine is not provided, control immediately passes to the abend installation exit.

IBM supplies an installation exit module, IFG0199I in SYS1.LPALIB, that handles abend situations caused by tape positioning errors. IFG0199I allows you to retry tape positioning when you receive a system completion code 613, return code 08 or 0C. To perform recovery actions for data management abend situations (other than those caused by tape positioning errors), you can replace installation exit module IFG0199I by modifying the source code supplied in SYS1.SAMPLIB.

IFG0199I receives control in protection key zero, supervisor state. IFG0199I checks the system completion code and the return code to determine whether the abend situation is the result of a tape positioning error. If the system completion code is other than 613 with return code 08 or 0C, control returns to the calling module with return code 0, indicating to continue with the abend. Otherwise, IFG0199I checks the counter in the 4-byte work area to determine if one attempt to reposition the tape has been made. If no attempt to reposition the tape has been made, IFG0199I issues a return code of 4, indicating to retry positioning. If one attempt to reposition the tape has been made, IFG0199I issues message IEC613A to the operator to determine whether to attempt repositioning. If the operator specifies that tape positioning is to be attempted again, a return code of 4 is set, indicating that OPEN rewind the tape and attempt positioning. If the operator specifies that tape positioning is not to be reattempted, control is returned to the calling module with a 0 return code.

When IFG0199I is given control, standard register linkage conventions are used for registers 1, 13, 14, and 15. IFG0199E passes an open abend installation exit list (OAIXL), in register 1, to the abend installation exit module.

The format of OAIXL follows:

Word Boundary

+0(00)	User Prot Key	Option Flats	Reserved	Reserved
+4(04)	Address of the protected copy of the DCB			
+8(08)	Address of the user's DCB Related to the abend			
+12(0C)	Address of the UCB Related to the abend			
+16(10)	Address of the JFCB Related to the abend			
+20(14)	Address of the TIOT Related to the abend			
+24(18)	Abend code — Example X'6130000C'			
+28(1C)	Installation work area (could be used as counter)			

0(00) Protection key of the user's DCB

1(01) Option flags:

Bits

0 OAIXEXIT; used to determine whether the DCB abend user exit was taken

On exit was taken
Off exit was not taken

1 OAIXREW; used to determine whether to rewind the tape volume

On rewind the tape volume
Off do not rewind the tape volume

8 Address of the user's DCB related to the abend used to distinguish each unique exit parameter list

28 4-byte work area used as a counter to determine the number of times tape positioning has been retried

The installation exit returns to IFG0199E one of the following return codes:

Code **Meaning**

0(00) Continue with the abend in process.

4(04) If the OAIXREW flag is set, indicating to rewind the tape, rewind the tape volume; set the UCBFSCT and UCBFSEQ fields in the UCB to zero; and retry the abend in process.

If the OAIXREW flag is not set, indicating not to rewind the tape, retry the abend in process.

For abend codes that the installation is allowed to retry, see Data Management Services in the section that defines the abend codes that the user abend exit may retry.

Modifying the IBM-Supplied Installation Exit Module: Because the IBM-supplied installation exit module only handles a particular abend situation, you may want to modify the source code of that module to perform corrective actions for other abend situations.

You can obtain a copy of the source code from SYS1.SAMPLIB for modification using the editing facility that is available to you. After you have modified the source code, link-edit it into SYS1.LPALIB. The source program is written in Assembler language, and uses only macros in SYS1.MACLIB. If you replace the supplied installation module, the exit module that you supply must have the entry point name IFG0199I and it must be reenterable.

OPEN/EOV USER EXIT FOR NONSPECIFIC TAPE MOUNT REQUESTS

This exit allows you to identify a specific tape volume to be requested in place of a nonspecific (scratch) volume. Only IBM-standard labeled tapes (SL) will be supported. The exit is invoked when open or EOV is to issue a mount request for a tape volume where no volume serial number has been specified, and will get control before the mount message is issued.

The exit address must be in the DCB exit list. The exit list entry code used to identify this exit in the DCB exit list is X'17'. The exit is called in user key; the state will be the same state as when the open or EOV was issued; no locks will be held.

At entry to your exit routine, register contents are as follows:

Register	Contents
0	Variable
1	Address of the exit parameter list (in key 5, nonfetch protected storage)
2-13	Contents before the macro instructions that gave Open/EOV control (OPEN, FEOV, EOV, PUT, and CHECK)
14	Return address (must not be altered by the exit routine)
15	Address of exit routine entry point

The conventions for saving and restoring register contents are as follows:

- The exit routine must preserve the contents of register 14. It need not preserve the contents of other registers. The operating system restores the contents of registers 2 through 13 before returning control to your program.

- The exit routine must not use the save area whose address is in register 13, because this area is used by the operating system. If the exit routine calls another routine or issues supervisor or data management macro instructions, it must provide the address of a new save area in register 13.

The exit parameter list contains the following:

- Flags indicating SL tape, first entry to the exit, and whether called from open or EOV

- The open option

- Addresses of the DCB, volume serial number, and JFCB

The high order bit of the last word in the list (the JFCB address) will be set to one.

The format of the parameter list, which is mapped by macro IECOENTE, is shown in Figure 32 on page 134.

The first time the exit is called (indicated by bit 7 of the first byte of the parameter list), the volume serial number field of the list contains a zero.

The following return codes (in register 15) are allowed:

Code	Meaning
0(00)	Open/EOV will continue with the nonspecific mount request.
4(04)	Open/EOV will use the user-specified volume. Register 0 contains the address of a 6-byte volume serial number. Open/EOV will request that the volume be mounted if the volume is not in use by this job or another job.

If open or EOV finds the supplied volume serial number is in use by this job or another job (that is, the volume is enqueued), the exit is taken a subsequent time (indicated by bit 7 of the first byte of the parameter list). The address of the supplied volume serial number is passed in the parameter list to the exit. The return codes will be the same as the first entry of the exit. The exit will be entered repetitively until return code 0 is passed back, or until return code 4 is passed back together with a volume serial number that is not in use.

Name	Offset	Bytes	Description
OENTWRD1	00(00)	0	Nonspecific tape request user exit parameter list
OENTFLG	00(00)	1	flags
OENTOEOV		0...	called by open
OENTOEOV		1...	called by EOV
		.xxx xxx.	reserved
OENTNTRY		0	first entry to exit
OENTNTRY		1	subsequent entry to exit
OENTOPTN	1 (01)	1	open options
		xxxx	reserved
		 0000	INPUT
		 1111	OUTPUT
		 0011	INOUT
		 0111	OUTIN
		 0001	RDBACK
	2 (02)	2	reserved
OENTDCBA	4 (04)	4	address of DCB
OENTVSRA	8 (08)	4	zero or address of volume serial number
OENTJFCB	12 (0C)	4	address of JFCB

Figure 32. Format of Parameter List for Nonspecific Tape Mount User Exit

If the tape volume is not in use, the exit will not be taken again even if some other reason (such as an I/O error, or invalid expiration date) causes the tape to be rejected.

OPEN/EOV USER EXIT FOR IBM-STANDARD LABELED TAPE SECURITY VERIFICATION

This exit allows authorized programs to verify that an IBM-standard labeled tape selected by open or EOV should, in fact, be used, and whether certain security checks may be bypassed. The exit supports only IBM-standard labeled tape (SL), and is taken only for APF authorized programs for which the program property "bypass password (and RACF) checking" is active for (for information on program properties, see System Programming Library: Job Management).

The exit address must be in the DCB exit list. The exit list entry code used to identify the exit is X'18'.

The exit is taken from open and EOV after volume verification and positioning, and before password and/or expiration date checking. The exit is called in user key; the state will be the same state as when the open or EOV was issued; no locks will be held.

At entry to your exit routine, register contents are as follows:

Register	Contents
0	Variable
1	Address of the exit parameter list (in key 5, nonfetch protected storage)
2-13	Contents before the macro instruction that gave Open/EOV control (OPEN, FEOV, EOV, PUT, CHECK, and GET)
14	Return address (must not be altered by the exit routine)
15	Address of exit routine entry point

The conventions for saving and restoring register contents are as follows:

- The exit routine must preserve the contents of register 14. It need not preserve the contents of other registers. The operating system restores the contents of registers 2 through 13 before returning control to your program.

- The exit routine must not use the save area whose address is in register 13, because this area is used by the operating system. If the exit routine calls another routine or issues supervisor or data management macro instructions, it must provide the address of a new save area in register 13.

The exit parameter list contains the following:

- Flags indicating whether the exit was called from open or EOV, and whether the first data set on the volume is to be written

- The open option

- Addresses of the DCB, volume serial number, tape label, and JFCB

The tape label is either the HDR1 label of a data set to be read forward or overwritten, the EOF1 label of a data set to be read backward, or the EOF1 label of a data set after which the new data set is to written. The high order bit of the last word in the parameter list (the JFCB address) is set to one.

The format of the parameter list, which is mapped by macro IECOEVSE, is shown in Figure 33.

Name	Offset	Bytes	Description
OEVSWRD1	00(00)	0	SL tape security verification user exit parameter list
OEVSFLG	00(00)	1	flags
OEVSOEOV		0...	called by open
OEVSOEOV		1...	called by EOV
		.xxx xxx.	reserved
OEVSFILE		0	first data set to be written on volume, or data set being read
OEVSFILE		1	second or subsequent data set to be written on volume
OEVSOPTN	01(01)	1	open options
		xxxx	reserved
		 0000	INPUT
		 1111	OUTPUT
		 0011	INOUT
		 0111	OUTIN
		 0001	RDBACK
	02(02)	2	reserved
OEVSDCBA	04(04)	4	address of DCB
OEVSVSRA	08(08)	4	address of volume serial number
OEVSHDR1	12(0C)	4	address of tape label (HDR1 or EOF1)
OEVSJFCB	16(10)	4	address of JFCB

Figure 33. Format of Parameter List for IBM-Standard Labeled Tape Security Verification User Exit

The following return codes (in register 15) are allowed:

Code **Meaning**

0(00) Use the volume as if the exit was not entered.

4(04) Output processing: reject the volume and request that a scratch tape be mounted (this will cause the open/EOV user exit for nonspecific tape volume mount requests to get control if that exit is defined). If the data set sequence number to be written is not 1, treat as return code 8.

Input processing: treat as return code 8.

Note: It is the user's responsibility to determine whether a data set open for INOUT or OUTIN is being processed for output or input at the time the exit is given control from EOV. Bit DCBOFLWR in field DCBOFLGS is set to 1 if the EOV is being processed for output.

8(08) Abnormally terminate the open or EOV, using the completion codes 913-34 for open and 937-29 for EOV.

12(0C) Use the volume; the password or expiration date of the tape label will not prevent the existing data set from being overwritten.

16(10) Use the volume. The password, expiration date of the tape label, or unlike data set names should not prevent the first data set on a volume from being written; however, in order to write other than the first data set, the data set must have the same security protection as the data set after which it will be written.

CHAPTER 7. SYSTEM MACRO INSTRUCTIONS

This chapter describes miscellaneous macro instructions that allow you to:

- Modify control blocks
- Obtain information from control blocks and system tables
- Perform track capacity calculations
- Allocate a data set based on a partial DSCB

Before reading this chapter, you should be familiar with the following publications:

- OS/VS-DOS/VSE-VM/370 Assembler Language contains the information necessary to code programs in the assembler language.
- Debugging Handbook contains format and field descriptions of the data areas referred to in this chapter.

INTRODUCTION

The system macro instructions are described in these functional groupings:

- Mapping (IEFUCBOB, IEFJFCBN, and CVT)
- Obtaining device characteristics (DEVTYPE)
- Manipulating the JFCB (RDJFCB)
- Data security (DEBCHK)
- Manipulating queues (PURGE and RESTORE)
- Performing track capacity calculations (TRKCALC)
- Allocating a data set based on a partial DSCB (REALLOC)

MAPPING SYSTEM DATA AREAS

The IEFUCBOB, IEFJFCBN, and CVT macro instructions are used as DSECT expansions that define the symbolic names of fields within the unit control block (UCB), job file control block (JFCB), and communication vector table (CVT), respectively.

The CVT, IEFUCBOB, and IEFJFCBN macro definitions are in a distribution library named SYS1.AMODGEN. Before you can issue the macros, you must copy them from SYS1.AMODGEN into SYS1.MACLIB (the IEBCOPY utility can be used to copy the macros), or SYS1.AMODGEN may be concatenated to the macro library before reference is made to SYS1.AMODGEN.

The fields in these blocks are shown and described in Debugging Handbook.

IEFUCBOB—MAPPING THE UCB

This macro instruction defines the symbolic names of the fields in the unit control block (UCB). The macro does not include a DSECT statement. However, if you specify PREFIX=YES, the DSECT statement is provided.

The format is:

[symbol]	IEFUCBOB	[LIST={NO|YES}] [,PREFIX={NO|YES}]

LIST={NO|YES}

NO
specifies that only the UCB prolog is to be printed.

YES
specifies that the UCB prolog and the rest of the UCB are to be printed.

PREFIX={NO|YES}

NO
specifies that no prefix is to be printed.

YES
specifies that the prefix and main body of the UCB are to be printed. A DSECT statement is included if you specify PREFIX=YES.

IEFJFCBN—MAPPING THE JFCB

This macro instruction defines the symbolic names of the fields in the job file control block (JFCB). The macro does not include a DSECT statement. If you require one, code a DSECT statement before the macro statement.

The format is:

[symbol]	IEFJFCBN	[LIST={NO|YES}]

LIST={NO|YES}

NO
specifies that only the JFCB prolog is to be printed.

YES
specifies that the JFCB prolog and the rest of the JFCB are to be printed.

CVT—MAPPING THE CVT

This macro instruction defines the symbolic names of all fields in the communication vector table (CVT).

The format is:

[symbol]	CVT	[DSECT={NO|YES}] [,LIST={NO|YES}]

DSECT={NO|YES}

NO
specifies that you do not want a DSECT.

YES
specifies that you want a DSECT.

LIST={NO|YES}

NO
specifies that only the CVT prolog is to be printed.

YES
specifies that the CVT prolog and the rest of the CVT are to be printed.

OBTAINING I/O DEVICE CHARACTERISTICS

Use the DEVTYPE macro instruction to request information relating to the characteristics of an I/O device, and to cause this information to be placed into a specified area. (The results of a DEVTYPE macro instruction executed before a checkpoint is taken should not be considered valid after a checkpoint/restart occurs.) The IHADVA macro maps the data returned by the DEVTYPE macro.

The topics that follow discuss the DEVTYPE macro, device characteristics, and particular output for particular devices.

DEVTYPE MACRO SPECIFICATION

The format is:

[symbol]	**DEVTYPE**	ddloc-addrx ,area-addrx **[,DEVTAB]** **[,RPS]**

ddloc-addrx
the name of an 8-byte field that contains the symbolic name of the DD statement to which the device is assigned. The name must be left justified in the 8-byte field, and must be followed by blanks if the name is less than eight characters. The doubleword need not be on a doubleword boundary.

area-addrx
the name of an area into which the device information is to be placed. The area can be two, five, or six fullwords, depending on whether or not the DEVTAB and RPS operands are specified. The area must be on a fullword boundary.

DEVTAB
This operand is only required for direct access devices. If DEVTAB is specified, the following number of words of information is placed in your area:

- For direct access devices: 5 words

- For nondirect access devices: 2 words

If you do not code DEVTAB, one word of information is placed in your area if the reference is to a graphics or teleprocessing device; for any other type of device, two words of information are placed in your area.

RPS
If RPS is specified, DEVTAB must also be specified. The RPS parameter causes one additional full word of RPS information to be included with the DEVTAB information.

Note: Any reference for a DUMMY data set in the DEVTYPE macro instruction will cause eight bytes of zeros to be placed in the output area. Any reference to a SYSIN or SYSOUT data set causes X'00000102' to be placed in word 0 and 32,760 (X'00007FF8') to be placed in word 1 in the output area. Any reference to a file allocated to a TSO terminal causes X'00000101' to be placed in word 0 and 32,760 (X'00007FF8') to be placed in word 1 in the output area.

DEVICE CHARACTERISTICS INFORMATION

The following information is placed into your area as a result of issuing a DEVTYPE macro:

Word 0

Describes the device as defined in the UCBTYP field of the UCB. For a complete description of this field, refer to Debugging Handbook.

Word 1

Maximum block size. For direct access devices, this value is the smaller of either the maximum size of an unkeyed block or the maximum block size allowed by the operating system; for magnetic or paper tape devices, this value is the maximum block size allowed by the operating system. For all other devices, this value is the maximum block size accepted by the device.

If DEVTAB is specified, the next three fullwords contain the following information about direct access devices:

Word 2

Bytes 0-1 The number of physical cylinders on the device, excluding alternates.

Bytes 2-3 The number of tracks per cylinder.

Word 3

Bytes 0-1 Maximum track length. Note that for the 2305, 3330/3333 Model 1 or 11, 3340/3344, 3350, 3375, and 3380 direct access devices, this value is not equal to the value in word 1 (maximum block size) as it is for other IBM direct access devices.

Note: Before using bytes 2 and 3, please read the description of word 4.

Byte 2 Block overhead, keyed block—the number of bytes required for gaps and check bits for each keyed block other than the last block on a track.

Byte 3 Block overhead—the number of bytes required for gaps and check bits for a keyed block that is the last block on a track.

Bytes 2-3 Block overhead—the number of bytes required for gaps and check bits for any keyed block on a track including the last block. Use of this form is indicated by a one in bit 4, byte 1 of word 4.

Basic overhead—the number of bytes required for the count field. Use of this form is indicated by a one in bit 3, byte 1 of word 4.

Word 4

Byte 0 Block overhead, block without key—the number of bytes to be subtracted from word 3, bytes 2 or 3 or bytes 2 and 3, if a block is not keyed.

If bit 3, byte 1 of word 4 is 1, this byte contains the modulo factor for a modulo device.

Byte 1

Bit 0 — If on, the number of cylinders, as indicated in word 2, bytes 0-1 are invalid. This bit will be on only for 3340 devices.

Bits 1-2 — Reserved.

Bit 3 — If on, indicates a modulo device (3375, 3380). To calculate the number of data bytes required for a data block for a modulo device, see the device data in Data Management Services.

Bit 4 — If on, bytes 2 and 3 of word 3 contain a halfword giving the block overhead for any block on a track, including the last block.

Bits 5-6 — Reserved.

Bit 7 — If on, a tolerance factor must be applied to all blocks except the last block on the track.

Bytes 2-3 — Tolerance factor—this factor is used to calculate the effective length of a block. The calculation should be performed as follows:

Step 1 — add the block's key length to the block's data length.

Step 2 — test bit 7 of byte 1 of word 4. If bit 7 is 0, perform step 3. If bit 7 is 1, multiply the sum computed in step 1 by the tolerance factor. Shift the result of the multiplication nine bits to the right.

Step 3 — add the appropriate block overhead to the value obtained above.

If bit 3, byte 1 of word 4 is 1, bytes (2-3) contain the overhead for the data or key field.

If DEVTAB and RPS are specified, the next fullword contains the following information:

Word 5

Bytes 0-1 — R0 overhead for sector calculations

Byte 2 — Number of sectors for the device

Byte 3 — Number of data sectors for the device

Figure 34 on page 142 shows the output for each device type that results from issuing the DEVTYPE macro.

Control is returned to your program at the next executable instruction following the DEVTYPE macro instruction. If the information concerning the ddname you specified has been successfully moved to your work area, register 15 will contain zeros. Otherwise, register 15 will contain X'04', indicating that the ddname was not found.

Device[1,2]	Maximum Record Size (Word 1, in Decimal)	DEVTAB (Words 2, 3, and 4, in Hexadecimal)	RPS (Word 5, in Hexadecimal)
2540 Reader	80	Not Applicable	Not Applicable
2540 Reader w/CI	80	Not Applicable	Not Applicable
2540 Punch	80	Not Applicable	Not Applicable
2540 Punch w/CI	80	Not Applicable	Not Applicable
2501 Reader	80	Not Applicable	Not Applicable
2501 Reader w/CI	80	Not Applicable	Not Applicable
2520 Reader-Punch	80	Not Applicable	Not Applicable
2520 Reader-Punch w/CI	80	Not Applicable	Not Applicable
1287 Optical Reader	80	Not Applicable	Not Applicable
1288 Optical Reader	80	Not Applicable	Not Applicable
3886 Optical Reader	80	Not Applicable	Not Applicable
3890 Document Processor	80	Not Applicable	Not Applicable
1419/1275 Reader/Sorter	80	Not Applicable	Not Applicable
3505 Reader	80	Not Applicable	Not Applicable
3505 Reader w/CI	80	Not Applicable	Not Applicable
3525 Punch	80	Not Applicable	Not Applicable
3525 Punch w/CI	80	Not Applicable	Not Applicable
1403 Printer	120[3]	Not Applicable	Not Applicable
1403 w/UCS	120[3]	Not Applicable	Not Applicable
1443 Printer	120[2]	Not Applicable	Not Applicable
3203 Model 5 Printer	132	Not Applicable	Not Applicable
3211 Printer	132[3]	Not Applicable	Not Applicable
3800 Printing Subsystem	136[4]	Not Applicable	Not Applicable
4245 Printer	132	Not Applicable	Not Applicable
2671 Paper Tape Reader	32760	Not Applicable	Not Applicable
1052 Printer-Keyboard	130	Not Applicable	Not Applicable
1053 Printer		Not Applicable	Not Applicable
3210 Printer-Keyboard	130	Not Applicable	Not Applicable

Figure 34 (Part 1 of 2). Output Obtained from Issuing DEVTYPE Macro

Device[1,2]	Maximum Record Size (Word 1, in Decimal)	DEVTAB (Words 2, 3, and 4, in Hexadecimal)	RPS (Word 5, in Hexadecimal)
3215 Printer-Keyboard	130	Not Applicable	Not Applicable
3895 Reader Inscriber	74	Not Applicable	Not Applicable
2400 (9-track)	32760	Not Applicable	Not Applicable
2400 (9-track, p.e.)	32760	Not Applicable	Not Applicable
2400 (9-track, d.d.)	32760	Not Applicable	Not Applicable
2400 (7-track)	32760	Not Applicable	Not Applicable
2400 (7-track, d.c.)	32760	Not Applicable	Not Applicable
2495 Tape Cartridge Reader	0	Not Applicable	Not Applicable
3400 (9-track, p.e.)	32760	Not Applicable	Not Applicable
3400 (9-track, d.d.)	32760	Not Applicable	Not Applicable
3400 (7-track)	32760	Not Applicable	Not Applicable
2314/2319 DAS Facility	7294	00CB00141C7E922D2D010216	Not Applicable
2305 Model 1 Fixed-Head Storage	14136	0030000838E8027ACA080200	02985A57
2305 Model 2 Fixed-Head Storage	14660	006000083A0A01215B080200	0140B4B1
3330/3333 Disk Storage	13030	019B0013336DBFBF38000200	00ED807C
3330V MSS Virtual Volume	13030	019B0013336DBFBF38000200	00ED807C
3330 Model 11 (or 3333 Model 11) Disk Storage	13030	032F0013336DBFBF38000200	00ED807C
3340 Disk Storage (35 megabytes)	8368	015D000C2157F2F24B000200	0125403D
3340/3344 Disk Storage (70 megabytes)	8368	0230001E4B36010B52080200	0125403D
3350 Disk Storage	19069	0230001E4B36010B52080200	0185807B
3375 Disk Storage	32760	03BF000C8CA000E0201000BF	0340C4BB
3380 Disk Storage	32760	0376000FBB6001002010010B	04E0DED6
2250 Model 1 Display Unit		Not Applicable	Not Applicable
2250 Model 3 Display Unit		Not Applicable	Not Applicable

Figure 34 (Part 2 of 2). Output Obtained from Issuing DEVTYPE Macro

Notes to Figure 34:

[1] CI—card image feature; d.c.—data conversion; d.d.—dual density; p.e.—phase encoding; UCS—universal character set; w/—with.

[2] Device codes are presented in System Programming Library: Debugging Handbook.

[3] Although certain models can have a larger line size, the minimum line size is assumed.

[4] The IBM 3800 Printing Subsystem can print 136 characters per line at 10-pitch, 163 characters per line at 12-pitch, and 204 characters per line at 15-pitch. The machine default is 136 characters per line at 10-pitch.

Communication Equipment	Record Size
1030,1050,83B3, TWX,2250,S360	Not Applicable
1060,115A,1130	Not Applicable
2780	Not Applicable
2740	Not Applicable

READING AND MODIFYING A JOB FILE CONTROL BLOCK

To accomplish the functions that are performed as a result of an OPEN macro instruction, the open routine requires access to information that you have supplied in a data definition (DD) statement. This information is stored by the system in a job file control block (JFCB).

In certain applications, you may find it necessary to modify the contents of a JFCB before issuing an OPEN macro instruction. For example, suppose you are adding records to the end of a sequential data set. You might want to add a secondary allocation quantity to allow the existing data set to be extended when the space currently allocated is exhausted. To assist you, the system provides the RDJFCB macro instruction. This macro instruction causes a specified JFCB to be moved from the SWA (scheduler work area), where it is stored, to an area specified in an exit list. (The use of the RDJFCB macro instruction with an exit list is shown under "RDJFCB—Read a Job File Control Block" on page 148. The symbolic names and field descriptions of the JFCB are contained in Debugging Handbook.) When you subsequently issue the OPEN macro instruction, you must indicate, by specifying the TYPE=J operand, that you want to open the data set using the JFCB in the area you specified.

At the conclusion of open processing, the JFCB is moved back to the SWA, unless you set the bit JFCNWRIT in the field JFCBTSDM to one before you issue the OPEN macro instruction.

Caution: If the JFCB, which the system used to open the data set, is not available in SWA during EOV or CLOSE processing, errors may occur.

Some of the modifications that are commonly made to the JFCB include:

- Moving the creation and expiration date fields of the DSCB into the JFCB (see "Using RDJFCB for MSS Virtual Volumes" below).
- Moving the secondary allocation quantity from the DSCB into the JFCB (see "Using RDJFCB for MSS Virtual Volumes" below).
- Moving the DCB fields from the DSCB into the JFCB.
- Adding volume serial numbers to the JFCB (see "Using RDJFCB for MSS Virtual Volumes" and "RDJFCB Security" below).

  Volume serial numbers in excess of five are written to the JFCBX (extension) located in the SWA. The JFCBX cannot be modified by user programs.
- Modifying the data set sequence number field in the JFCB.
- Modifying the number-of-volumes field in the JFCB (see "Using RDJFCB for MSS Virtual Volumes" below).
- Setting bit JFCDQDSP in field JFCBFLG3 to invoke the tape volume DEQ at demount facility (see "DEQ at Demount Facility for Tape Volumes," below).

USING RDJFCB FOR MSS VIRTUAL VOLUMES: Care must be taken in using RDJFCB if the data set resides on MSS virtual volumes such that:

- The expiration date added does not conflict with other volumes within the specified MSVGP.
- The secondary allocation quantity should be in cylinder increments and be a multiple or sub-multiple of the primary allocation quantity to avoid fragmentation.
- The number of volumes must not exceed the number available in the specified MSVGP.
- Any volume serial numbers added to the JFCB should exist in the MSVGP.

RDJFCB SECURITY: The volume serial numbers specified in the user-supplied JFCB will be compared with the volume serial numbers in the system JFCB located in the SWA. Each different volume serial number will be enqueued exclusively. The volumes will stay enqueued until the job step terminates since the close routines will not dequeue the volumes. If the job step already has the volume open, OPEN TYPE=J will continue. If the volume is enqueued by another job step, a 413 abend will occur with a return code of 04.

Some JFCB modifications can compromise the security of existing password-protected data sets. The following modifications are specifically not allowed, unless the program making the modifications is authorized or can supply the password:

- Changing the disposition of a password-protected data set from OLD or MOD to NEW.
- Changing the data set name of one or more of the volume serial numbers when the disposition is NEW.
- Changing the label processing specifications to bypass label processing.

Note: An authorized program is one that is either in supervisor state, executing in one of the system protection keys (keys 0 through 7), or authorized under the Authorized Program Facility.

RDJFCB USE BY AUTHORIZED PROGRAMS: If you change the data set name in the JFCB, you should do a system enqueue on the major name of "SYSDSN" for the substituted data set name. To use the correct interface with other system functions (for example, partial release), the ENQUEUE macro should include the TCB of the initiator and the length of the data set name (with no trailing blanks). When you complete processing of the data set, you should use the DEQ macro to release the resources.

If you rewrite the JFCB, you must set bit zero at JFCBMASK + 4 to one.

DEQ AT DEMOUNT FACILITY FOR TAPE VOLUMES

This facility is intended to be used by long-running programs which create an indefinitely long-running tape data set (such as a log tape). Use of this facility by such a program permits the processed volumes to be allocated to another job for processing (such as data reduction). This processing is otherwise prohibited unless the indefinitely long data set is closed and dynamically unallocated.

You may invoke this facility only through the RDJFCB/OPEN TYPE=J interface by setting bit JFCDQDSP (bit 0) in field JFCBFLG3 (offset 163 or X'A3') to 1. The volume serial of the tape is dequeued when the volume is demounted by OPEN or EOV with message IEC502E when all of the following conditions are present:

- The tape volume is verified for use by OPEN or EOV.
- JFCDQDSP is set to 1.
- The program is APF authorized (protect key and supervisor/problem state are not relevant).
- The tape volume is to be immediately processed for output. That is, either OPEN verifies the volume and the OPEN option is OUTPUT, OUTIN, or OUTINX; or EOV verifies the volume and the DCB is opened for OUTPUT, OUTIN, INOUT, or EXTEND, and the last operation against the data set was an output operation (DCBOFLWR is set to 1).

Note that in order for EOV to find JFCDQDSP set to 1, the program must not inhibit the rewrite of the JFCB by setting bit 4 of JFCBTSDM to 1.

The tape volume is considered verified after file protect, label type, and density conflicts have been resolved. The volume is dequeued when demounted after this verification, even if further in OPEN or EOV processing the volume is rejected because of expiration date, security protection, checkpoint data set protection, or an I/O error.

When the volume serial is dequeued, the volume becomes available for allocation to another job. However, because the volume DEQ is performed without unallocating the volume, care must be exercised both by the authorized program and the installation to prevent misuse of the DEQ at demount facility. A discussion of such misuse follows.

1. The authorized program must not close and reopen the data set using the tape volume DEQ at demount facility. If it does, one of the following can occur:

   a. The dequeued volume may be mounted and in use by another job. When the volume is requested for mounting, for the authorized program, the operator is unable to satisfy the mount. Therefore, the operator must either cancel the requesting job, cancel the job using the volume, wait for the requesting job to time out, or wait for the job using the volume to terminate.

b. The dequeued volume may be allocated to another job but not yet in use. The operator mounts the volume to satisfy the mount request of the authorized job. When the volume is requested for mounting by the other job, the operator is unable to satisfy the mount request, and is faced with the same choices as in a, above.

c. The dequeued volume may not yet be allocated to another job and the volume is mounted to satisfy the mount request of the authorized job. Another job may allocate the volume and when the volume is requested for mounting, the situation is the same as in b, above.

It is the responsibility of the installation that permits a program to run with APF authorization to ensure that it does not close and reopen a data set using the DEQ at demount facility.

2. Care should be exercised when an authorized program uses the DEQ at demount facility (data set 1) but processes another tape data set (data set 2). Assume the same volume serial numbers have been coded in the DD statements for data set 1 and data set 2. As the volumes of data set 1 are demounted, they are dequeued even though those volumes may yet be requested for data set 2. All of the problems explained in a, b, and c in 1 above, may occur as data set 2 and another job contend for a dequeued volume.

   This problem should not occur, given the intended use of the DEQ at demount facility. That is, a long-running application creating an indefinitely long tape data set. This type of application is not normally invoked through batch execution with user-written DD statements.

3. Once a volume has been demounted and dequeued because of the DEQ at demount facility, the volume is not automatically rejected by the control program when mounted in response to a specific or nonspecific mount request. Without the use of the facility, the control program can recognize (by the ENQ) that the volume is in use, and reject the volume. Therefore, operations procedures, in effect to prevent incorrect volumes from being mounted, should be reviewed in the light of reduced control program protection from such errors when the DEQ at demount facility is used. Specifically, if a volume is remounted for an authorized program and the volume had been used previously by that authorized program, duplicate volume serial numbers will exist in the JFCB and the control program will be unable to release the volume during EOV processing.

4. Checkpoint/restart considerations are discussed in Checkpoint/Restart.

OPEN—INITIALIZE DATA CONTROL BLOCK FOR PROCESSING THE JFCB

The OPEN macro instruction initializes one or more data control blocks so that their associated data sets can be processed.

A full explanation of the operands of the OPEN macro instruction, except for the TYPE=J option, is contained in Data Management Macro Instructions. The TYPE=J option, because it is used in conjunction with modifying a JFCB, should be used only by the system programmer or only under the system programmer's supervision.

[symbol]	OPEN	(dcb-addr,[(options)],...) [,TYPE=J]

TYPE=J
specifies that for each data control block referred to, you have supplied a job file control block (JFCB) to be used during initialization. A JFCB is an internal representation of information in a DD statement.

During initialization of a data control block, its associated JFCB may be modified with information from the data control block or an existing data set label or with system control information.

The system always creates a job file control block for each DD control statement. The job file control block is placed in the SWA (scheduler work area). Its position, in relation to other JFCBs created for the same job step, is noted in a table in virtual storage.

When this operand is specified, you must also supply a DD statement. However, the amount of information given in the DD statement is at your discretion because you can modify many fields of the system-created job file control block. If you specify DUMMY on your DD statement, the open routine will ignore the JFCB DSNAME and open the data set as dummy. (See the examples of the RDJFCB macro instruction for a coding example that modifies a system-created JFCB.)

Note: The DD statement must specify at least:

- Device allocation (refer to JCL for methods of preventing share status)
- A ddname corresponding to the associated data control block DCBDDNAM field

RDJFCB—READ A JOB FILE CONTROL BLOCK

The RDJFCB macro instruction causes a job file control block (JFCB) to be moved from the SWA (scheduler work area) into an area of your choice as identified via the EXLST parameter of RDJFCB for each data control block specified.

[symbol]	RDJFCB	(dcb-address ,[(options)],...)

dcb-address,(options)
(same as the dcbaddress, option1, and option2 operands of the OPEN macro instruction, as shown in Data Management Macro Instructions).

Although the option operands are not meaningful during the execution of the RDJFCB macro instruction, these operands can appear in the list form of either the RDJFCB or OPEN macro instruction to generate identical parameter lists, which can be referred to with the execute form of either macro instruction.

Examples: In Figure 35 on page 149, the macro instruction at EX1 creates a parameter list for two data control blocks: INVEN and MASTER. In creating the list, both data control blocks are assumed to be opened for input; option2 for both blocks is assumed to be DISP. The macro instruction at EX2 reads the system-created JFCBs for INVEN and MASTER from the SWA into the area you specified, thus making the JFCBs available to your problem program for modification. The macro instruction at EX3 modifies the parameter list entry for the data control block named INVEN and indicates, through the TYPE=J operand, that the problem program is supplying the JFCBs for system use.

```
EX1       RDJFCB (INVEN,,MASTER),MF=L
          .
          .
          .
EX2       RDJFCB MF=(E,EX1)
          .
          .
          .
EX3       OPEN (,(RDBACK,LEAVE)),TYPE=J,MF=(E,EX1)
          .
          .
          .
INVEN     DCB      EXLST=LSTA,...
MASTER    DCB      EXLST=LSTB,...
LSTA      DS       0F
          DC       X'07'
          DC       AL3(JFCBAREA)
          .
          .
          .
JFCBAREA  DS       0F,176C
          .
          .
          .
LSTB      DS       0F
          .
          .
          .
```

Figure 35. Sample Code Using RDJFCB Macro

Multiple data control block addresses and associated options may be specified in the RDJFCB macro instruction. This facility makes it possible to read several job file control blocks in parallel.

An exit list address must be provided in each data control block specified by an RDJFCB macro instruction. Each exit list must contain an active entry that specifies the virtual storage address of the area into which a JFCB is to be placed. A full discussion of the exit list and its use is contained in Data Management Services. The format of the job file control block exit list entry is as follows:

Types of Exit List Entry	Hexadecimal Code (High-Order Byte)	Contents of Exit List Entry (Low-Order Bytes)
Job file control block	07	Address of a 176-byte area to be provided if the RDJFCB or OPEN (TYPE=J) macro instruction is used. This area must begin on a fullword boundary and must be located within the user's region.

The virtual storage area into which the JFCB is read must be at least 176 bytes long.

The data control block may be open or closed when this macro instruction is executed.

If the JFCB is read successfully for all DCBs in the parameter list, a return code of zero is placed in register 15. If the JFCB is not read for any of the DCBs because the DDNAME is blank, or a DD statement is not provided, a return code of 4 is placed in register 15.

Warning: The following errors cause the results indicated:

Error	Result
A DD statement has not been provided.	A return code of 4 is placed in register 15.
DDNAME field in DCB is blank.	A write-to-programmer is issued, the request for this DCB is ignored, and a return code of 4 is placed in register 15.
A virtual storage address has not been provided.	Abnormal termination of task.

Note that if you want to open a VTOC data set to change its contents (that is, open it for OUTPUT, OUTIN, INOUT, UPDAT, OUTINX, or EXTEND), your program must be authorized under the Authorized Program Facility (APF). APF provides security and integrity for your data sets and programs. Details on how you authorize your program are provided in System Programming Library: Supervisor Services and Macro Instructions.

If the RDJFCB routine fails while processing a DCB associated with your RDJFCB request, your task is abnormally terminated. None of the options available through the DCB ABEND exit, as described in Data Management Services, is available when a RDJFCB macro instruction is issued.

When using concatenated data sets, the RDJFCB routine will modify only the first JFCB.

ENSURING DATA SECURITY BY VALIDATING THE DATA EXTENT BLOCK

Protecting one user's data from inadvertent or malicious access by an unauthorized user depends on protection of the data extent block (DEB). The DEB is a critical control block because it contains information about the device a data set is mounted on, and describes the location of data sets on direct access device storage volumes. The DEB also contains the address of the appendage vector table (AVT). Using the AVT, an unauthorized user can modify the AVT to give control to a routine in supervisor state to read from and write to data sets to which access would otherwise be denied.

To guarantee protection of the DEB, the DEBCHK macro instruction is provided. The DEBCHK macro instruction can be found in SYS1.MACLIB. The DEBCHK macro is issued by several components of the system control program. For example:

- The open access method executors issue the macro to add the address of a DEB they have built to a list of valid addresses called the DEB table. The DEB validity checking routine builds and maintains a DEB table for each job step.

- The EXCP Processor uses the macro to verify that the DEB passed with each EXCP request is in the DEB table.

- The close component issues the macro to remove a DEB from the DEB table.

If you code a routine that builds a DEB, you must add the address of the DEB you built to the DEB table. If you code a routine that depends on the validity of a DEB that is passed to your routine, you should verify that the DEB passed to your routine has a valid entry in the DEB table and points to your

DCB or access method control block (ACB). Use the **TYPE=ADD** and the **TYPE=VERIFY** operands of the macro, respectively.

To prevent an asynchronous routine from changing or deleting, or assigning a new DEB to a DCB, you must hold the local lock. In this case, you must use the branch entry to the DEBCHK verify routine.

Additional details about the functions provided by the DEB validity checking routine and about the contents of the DEB table are available in Open/Close/EOV Lcgic.

The DEBCHK macro instruction provides four functions:

- Adds the address of a DEB to the DEB table, which is located in protected storage. The DEB table contains the address of every user DEB associated with a given job step. Every system control program component that builds a user DEB must add the address of that DEB to a DEB table.

- Verifies that the DEB table associated with a given job step contains the address of a valid DEB and that the DEB points to the DCB (or ACB). Any system control program component or problem program can use this function to verify that a DEB is valid.

- Deletes the address of a DEB from the DEB table. Any program that deletes a user DEB must, before it deletes the DEB, issue a DEBCHK macro with a **TYPE=DELETE** operand to delete the address of the DEB from the DEB table. If the DEB validity checking routine encounters an error while deleting the address from the DEB table, the job step is abnormally terminated.

- Deletes the address of a DEB from the DEB table in the same way as the preceding function, except that, instead of terminating the job step, this function merely returns an error code in register 15. This function is provided to prevent recurring abnormal termination. The format of the DEBCHK and a description of the operands follow:

DEBCHK—MACRO SPECIFICATION

<table>
<tr><td>[symbol]</td><td><b>DEBCHK</b></td><td>cbaddr
[,TYPE={VERIFY|:ADD|DELETE|PURGE}]
[,AM={amtype|(amaddr)|((amreg))}]
[,BRANCH={NO|YES}]
[,TCBADDR=address]
[,KEYADDR=address]
[,SAVREG=reg]
[,MF=L]</td></tr>
</table>

cbaddr

for **BRANCH=NO**
RX-type address, (2-12), or (1)

A control block address passed to the DEBCHK routine. This operand is ignored if MF=L is coded. For verify, add, and delete requests, cbaddr is the address of a DCB or ACB that points to the DEB whose address is either verified to be in the DEB table, added to the DEB table, or deleted from the DEB table. For the purge function, cbaddr is the address of the DEB whose pointer is to be purged from the table: no reference is made to the DCB or ACB.

Note: A spooled DCB's DEB does not point back to the DCB, but to the spooled ACB; in this case, the DEBCHK should be issued against the ACB.

for **BRANCH=YES**

The A-type address of a 4-byte field, or a register (1) or (3-12), that points to the DCB or ACB containing the DEB to be verified.

TYPE={VERIFY|ADD|DELETE|PURGE}

indicates the function to be performed. If MF=L is coded, TYPE is ignored. The functions are:

VERIFY

This function is assumed if the **TYPE** operand is not coded. The control program checks the DEB table to determine whether the DEB pointer is in the table at the location indicated by the DEBTBLOF field of the DEB. The DEB is also checked to verify that DEBDCBAD points to the DCB (or ACB) passed to DEBCHK. The DEBAMTYP field in the DEB is compared to the AM operand value, if given. The two must be equal. **TYPE=VERIFY** can be issued in either supervisor or problem state.

ADD

The DEB and the DCB (or ACB) must point to each other before the DEB address can be added to the DEB table. Before the DEB pointer can be added to the table, the DEB itself must be queued on the current TCB DEB chain (the TCBDEB field contains the address of the first DEB in the chain). The DEB address is added to the DEB table at some offset into the table. That offset value is placed in the DEBTBLOF field of the DEB, and the access method type is inserted into the DEBAMTYP field of the DEB. A zero is placed in the DEBAMTYP field if the AM operand is not coded. **TYPE=ADD** can be issued only in supervisor state.

DELETE

The DEB and the DCB (or ACB) must point to each other before the DEB address can be deleted from the DEB table. **TYPE=DELETE** can be issued only in supervisor state.

PURGE

The DEB pointer is removed from the DEB table without checking the DCB (or ACB). **TYPE=PURGE** can be issued only in supervisor state.

AM

specifies an access method value. Each value corresponds to a particular access method type (note that BPAM and SAM have the same values):

Type	**Value**
TCAMAP	X'84'
SUBSYS	X'81'
ISAM	X'80'
BDAM	X'40'
SAM	X'20'
BPAM	X'20'
TAM	X'10'
GAM	X'08'
TCAM	X'04'
EXCP	X'02'
VSAM	X'01'
NONE	X'00'

The operand can be coded in one of the following three ways, only the first of which is valid for the list form (MF=L) of the instruction.

amtype
: refers to the access method: **ISAM, BDAM, SAM, BPAM, TAM** (which refers to BTAM only), **GAM, TCAM, EXCP,** or **VSAM. TCAMAP** identifies a TCAM application-program DEB. **SUBSYS** identifies a subsystem of the operating system, such as a job entry subsystem. **NONE** indicates that no access method or subsystem is specified.

(amaddr)
: is the RS-type address of the access method value. This format may not be coded when **MF=L** is used.

((amreg))
: is one of the general registers 1 through 14 that contains the access method value in its low-order byte (bit positions 24 through 31). The high-order bytes are not inspected. This form may not be used when **MF=L** is coded.

The use of amaddr and amreg should be restricted to those cases where the access method value has been generated previously by the **MF=L** form of DEBCHK. If **MF=L** is not coded, the significance of the AM operand depends upon the **TYPE.**

If **TYPE** is **ADD** and **AM** is specified, the access method value is inserted in the DEBAMTYP field of the DEB, and all subsequent DEBCHK macros referring to this DEB must either specify the same **AM** or omit the operand. When the **AM** operand is omitted for **TYPE=ADD**, a null value (0) is placed in the DEB and all subsequent DEBCHK macros must omit the **AM** operand.

If **AM** is specified when the **TYPE** is **PURGE, DELETE,** or **VERIFY,** the access method value is compared to the value in the DEBAMTYP field of the DEB. If **AM** is omitted, no comparison is made.

BRANCH={NO|YES}
specifies whether you want to use the branch entry to the DEBCHK verify routines.

NO
specifies branch entry is not to be used. The operands SAVREG, TCBADDR, and KEYADDR are ignored.

YES
specifies the branch entry is to be used. TYPE=VERIFY must be implicitly or explicitly specified. The operands TCBADDR and KEYADDR are required. AM and MF are ignored. Notes for BRANCH=YES:

- Registers 1, 2, 10, 11, 14, and 15 must not be used for SAVREG=.
- Registers 1, 2, 10, 11, 14, 15, and the register specified for SAVREG= must not be used for cbaddr, TCBADDR=, or KEYADDR=.
- The contents of registers 10, 11, and 14 are unpredictable on completion. Also, if you do not specify SAVREG=, the contents of register 2 are unpredictable.
- At completion time, register 1 contains the address of the DEB, and register 15 contains either 0, 4, or 16 (see below for codes and their meanings).

TCBADDR=address—A-type address or (3-12)
specifies the location or register containing the address of the TCB to be used by the DEBCHK verify routine. Use this operand only when BRANCH=YES.

KEYADDR=address—A-type address or (3-12)
specifies the location, or a register pointing to the location of a field containing the key to be used when accessing the DCB (or ACB). Use this operand only when BRANCH=YES.

SAVREG=reg
specifies the register in which register 2 is to be saved. Use this operand only when BRANCH=YES.

MF=L
indicates the list form of the DEBCHK macro instruction. When MF=L is coded, a parameter list is built consisting of the access method value that corresponds to the AM keyword. This value may be referenced by name in another DEBCHK macro by coding AM=(amaddr), or it may be inserted into the low-order byte of a register before issuing another DEBCHK macro by coding AM=((amreg)).

If the DEBCHK routine completes successfully, register 15 will be set to 0 and register 1 will contain the address of the DEB when control is returned to your program. Otherwise, register 15 will contain one of the following decimal codes:

Code	**Meaning**
4(04)	Either (a) the DEB table associated with the job step does not exist; or (b) the DEBTBLOF field of the DEB was set to zero or a negative number, or was larger than the DEB table; or (c) register 1 did not contain the same address as the DEB table entry.
8(08)	An invalid TYPE was specified. (The DEBCHK routine was entered by a branch, not by the macro.)
12(0C)	Your program was not authorized and TYPE was not VERIFY.
16(10)	DEBDCBAD did not contain the address of the DCB (or ACB) that was passed to the DEBCHK routine.
20(14)	The AM value does not equal the value in the DEBAMTYP field.
24(18)	The DEB is not on the DEB chain and TYPE=ADD was specified.
28(1C)	TYPE=ADD was specified for a DEB that was already entered in the DEB table.
32(20)	The DEB table exceeded the maximum size (32,760 bytes) and TYPE=ADD.

PURGING AND RESTORING I/O REQUESTS

The system's purge routines, guided by a parameter list you pass them, perform either a halt or a quiesce operation. In a halt operation, the purge routines stop the processing of specified I/O requests that were initiated with an EXCP macro instruction. In a quiesce operation, the purge routines:

- Allow the completion of I/O requests that were initiated with an

  passed to the I/O supervisor for execution

- Stop the processing of those requests that have not as yet been passed to the I/O supervisor, but save the IOBs of the requests so that they can be reprocessed (restored) later.

The system's restore routines make it possible to reprocess I/O requests that are quiesced. (**Note**: Not covered here is the purge and restore processing that takes in I/O requests not initiated by an EXCP macro instruction. If you want to know the full scope of purge and restore processing, see _I/O Supervisor Logic_.)

You can give control to the purge and restore routines in two ways: (1) by loading register 1 with the address of the parameter list and issuing specific SVC instructions or (2) by issuing the PURGE and RESTORE macro instructions. If your installation requires the use of macro instructions, you must add the macro definitions to the macro library (SYS1.MACLIB) or place them in a partitioned data set and concatenate this data set to the macro library. The macro definitions, JCL, and utility statements needed to add the macros to your macro library are presented in Figure 36 , and Figure 37 on page 156. Whether you issue the macro instructions or the SVC instructions, you must first build a parameter list. The SVC instructions are SVC 16 for PURGE and SVC 17 for RESTORE.

PURGE Macro Definition

```
      MACRO
&NAME PURGE       &LIST
      AIF         ('&LIST' EQ '').E1
&NAME IHBINNRA    &LIST             LOAD REG 1
      SVC         16
      MEXIT
.E1   IHBERMAC    01,147            LIST ADDR MISSING
      MEND
```

Control Statements Required

```
//jobname    JOB        {parameter}
//stepname   EXEC       PGM=IEBUPDTE,PARM=NEW
//SYSPRINT   DD         SYSOUT=A
//SYSUT2     DD         DSNAME=SYS1.MACLIB,DISP=OLD
//SYSIN      DD         *
./  ADD      NAME=PURGE,LIST=ALL
             .
             .
             .
             PURGE macro definition
             .
             .
             .
./  ENDUP
/*
```

Figure 36. Macro Definition, JCL, and Utility Statements for Adding PURGE Macro to the System Macro Library

RESTORE Macro Definition

```
       MACRO
&NAME  RESTORE      &LIST
       AIF          ('&LIST' EQ '').E1
&NAME  IHBINNRA     &LIST              LOAD REG 1
       SVC          17                 ISSUE SVC FOR RESTORE
       MEXIT
.E1    IHBERMAC     01,150             LIST ADDR MISSING
       MEND
```

Control Statements Required

```
//jobname     JOB        {parameters}
//stepname    EXEC       PGM=IEBUPDTE,PARM=NEW
//SYSPRINT    DD         SYSOUT=A
//SYSUT2      DD         DSNAME=SYS1.MACLIB,DISP=OLD
//SYSIN       DD         DATA
./  ADD       NAME=RESTORE,LIST=ALL
                .
                .
                .
              RESTORE macro definition
                .
                .
                .
./  ENDUP
/*
```

Figure 37. Macro Definition, JCL, and Utility Statements for Adding RESTORE Macro to the System Macro Library

PURGE—HALT OR FINISH I/O-REQUEST PROCESSING

The macro instruction used to call the purge routines is coded as follows:

[symbol]	PURGE	parameter-list address

parameter list address—RX-type address, (2-12) or (1)
address of a parameter list, 12 or 16 bytes long, that you have built on a fullword boundary in your storage. The parameter list address can be specified as an RX-type constant or in registers 2 through 12 or 1.

The format and contents of the parameter list are as follows:

Byte **Contents**

0 A byte in which you specify what the purge routines will do. These are the bit settings and their meanings:

1...	Purge I/O requests to a single data set.
0...	Either purge I/O requests associated with a TCB or address space, or purge I/O requests to more than one data set.
.1..	Post ECBs associated with purged I/O requests.
..1.	Halt I/O-request processing. (Quiesce I/O-request processing, if 0.)
...1	Purge related requests only. (Valid only if a data-set purge is requested.)

Byte **Contents**

.... 0... Reserved—must be zero.

.... .1.. Do not purge the TCB's request-block chain of asynchronously scheduled processing.

.... ..1. Purge I/O requests associated with a TCB.

....1 This is a 16-byte parameter list. Additional purge options are specified in bytes 12 to 15. (If this bit is off, the list is 12 bytes long, and the purge routines do not put a return code in byte 4 of this list or in register 15.)

1,2,3 The address of a DEB if you're purging I/O requests to a single data set. The address of the first DEB in a chain of DEBs if you're purging I/O requests to more than one data set. (The next-to-the-last word of each DEB must point to the next DEB in the chain; the second word of the last DEB must contain zeros.)

4 A byte of zeros. (If bit 7 of byte 0 is on, the purge routines will put a code in this byte: X'7F' if the purge operation is successful; X'40' if it is not successful.)

5,6,7 The address of the TCB associated with the I/O requests you want purged (but only if you turned on bit 6 of byte 0). May be zeros if the TCB is the one you're running under.

8 A byte of zeros.

9,10,11 The address of a word in your storage or the address of the DEBUSPRG field (which is X'11' bytes more than the DEB address in this parameter list). At whichever address you specify, the purge routines store a pointer to the purged I/O restore list, PIRL. In the PIRL is a pointer to the first IOB in the chain of IOBs. The location of the pointer and format of the chain are shown in Figure 38 on page 159 .

12 A byte in which you can specify additional purge options. These are the bit settings and their meanings:

Note: The following applies only if bit 7 of byte 0 is set to one.

..1. .... Purge I/O requests associated with an address space. (You must be in supervisor state.)

...1 .... Check the validity of all the DEBs associated with the purge operation if this is a data-set purge. Validate this parameter list, whatever the type of purge operation, by ensuring that there are no inconsistencies in the selection of purge options. (If the caller is in problem state, these actions are taken regardless of the bit setting.)

.... 1... Ensure that I/O requests will be reprocessed (restored) under their original TCB. (If zero and this byte is meaningful (bit 7 of byte 0 is on), the I/O requests will be reprocessed under the TCB of the program making the restore request.)

.... .0.. Must be zero.

Byte **Contents**

13 A byte of zeros.

14,15 The two-byte ID of the address space associated with the I/O requests you want purged. (Only meaningful if bit 2 of byte 12 is on.)

Control will be returned to your program at the instruction following the PURGE macro instruction. If the purge operation was successful, register 15 will contain zeros. Otherwise, register 15 will contain one of the following return codes:

Code **Meaning**

4(04) Your request to purge I/O requests associated with a given TCB was not honored because that TCB did not point to the job step TCB, as it must when the requestor is in problem state.

8(08) Either you requested an address-space purge operation but were not in supervisor state, or you requested a data-set purge operation but supplied no data-area address in bytes 1, 2, and 3 of the purge parameter list.

20(14) Another purge request has preempted your request. You may want to reissue your purge request in a time-controlled loop.

Note: Register 15 will contain zeros, regardless of the outcome of the purge operation, if you set bit 7 in byte 0 of the parameter list to zero.

MODIFYING THE IOB CHAIN

Note, it is not a recommended procedure but, if you want to change the order in which purged I/O requests will be restored or prevent a purged request from being restored, you may change the sequence of IOBs in the IOB chain or remove an IOB from the chain. The address of the IOB chain can be obtained from the PIRL (see Figure 38 on page 159 . (The address of the PIRL will be at the location pointed to by bytes 9 through 11 of the purge parameter list.)

RESTORE—REPROCESS I/O REQUESTS

The RESTORE macro is coded as follows:

[symbol]	**RESTORE**	restore address

restore address—RX-type address, (2-12) or (1)
address you specified at byte 9 of the purge parameter list.

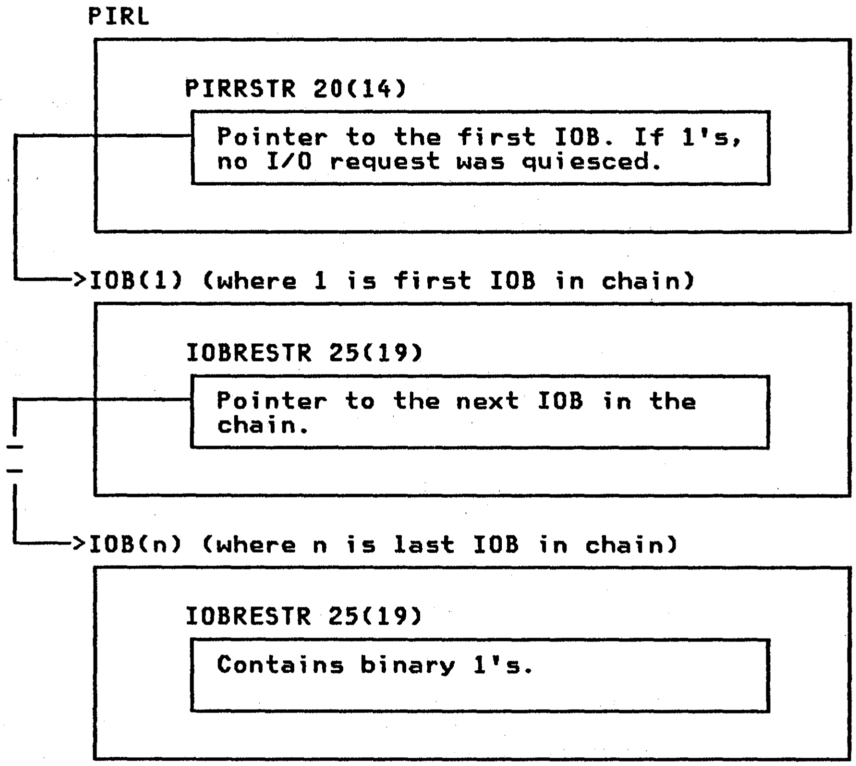

PIRL
PIRRSTR 20(14)
Pointer to the first IOB. If 1's, no I/O request was quiesced.
>IOB(1) (where 1 is first IOB in chain)
IOBRESTR 25(19)
Pointer to the next IOB in the chain.
>IOB(n) (where n is last IOB in chain)
IOBRESTR 25(19)
Contains binary 1's.

Figure 38. The PIRL and IOB Chain

PERFORMING TRACK CALCULATIONS

The TRKCALC macro performs track capacity calculations. The standard, list, execute, and DSECT form of the macro are described. Examples of the TRKCALC macro follow the macro descriptions. Using TRKCALC you may do the following:

- Perform track capacity calculations
- Determine the number of records of a given size which can be written on a fulltrack or the remainder of a track
- Perform track balance calculations as follows:
  - Determine if a given record size can be written in the space remaining on the track and return the new track balance.
  - Determine the maximum size record which can be written on the track if the given record does not fit.
  - Determine the track balance if the last physical record is removed from the track.

TRKCALC—STANDARD FORM

The format of the TRKCALC macro is:

[symbol]	TRKCALC	FUNCTN={TRKBAL|TRKCAP} {,DEVTAB=addr|,UCB=addr|,TYPE=addr} [,BALANCE=addr] [,REMOVE={YES|NO}] [,MAXSIZE={YES|NO}] {,RKDD=addr|,R=addr,K=addr,DD=addr} [,REGSAVE={YES|NO}] [,MF=I]

FUNCTN={TRKBAL|TRKCAP}
specifies the function to be performed.

Note: You must specify one of the three keywords, DEVTAB, UCB, or TYPE, to provide the macro a source for information.

TRKBAL
if REMOVE=NO is specified, TRKBAL calculates whether an additional record fits on the track, and what new track balance would be if the record were added. If REMOVE=YES is specified, TRKBAL calculates what the track balance would be if a record were removed from the track. The record to be added or removed from the track is defined by the RKDD parameter, or by the R, K, and DD parameters.

If R=1 (or the R value in the RKDD parameter is 1) and REMOVE=NO is specified, record 1 is added to an empty track; if R=1 and REMOVE=YES is specified, record 1 is deleted from the track, leaving an empty track.

If R≠1, the specified record is added to or removed from the track. The input track balance may be supplied through the BALANCE parameter; if it is not supplied, it is assumed that the track contains equal sized records as specified in the RKDD parameter (or R, K, and DD parameters).

When REMOVE=NO is specified, one of the following occurs:

- If the record fits on the track, register 0 contains the new track balance.
- If the record does not fit on the track and MAXSIZE=NO is specified, a "record does not fit" return code is given in register 15.
- If the record does not fit and MAXSIZE=YES is specified, one of the following happens:
  - The data length of the largest record that fits in the remaining space is returned in register 0.
  - A code is returned that indicates no record fits in the remaining space.

When REMOVE=YES is specified, one of the following occurs:

- If R=1, register 0 contains the track capacity.
- If R≠1, registers 0 contains the input track balance (supplied through the BALANCE parameter) incremented by the track balance used by the input record. If the input balance is not supplied,

register 0 contains the track capacity left after R-1 records are written on the track.

TRKCAP
calculates, and returns in register 0, the number of fixed length records that may be written on a whole track (R=1) or on a partially filled track (R≠1). The records are defined by the K and DD values of the RKDD parameter, or by the K and DD parameters.

One of the following occurs:

- If R=1, the BALANCE parameter is ignored and the calculation is made on an empty track.
- If R≠1 and the BALANCE parameter is omitted, the calculation is made for a track that already contains R-1 records of the length defined by the K and DD values.
- If R≠1 and the BALANCE parameter is supplied, the calculation is made for a track whose remaining track balance is the value of the BALANCE parameter.

DEVTAB=addr—RX-type address, (2-12), (0), (14)
addr specifies a word that contains the address of the Device Characteristics Table Entry (DCTE). If you specify a register, it contains the address of the DCTE, not the address of a word containing the address of the DCTE. The address of the DCTE can be found in the DCBDVTBA field of an opened DCB.

UCB=addr—RX-type address, (2-12), (0), (14)
addr specifies a word that contains the address of the UCB. If you specify a register, it contains the address of the UCB, not the address of a word containing the address of the UCB.

TYPE=addr—RX-type address, (2-12), (0), (14)
you may specify the address of the UCB device type (UCBTBYT4), or you may specify the one-byte UCB device type in the low-order byte of a register.

BALANCE=addr—RX-type address, (2-12), (0), (14)
you may specify either the address of a halfword containing the current track balance, or you may specify the balance in the low-order two bytes of a register. The value supplied may be the value returned when you last issued TRKCALC. If R=1, the balance is reset to track capacity by TRKCALC and your supplied value is ignored. This is an input value and is not modified by the TRKCALC macro. The resulting track balance is returned in register 0 and in the TRKCALC parameter list field STARBAL.

REMOVE={YES|NO}
indicates if a record is to be deleted from the track.

YES
specifies the record number (specified in the R keyword) is being removed from the track. The track balance is incremented instead of decremented.

Note: YES is valid only on a FUNCTN=TRKBAL call.

NO
specifies a record is not to be deleted from the track. NO is the default.

MAXSIZE={YES|<u>NO</u>}

YES
If the specified record does not fit, the largest length of a record with the specified key length that fits is returned (register 0).

Note: YES is valid only on a FUNCTN=TRKBAL call.

NO
Maximum size is not returned. NO is the default.

RKDD=<u>addr</u>—RX-type address, (2-12), (0), (14)
addr specifies a word containing a record number (1 byte), keylength (1 byte), and data length (2 bytes) (bytes 0, 1, and 2 and 3, respectively) or a register containing the record number, key length, and data length. R, K, and DD may be specified by this keyword, or you may use the following three keywords instead.

R=<u>addr</u>—RX-type address, (2-12), (0), (14), or n
you may specify either the address of the record number, or you may specify the record number using the low-order byte of a register or immediate data (n). Specify a decimal digit for n (immediate data).

K=<u>addr</u>—RX-type address, (2-12), (0), (14), or n
you may specify either the address of a field containing the hex value of the record's key length, or you may specify the record's key length using the low-order byte of a register or :mmediate data (n). Specify a decimal digit for n (immediate data).

DD=<u>addr</u>—RX-type address, (2-12), (0), (14), or n
you may specify either the address of a field containing the hex value of the record's data length, or you may specify the record's data length using the low-order two bytes of a register or immediate data (n). Specify a decimal digit for n (immediate data).

REGSAVE={YES|<u>NO</u>}

YES
specifies registers 1 through 14 are saved and restored in the caller-provided save area (pointed to by register 13) across the TRKCALC call. Otherwise, registers 1, 9, 10, 11, and 14 are modified. Registers 0 and 15 are always modified by a TRKCALC call.

NO
specifies registers are not saved across a TRKCALC call. NO is the default.

MF=I
specifies to define the storage for the TRKCALC parameter list and initialize the parameter list using the given keywords and call the TRKCALC function. MF=I is the default.

INPUT REGISTER USAGE FOR ALL FORMS OF 'MF'

Registers 0, 2-12, and **14** are available to provide input for keywords.

Register 1 is used only to provide the address of the parameter list for an MF=E call.

Register 13 may be used as input for keywords, if REGSAVE=YES is not specified.

Register 15 is used as a work register to build the TRKCALC parameter list for the MF=E call; it is not available as an input register.

OUTPUT FROM TRKCALC

FUNCTN=TRKBAL

Register 15=0
: The record fits on the track. Register 0 and STARBAL contain the new track balance.

Register 15=4
: Record does not fit on the track. If MAXSIZE=YES is specified, a partial record does not fit either. Register 0 and STARBAL are set to zero.

Register 15=8
: Record does not fit on the track. MAXSIZE=YES is specified and a partial record does fit. Register 0 and STARBAL are set to the maximum number of data bytes that fit on the remainder of the track with the specified keylength.

Note: The keylength is excluded from the count of maximum data bytes.

STARBAL
: This is the track balance field of the TRKCALC parameter list. This field is first set to the track capacity if R=1, or to the supplied BALANCE value if R≠1, or to the calculated balance if R≠1 and BALANCE is omitted. STARBAL is updated to the new track balance if the record fits; otherwise, STARBAL is left with the input track balance value.

FUNCTN=TRKCAP

Register 15=0
: Register 0 contains the number of records that fit on the track if R = 1, or the number of records that fit on the remainder of the track if R ≠ 1.

Register 15=4
: No records of the length specified fit on a full track (R = 1) or a partial track (R ≠ 1). Register 0 is set to zero.

STARBAL
: This is the track balance field of the TRKCALC parameter list. This field is first set to the track capacity if R=1, or to the supplied BALANCE value if R≠1, or to the calculated balance if R≠1 and BALANCE is omitted.

TRKCALC—LIST FORM

The list form of the TRKCALC macro is used to construct an empty, in-line parameter list. By coding only MF=L you construct a parameter list and the actual values can be supplied by the execute form of the TRKCALC macro. Any parameters other than MF=L are ignored.

[symbol]	TRKCALC	MF=L

TRKCALC—EXECUTE FORM

A remote parameter list is referred to and can be modified by the execute form of the TRKCALC macro. The TRKCALC routine is called. The description of the standard form of the macro provides the explanation of the function of each operand.

[symbol]	TRKCALC	[FUNCTN={TRKBAL|TRKCAP}] [{,DEVTAB={addr|*}| ,UCB={addr|*}|,TYPE={addr|*}}] [,BALANCE={addr|*}] [,REMOVE={YES|NO}] [,MAXSIZE={YES|NO}] [{,RKDD=addr|,R=addr,K=addr,DD=addr}] [,REGSAVE={YES|NO}] ,MF=(E,addr)

FUNCTN={TRKBAL|TRKCAP}
it is coded as shown in the standard form. If this keyword is omitted, any specification of REMOVE, MAXSIZE, LAST, and the RX form of BALANCE, is ignored. In addition, DEVTAB is assumed, if UCB is coded and a failure occurs, if TYPE is specified. When you use FUNCTN, one of the keywords (DEVTAB, UCB, or TYPE) must be specified to provide an information source.

DEVTAB=addr|*—RX-type address, (2-12), (0), (14)
it is coded as shown in the standard form except for the * subparameter. Specify an * when you have inserted the address of the Device Characteristics Table Entry (DCTE) in the parameter list.

UCB=addr|*—RX-type address, (2-12), (0), (14)
it is coded as shown in the standard form except for the * subparameter. Specify an * when you have inserted the address of the UCB in the parameter list.

TYPE=addr|*—RX-type address, (2-12), (0), (14)
it is coded as shown in the standard form except for the * subparameter. Specify an * when you have inserted the address of the UCB type (UCBTYP) in the parameter list.

BALANCE=addr|*—RX-type address, (2-12), (0), (14)
it is coded as shown in the standard form except for the * subparameter. Specify an * when you have inserted the balance in the parameter list.

REMOVE={YES|NO}
it is coded as shown in the standard form.

MAXSIZE={YES|NO}
it is coded as shown in the standard form.

RKDD=addr—RX-type address, (2-12), (0), (14)
it is coded as shown in the standard form.

R=addr—RX-type address, (2-12), (0), (14) or n
it is coded as shown in the standard form.

K=addr—RX-type address, (2-12), (0), (14), or n
it is coded as shown in the standard form.

DD=addr—RX-type address, (2-12), (0), (14), or n
it is coded as shown in the standard form.

REGSAVE={YES|NO}
it is coded as shown in the standard form.

MF=(E,addr)
this operand specifies that the execute form of the TRKCALC macro instruction is used, and an existing data management parameter list is used.

E
Coded as shown

addr—RX-type address, (0), (1), (2-12), or (14)
specifies an in-storage address of the parameter list.

TRKCALC—DSECT ONLY

This call gives a symbolic expansion of the parameter list for the TRKCALC macro. No DSECT statement is generated. If a name is specified on the macro call, it applies to the beginning of the list, after any necessary boundary alignment. The macro generated symbols all begin with "STAR".

[symbol]	TRKCALC	MF=D

TRKCALC MACRO EXAMPLES

In this example, TRKCALC is coded to determine how many records of a given size with 10-byte keys fit on a 3330 track. After issuing the macro, the number of records is saved in NUMREC:

```
TRKCALC   FUNCTN=TRKCAP,TYPE=UTYPE,R=1,K=10,DD=DL,MF=(E(1))
          .
          .
          ST    0,NUMREC    SAVE NUMBER OF RECORDS
          .
          .
DL        DC    H'xxxx'     DATA LENGTH
UTYPE     DC    X'09'
NUMREC    DS    F           MAX # OF RECORDS
```

In this example, TRKCALC is coded to determine if another record can fit on a track of a 3350, given a track balance.

```
TRKCALC   FUNCTN=TRKBAL,TYPE=UTYPE,R=REC,K=KL,DD=DD,BALANCE=BAL,
          MAXSIZE=YES,MF=(E(1))
          .
          .
UTYPE     DC    X'0B'
REC       DC    X'xx'
KL        DC    X'xx'
DD        DC    H'xxxx'
BAL       DC    H'xxxx'
```

After issuing the macro, you would receive either:

Register 15=0. Register 0 contains the new balance.

Register 15=4. Register 0=0 (record did not fit).

Register 15=8. Register 0 contains the maximum data length that does fit (record did not fit).

ALLOCATING A DATA SET

The REALLOC macro builds a parameter list and issues SVC 32 to allocate a new data set based on a partial DSCB that describes the attributes of that data set. You can use the OBTAIN macro to get the format-1 DSCB of the other data set and use it as a model for the new data set's DSCB which REALLOC will construct and write in the VTOC.

The maximum number of extents that may be allocated are determined by the type of data set requested as defined by the data set organization (DS1DSORG) bytes and the data set indicator (DS1DSIND) byte in the partial DSCB. If the DS1DSORG indicates a VSAM data set organization and DS1DSIND indicates the data set is cataloged in an integrated catalog facility (ICF) catalog, the maximum number of extents will be 123. Otherwise, the maximum number of extents will be 16.

Note: User label data sets, ISAM data sets, and absolute track allocated data sets are not supported by the REALLOC macro. If a VSAM data set or data space is requested, REALLOC does not interface with VSAM or ICF catalog management.

The DS1SCALO field of the partial format-1 DSCB has a high order flag byte that describes the type of request and a 3-byte field containing the secondary allocation quantity. The following describes the flag byte:

Contents	**Meaning**
X'C0'	Cylinder request
X'C8'	Cylinder with CONTIG request
X'80'	Track request
X'88'	Track with CONTIG request
X'40'	Average block length request
X'41'	Average block length with ROUND request
X'48'	Average block length with CONTIG request
X'49'	Average block length with CONTIG and ROUND request

Any settings other than the above will be ignored.

The REALLOC macro may be coded in the execute, dsect, and list forms, but not the standard form. The calling program may be in supervisor or problem program state, and may be running in any key. The calling program must be APF authorized.

REALLOC—EXECUTE FORM

The format of the REALLOC macro in execute form is:

[symbol	**REALLOC**	**MF=(E,addr)** **,DSSIZE=**addr|(reg) **,PDSCB=**addr **,UCB=**addr **[,MINAU=**addr|(reg)**]** **[,PDSDIR=**addr|(reg)**]**

MF=(E,addr**)**
specifies that the execute form of the macro and an existing REALLOC parameter list will be used.

addr—RX-type address, (0-12)
specifies an in-storage address of the REALLOC parameter list.

E
Code as shown.

DSSIZE=addr|(reg)

specifies the size of the data set to be allocated in tracks. If a cylinder request (X'C0' in the flag byte of DS1SCALO) or average block with round request (X'41') is made, the number of tracks specified will be rounded up to the next full cylinder, if necessary.

You may not specify the DSSIZE in terms of average block size, even though the original data set may have been allocated with the number of average blocks (X'40').

addr—RX-type address

specifies an in-storage address of a full word containing the data set size.

(reg)

specifies a register containing the size of the data set. Valid registers are 0 and 2-12.

PDSCB=addr—RX-type address, (0), (2-12)

specifies the address of the partial DSCB. The partial DSCB is comprised of the first 98 bytes of a format-1 DSCB. The first 44 bytes contains the data set name to be allocated. The contents of the partial DSCB will be used, unchanged, in constructing the format-1 DSCB. Only the field DS1NOEPV (number of extents on the volume) of the partial format-1 DSCB will be modified by allocation to reflect the actual number of extents allocated.

UCB=addr—RX-type address, (0), (2-12)

specifies the address of the UCB of the volume where the data set is to be allocated. The volume must be mounted and the caller is responsible for ensuring that the volume remains mounted on the unit.

MINAU=addr|(reg)

specifies the size of the minimum allocation unit in tracks. All primary extents for this data set will be in multiples of this minimum allocation unit. This value will not apply to subsequent extensions of the data set. If the partial DSCB indicates the data set is to be allocated in cylinders (X'C0' or average block with round request (X'41'), this parameter will be ignored.

addr—RX-type address

specifies an in-storage address of a full word containing the minimum allocation unit.

(reg)

specifies a register containing the minimum allocation unit. Valid registers are 0 and 2-12.

PDSDIR=addr|(reg)

specifies the number of 256 byte directory blocks for a partitioned data set (PDS). This is a required keyword if the DS1DSORG indicates a partitioned data set. Otherwise, it is ignored.

addr—RX-type address

specifies an in-storage address of a full word containing the number of 256 byte PDS directory blocks.

(reg)

specifies a register containing the number of 256 byte PDS directory blocks. Valid registers are 0 and 2-12.

RETURN CODES FROM REALLOC

Control will be returned at the instruction following the SVC 32 generated by the REALLOC macro. If the data set was successfully allocated, register 15 will contain zeros. Otherwise, register 15 will contain one of the following return codes:

Code	Meaning
004(04)	Data set name of request already exists on this volume. Initial allocation not possible under the name given.
008(08)	No room available in the VTOC or VTOC index.
012(0C)	One of the following errors was encountered: • Permanent I/O error • Error returned by CVAF
020(14)	Requested quantity not available.
028(1C)	ISAM DSORG is not supported.
048(30)	Invalid REALLOC parameter list.
052(34)	Invalid partial DSCB pointer.
056(38)	Not enough space on volume for directory.
072(48)	DOS VTOC cannot be converted to an OS VTOC.
116(74)	User labels not supported.
120(78)	DSSIZE=0 and MINAU is greater than 0.
124(7C)	DSSIZE is not a multiple of MINAU.
128(80)	Directory space requested is larger than primary space.
148(94)	Overlapping extents in the VTOC.
152(98)	Overlapping DOS split cylinder extents in the VTOC.
156(9C)	DADSM allocation terminated due to possible VTOC errors.
164(A4)	Allocation terminated due to DOS stacked pack format.
168(A8)	RACF DEFINE failed, data set already defined.
172(AC)	User not authorized to RACF define data set.
176(B0)	Installation exit rejected this request with a return code of 8.
180(B4)	Installation exit rejected this request with a return code of 4.

REALLOC—DSECT ONLY

The dsect form of REALLOC is specified as follows:

[symbol	REALLOC	MF=D

An example of the dsect form expansion is:

```
REALPL   REALLOC MF=D
REALPL   DSECT                           DSECT FOR PARAMETER LIST
RALPLID  DS    CL4                       EBCDIC 'REAL' FOR REALLOC
RALNGTH  DS    AL2                       LENGTH OF PARAMETER LIST
RAERRCDE DS    H                         ERROR CODE RETURNED FROM
*                                        ALLOCATE (SVC 32)
RALRSVD  DS    F                         RESERVED
RALDSSZ  DS    F                         DATA SET SIZE
RALMAU   DS    F                         MINIMUM ALLOCATION UNIT
RALPDSCB DS    A                         PARTIAL DSCB POINTER
RALUCB   DS    A                         UCB POINTER
RALDQTY  DS    F                         PDS DIRECTORY QUANTITY
RALEND   EQU   *                         END OF PARAMETER LIST
RALENGTH EQU   RALEND-REALPL             LENGTH OF PARAMETER LIST
```

REALLOC—LIST FORM

The list form of the REALLOC macro is specified as follows:

[symbol	REALLOC	MF=L ,DSSIZE=addr|(reg) ,PDSCB=addr ,UCB=addr [,MINAU=addr|(reg) [,PDSDIR=addr|(reg)

Refer to the execute form for an explanation of the parameters.

An example of the list form expansion is:

```
REALPL   REALLOC MF=L
         CNOP  0,4
REALPL   EQU   *
         DC    CL4'REAL'                 EBCDIC 'REAL' FOR REALLOC
         DC    AL2(32)                   LENGTH OF PARAMETER LIST
         DC    H'0'                      ERROR CODE RETURNED FROM
*                                        ALLOCATE (SVC 32)
         DC    F'0'                      RESERVED
         DC    F'0'                      DATA SET SIZE
         DC    F'0'                      MINIMUM ALLOCATION UNIT
         DC    A(0)                      PARTIAL DSCB POINTER
         DC    A(0)                      UCB POINTER
         DC    F'0'                      PDS DIRECTORY QUANTITY
RAL01E   EQU   *                         END OF PARAMETER LIST
```

CHAPTER 8. MAINTAINING SYS1.IMAGELIB

This chapter describes how to maintain the system image library (SYS1.IMAGELIB) UCS and FCB images for the 1403, 3203, and 3211 printers. It also describes how to maintain FCB images for the 4245 printer, the UCS image table in SYS1.IMAGELIB for the 4245, and how to retrieve an FCB image from SYS1.IMAGELIB in order to modify it.

The IEBIMAGE utility program is used to create and maintain control modules for the 3800 printing subsystem: character arrangement table modules, graphic character modification modules, copy modification modules, library character set modules, and FCB modules. For further information on IEBIMAGE, see Utilities.

To use the information presented in this chapter, you should be familiar with the subjects of the following publications:

- Data Management Macro Instructions describes the SETPRT macro, which can specify the UCS and/or FCB images to be used.

- JCL describes the UCB and FCB parameters of the DD statement, which are processed at OPEN time.

- IBM 2821 Control Unit Component Description contains information on creating a user-designed chain/train for the 1403 printer.

- IBM 3203 Printer Component Description and Operator's Guide contains information on creating a user-designed train for the 3203 printer.

- IBM 3211 Printer, 3216 Interchangeable Train Cartridge, and 3811 Printer Control Unit Component Description and Operator's Guide contains information on creating a user-designed train for the 3211 printer.

- System Programming Library: JES2 or System Programming Library: Network Job Entry Facility for JES2 contains reference information for JES2.

- System Programming Library: JES3 contains reference information for JES3.

The 4245 printer has no UCS images supplied in SYS1.IMAGELIB. To determine which UCS images are available, see:

- IBM 4245 Printer Model 1 Component Description and Operator's Guide contains information on band IDs for the 4245 printer.

The SPZAP service aid can be used to display and modify an existing member of SYS1.IMAGELIB. Use of SPZAP on load modules is described in System Programming Library: Service Aids.

UCS IMAGES IN SYS1.IMAGELIB

Most IBM standard character set images are included in SYS1.IMAGELIB at system generation time, through the DATAMGT macro and an IODEVICE macro for the specified printer. (See System Generation Reference for details on the DATAMGT and IODEVICE macros.) The standard character set images for the 1403, 3203, and 3211 printers are shown in the table below.

Printer	Images
1403 or 3203	AN, HN, PCAN, PCHN, PN, QNC, QN, RN, SN, TN, XN, YN
3211	A11, G11, H11, P11, T11

For the 4245, no UCS images are supplied in SYS1.IMAGELIB at system generation. Instead, a new UCS image is loaded into the buffer at power-on time or whenever the operator mounts a new band. See "Adding a UCS Image Name/Alias to a UCS Image Table" on page 175 for information on how to access UCS images that are not supplied in SYS1.IMAGELIB.

The 4245 printers also load a default FCB image at power-on time. For the 4245, the default FCB is an 11-inch form with 6 LPI and a Channel 1 on the first print line.

The alias names are defined for most installation-standard print chains/trains/bands installable on a given printer. Alias names are included in SYS1.IMAGELIB (in the UCS image table) at system generation time, with the real name of each image.

Some print chains/trains/bands, such as SN and G11, do not have alias names because there is no equivalent chain/train/band on other printers. You can assign an alias for these chains/trains/bands with the ALIAS statement of the linkage editor. (See Linkage Editor and Loader for more information on the ALIAS statement.) For the 4245 printer, you can also add an alias name by modifying an entry in the UCS image table. See "Adding a UCS Image Name/Alias to a UCS Image Table" on page 175.

If an alias name is supplied, it is used to schedule a printer for SYSOUT data sets. If no alias is supplied, an installation-defined SYSOUT class or a printer routing code (specified with the DEST parameter of JCL) should be used to assign the data set to the correct printer.

ADDING A UCS IMAGE TO THE IMAGE LIBRARY

Using the assembler and linkage editor, you may add a UCS image to those that reside in SYS1.IMAGELIB. No executable code is generated; the assembler prepares DCs and the linkage editor puts them into SYS1.IMAGELIB. The new UCS image must be structured according to the following rules:

1. The member name must be 5 to 8 characters long; the first 4 characters must be the appropriate UCS prefix, as shown below.

   UCS1 - 1403 printer

   UCS2 - 3211 printer

   UCS3 - 3203 printer

   These first 4 characters must be followed by a character set code, 1 to 4 characters long. Any valid combination of letters and numbers under assembler language rules is acceptable. However, the single letters U or C must not be used, because they are symbols for special conditions recognized by the system. The assigned character set code must be specified on the DD statement or SETPRT macro to load the image into the UCS buffer.

   You can supply an alias name for a new image with the ALIAS statement of the linkage editor. (See Linkage Editor and Loader for more information on the ALIAS statement.)

2. The first byte of the character set image load module specifies whether the image is a default. (Default images may be used by the system for jobs that do not request a specific image.) Specify the following in the first byte:

   For JES2:

   X'80' indicates a default image

   X'40' indicates that the output is to be folded

   X'C0' indicates default image and folding

   X'00' indicates that the image is not to be used as a default

   For non-JES2:

   X'80' indicates a default image

   X'00' indicates that the image is not to be used as a default

3. The second byte of the load module indicates the number of lines (n) to be printed for image verification. See "Verifying the UCS Image" on page 178 for more information on image verification.

4. Each byte of the next n bytes indicates the number of characters to be printed on each verification line. For the 3211 printer, the maximum number of characters printed per line is 48; the bytes of associative bits (see note 5) are not printed during verification.

5. The UCS image itself must follow the previously described fields. The image must fill the number of bytes required by the printer; see the table below for image lengths. Note that, because of assembler language syntax, two apostrophes or two ampersands must be coded to represent a single apostrophe or a single ampersand, respectively, within a character set image.

Printer	**Image Length**
1403	240 bytes
3203	304 bytes (240 characters followed by 64 bytes of associative bits)
3211	512 bytes (432 characters followed by 15 bytes of X'00', 64 bytes of associative bits, and one reserved byte of X'00')

Associative bits must be coded to prevent data checks when adding a UCS image to SYS1.IMAGELIB. See the appropriate printer manual for more information on coding associative bits.

Figure 39 on page 173 contains an example of adding a 1403 UCS image, YN, to SYS1.IMAGELIB. Notes follow Figure 41 on page 174.

```
//ADDYN          JOB  MSGLEVEL=1
//STEP           EXEC PROC=ASMFCL,PARM.ASM='NODECK,LOAD',
//                    PARM.LKED='LIST,OL,REFR,RENT,XREF'
//ASM.SYSIN      DD   *
UCS1YN           CSECT
                 DC   X'80'         (THIS IS A DEFAULT IMAGE)
                 DC   AL1(6)        (NUMBER OF LINES TO BE PRINTED)
                 DC   AL1(39)       (39 CHARACTERS TO BE PRINTED ON LINE 1)
                 DC   AL1(42)       (42 CHARACTERS TO BE PRINTED ON LINE 2)
                 DC   AL1(39)       (39 CHARACTERS TO BE PRINTED ON LINE 3)
                 DC   AL1(39)       (39 CHARACTERS TO BE PRINTED ON LINE 4)
                 DC   AL1(42)       (42 CHARACTERS TO BE PRINTED ON LINE 5)
                 DC   AL1(39)       (39 CHARACTERS TO BE PRINTED ON LINE 6)
*       THE FOLLOWING SIX LINES REPRESENT THE TRAIN IMAGE
                 DC   C'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*,.'
                 DC   C'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*,.#-$'
                 DC   C'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*,.'
                 DC   C'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*,.'
                 DC   C'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*,.#-$'
                 DC   C'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*,.'
                 END
/*
//LKED.SYSLMOD DD     DSNAME=SYS1.IMAGELIB(UCS1YN),DISP=OLD,
//                    SPACE=         (OVERRIDE SECONDARY ALLOCATION)
```

Figure 39. Sample Code to Add a 1403 UCS Image to SYS1.IMAGELIB

Figure 40 shows the code used to add a 3203 UCS image, YN, to the image library.

```
//ADYN3203       JOB  MSGLEVEL=1
//STEP           EXEC PROC=ASMFCL,PARM.ASM='NODECK,LOAD',
//                    PARM.LKED='LIST,OL,REFR,RENT,XREF'
//ASM.SYSIN      DD   *
UCS3YN           CSECT
                 DC   X'80'         (THIS IS A DEFAULT IMAGE)
                 DC   AL1(6)        (NUMBER OF LINES TO BE PRINTED)
                 DC   AL1(39)       (39 CHARACTERS TO BE PRINTED ON LINE 1)
                 DC   AL1(42)       (42 CHARACTERS TO BE PRINTED ON LINE 2)
                 DC   AL1(39)       (39 CHARACTERS TO BE PRINTED ON LINE 3)
                 DC   AL1(39)       (39 CHARACTERS TO BE PRINTED ON LINE 4)
                 DC   AL1(42)       (42 CHARACTERS TO BE PRINTED ON LINE 5)
                 DC   AL1(39)       (39 CHARACTERS TO BE PRINTED ON LINE 6)
*       THE FOLLOWING SIX LINES REPRESENT THE TRAIN IMAGE
                 DC   C'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*,.'
                 DC   C'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*,.#-$'
                 DC   C'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*,.'
                 DC   C'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*,.'
                 DC   C'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*,.#-$'
                 DC   C'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*,.'
*       THE FOLLOWING FOUR DC INSTRUCTIONS DEFINE THE ASSOCIATIVE BITS,
*       UCSB BYTE POSITIONS 241-304
                 DC   X'C010101010101010101000400000000000010'
                 DC   X'101010101010101000404000000040001010'
                 DC   X'101010101010004000000000101010101010'
                 DC   X'10101010004000000000'
                 END
/*
//LKED.SYSLMOD DD     DSNAME=SYS1.IMAGELIB(UCS3YN),DISP=OLD,
//                    SPACE=           (OVERRIDE SECONDARY ALLOCATION)
```

Figure 40. Sample Code to Add a 3203 UCS Image to SYS1.IMAGELIB

Figure 41 shows the code used to add a 3211 UCS image, A11, to SYS1.IMAGELIB.

```
//ADDA11        JOB  MSGLEVEL=1
//STEP          EXEC PROC=ASMFCL,PARM.ASM='NODECK,LOAD',
//                   PARM.LKED='LIST,OL,REFR,RENT,XREF'
//ASM.SYSIN     DD   *
UCS2A11         CSECT
                DC   X'80'          (THIS IS A DEFAULT IMAGE)
                DC   AL1(9)         (NUMBER OF LINES TO BE PRINTED)
                DC   AL1(48)        (48 CHARACTERS TO BE PRINTED ON LINE 1)
                DC   AL1(48)        (48 CHARACTERS TO BE PRINTED ON LINE 2)
                DC   AL1(48)        (48 CHARACTERS TO BE PRINTED ON LINE 3)
                DC   AL1(48)        (48 CHARACTERS TO BE PRINTED ON LINE 4)
                DC   AL1(48)        (48 CHARACTERS TO BE PRINTED ON LINE 5)
                DC   AL1(48)        (48 CHARACTERS TO BE PRINTED ON LINE 6)
                DC   AL1(48)        (48 CHARACTERS TO BE PRINTED ON LINE 7)
                DC   AL1(48)        (48 CHARACTERS TO BE PRINTED ON LINE 8)
                DC   AL1(48)        (48 CHARACTERS TO BE PRINTED ON LINE 9)
*       THE FOLLOWING NINE LINES REPRESENT THE TRAIN IMAGE
*       NOTE 2 AMPERSANDS MUST BE CODED TO GET 1 IN ASSEMBLER SYNTAX
                DC   C'1<.+IHGFEDCBA*$-RQPONMLKJ%,&&ZYXWVUTS/@#098765432'
                DC   C'1<.+IHGFEDCBA*$-RQPONMLKJ%,&&ZYXWVUTS/@#098765432'
                DC   C'1<.+IHGFEDCBA*$-RQPONMLKJ%,&&ZYXWVUTS/@#098765432'
                DC   C'1<.+IHGFEDCBA*$-RQPONMLKJ%,&&ZYXWVUTS/@#098765432'
                DC   C'1<.+IHGFEDCBA*$-RQPONMLKJ%,&&ZYXWVUTS/@#098765432'
                DC   C'1<.+IHGFEDCBA*$-RQPONMLKJ%,&&ZYXWVUTS/@#098765432'
                DC   C'1<.+IHGFEDCBA*$-RQPONMLKJ%,&&ZYXWVUTS/@#098765432'
                DC   C'1<.+IHGFEDCBA*$-RQPONMLKJ%,&&ZYXWVUTS/@#098765432'
                DC   C'1<.+IHGFEDCBA*$-RQPONMLKJ%,&&ZYXWVUTS/@#098765432'
                DC   15X'00'        (RESERVED FIELD, BYTES 433-447)
*       THE FOLLOWING FOUR DC INSTRUCTIONS DEFINE THE ASSOCIATIVE BITS,
*       UCSB BYTE POSITIONS 448-511
                DC   X'C010101010101010101000040404240004010'
                DC   X'1010101010101010000404041000040401010'
                DC   X'1010101010100040400000000101010101010'
                DC   X'101010100004040444800'
                DC   X'00'          (RESERVED FIELD, BYTE 512)
                END
/*
//LKED.SYSLMOD DD    DSNAME=SYS1.IMAGELIB(UCS2A11),DISP=OLD,
//                   SPACE=         (OVERRIDE SECONDARY ALLOCATION)
```

Figure 41. Sample Code to Add a 3211 UCS Image to SYS1.IMAGELIB

Notes to Figure 39 on page 173, Figure 40 on page 173, and Figure 41:

1. The RENT and REFR linkage editor attributes are used for performance considerations in a paging environment. They may be omitted.

2. For the 3203 and 3211 printers, the 64 bytes of associative bits must be coded to avoid data checks. To determine how to code these bits for a particular image, see IBM 3203 Printer Component Description and Operator's Guide or IBM 3211 Printer, 3216 Interchangeable Train Cartridge, and 3811 Printer Control Unit Component Description and Operator's Guide.

3. Executing the ASMFCL procedure does not actually generate executable code. The assembler/linkage editor is used to place the UCS image into SYS1.IMAGELIB.

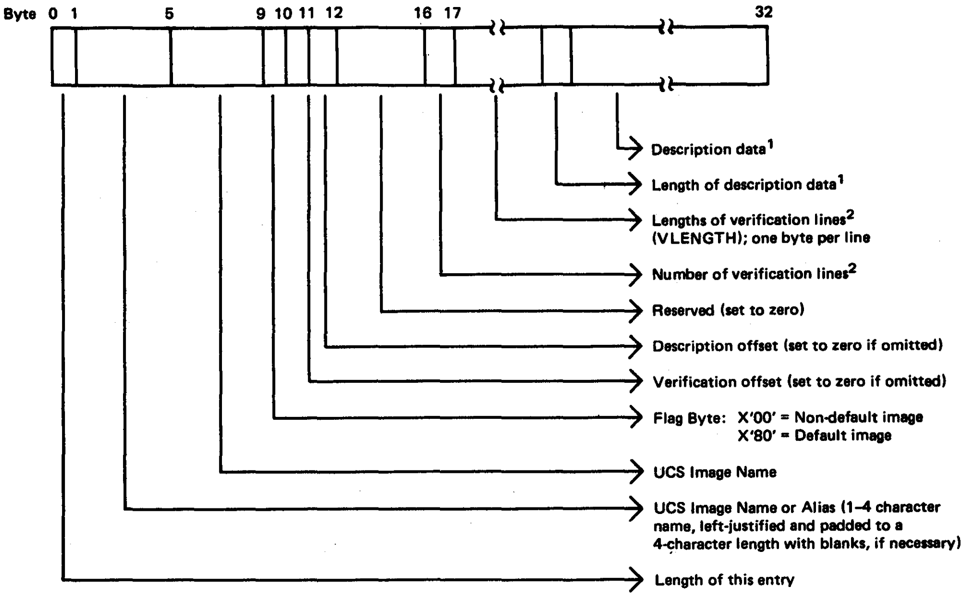
4. The SPACE parameter is overridden here because the ASMFCL cataloged procedure has secondary allocation specified. Elimination of the override causes the original secondary allocation amount to be used.

ADDING A UCS IMAGE NAME/ALIAS TO A UCS IMAGE TABLE

For the 4245 printer, no UCS images are stored in SYS1.IMAGELIB. Instead, the image for each band is stored in the printer, and automatically loaded into the UCS buffer at power-on time or when a new band is installed. Information about these images is recorded in the IBM-supplied **UCS image table**, which resides in SYS1.IMAGELIB.

UCS Image Table Structure

SYS1.IMAGELIB contains one UCS image table for each type of printer that supports image tables. For the 4245, the table is called UCS5. The image table contains an entry for most installation-standard IBM-supplied bands. A typical UCS image table entry takes the form shown in Figure 42.



Byte 0 1 5 9 10 11 12 16 17 32
Description data[1]
Length of description data[1]
Lengths of verification lines[2] (VLENGTH); one byte per line
Number of verification lines[2]
Reserved (set to zero)
Description offset (set to zero if omitted)
Verification offset (set to zero if omitted)
Flag Byte: X'00' = Non-default image X'80' = Default image
UCS Image Name
UCS Image Name or Alias (1–4 character name, left-justified and padded to a 4-character length with blanks, if necessary)
Length of this entry

Figure 42. UCS Image Table Entry Format

Notes to Figure 42:

1. This field is optional.
2. This field is optional for the 4245 printer.

The contents of the UCS image table UCS5 (IGGUCS5 macro), for the 4245 printer, are shown in Figure 43.

Name	Alias	Default	Description
AN21	AN21	YES	Default UCS image
AN21	AN	NO	1403/3203 AN image
AN21	A11	NO	3211 A11 image
AN21	40E1	NO	4248 40E1 image
HN21	HN21	NO	Nondefault UCS image
HN21	HN	NO	1403/3203 HN image
HN21	H11	NO	3211 H11 image
HN21	4101	NO	4248 4101 image
PL21	PL21	NO	Nondefault UCS image
PL21	PN	NO	1403/3203 PN image
PL21	P11	NO	3211 P11 image
PL21	4121	NO	4248 4121 image
SN21	SN21	NO	Nondefault UCS image
SN21	4201	NO	4248 4201 image
TN21	TN21	NO	Nondefault UCS image
TN21	TN	NO	1403/3203 TN image
TN21	T11	NO	3211 T11 image
TN21	4181	NO	4248 4181 image
GN21	GN21	NO	Nondefault UCS image
GN21	G11	NO	3211 G11 image
GN21	41C1	NO	4248 41C1 image
RN21	RN21	NO	Nondefault UCS image
RN21	RN	NO	1403/3203 RN image
KA21	KA21	NO	Nondefault UCS image
KA21	4041	NO	4248 4041 image
KA22	KA22	NO	Nondefault UCS image
FC21	FC21	NO	Nondefault UCS image
FC21	4161	NO	4248 4161 image

Figure 43. UCS5 Image Table Contents

Note: The image table for the 4245 printer includes USA and Canada band IDs only. To support other national band IDs, you will have to modify the UCS image table. See "Adding/Modifying a UCS Image Table Entry."

Adding/Modifying a UCS Image Table Entry

If you plan to use a new UCS image name/alias with the 4245 printer, you must add an entry for that image name/alias to the appropriate UCS image table. Similarly, if you want to select a new default image or change the description on an old image, you must make the change in the image table.

To build new UCS table entries, or to change the format of old entries, use the following procedure:

1. Issue the IGGUCSIT macro, as described below, to build a new UCS image table entry. A new entry is built even if it is intended to replace an existing entry supplied by IBM. Because the new entry is found first, the previous entry is never found and thus is effectively replaced.

2. Include the UCS image table, source using the macro IGGUCS5, which is found in SYS1.MACLIB.

3. Reassemble the image table module (UCS5).

4. Link-edit the reassembled module into SYS1.IMAGELIB.

The IGGUCSIT macro instruction has the following format:

IGGUCSIT	**MF={LIST|DSECT}** **,NAME=**image name **[,ALIAS=**image alias**]** **[,DEFAULT={YES|NO}]** **[,DESCR=**description**]** **[,VLENGTH=(**n1,n2,. . .n**)]**

MF={LIST|DSECT}
specifies the form of the macro instruction.

LIST
produces a UCS image table entry based on the information supplied in other IGGUCSIT parameters. If LIST is selected or allowed to default, the NAME parameter must also be coded.

DSECT
produces a DSECT for a single UCS image table entry, similar to the sample entry shown in Figure 42 on page 175. If DSECT is coded, all other parameters of IGGUCSIT are ignored.

LIST is the default.

NAME=image name
specifies the 1- to 4-character UCS image name.

ALIAS=image alias
specifies a 1- to 4-character alias name for the UCS image. If ALIAS is not specified, the image name coded in the NAME parameter will be entered in the UCS image table.

DEFAULT={YES|NO}
indicates whether the new UCS image is to be used as a default value.

YES
indicates that this UCS image is a default. Default images are used by the system for jobs that do not request a specific image.

NO
indicates that this UCS image should not be used as a default.

If the DEFAULT parameter is not specified, the new UCS image is not used as a default.

DESCR=description

specifies descriptive information about the new UCS image. description can be up to 32 EBCDIC or hexadecimal characters in length, although EBCDIC and hexadecimal characters cannot be used in combination.

The descriptive information is placed in the header line of the verification display, following the real UCS image name. If the DESCR parameter is omitted, no description will appear in the display. For more information on the verification display, see "Verifying the UCS Image."

If VLENGTH is not specified for the 4245 printer, the DESCR parameter is ignored.

VLENGTH=(n1,n2,. . . n)

specifies the length(s) of each line in the UCS verification display. The length of each line must be specified separately, even if all lines are the same length.

n1 is the length of print line 1; n2 is the length of print line 2; n is the length of the last print line. The sum of the verification line lengths should be equal to 350 in order to display the complete image.

See "Verifying the UCS Image" for details on the verification report.

Verifying the UCS Image

For the 1403 (with the UCS feature), 3203, 3211, 3262, and 4245 printers, the UCS image can be displayed on the printer for visual verification using either of the following parameters:

- In JCL: UCS=(character set code,,VERIFY)
- In the SETPRT macro: UCS=(character set code,,V)

The verification display header appears in the format shown below.

```
UCS IMAGE VERIFICATION image id [,FOLD] [description]
```

image id

The 1- to 4-character name of the UCS image.

description

The descriptive information supplied for this UCS image in the UCS image table.

The 4245 also, optionally, prints the image.

See JCL and Data Management Macro Instructions for more information on the UCS VERIFY parameters.

EXAMPLES OF ADDING TO THE UCS IMAGE TABLE

Example 1: Adding a New Band ID to the 4245 Image Table (UCS5)

In this example, the band name RPQ1 with description "RPQ BAND" is added to UCS5. In the UCS verification display, 7 lines of 50 characters each are printed. Macro IGGUCS5 causes the UCS image table source (as distributed by IBM) to be included.

```
                                                            72
//UCS5     JOB  . . .
//         EXEC ASMFCL,
//              PARM.ASM='NODECK,LOAD',
//              PARM.LKED='OL,RENT,REUS'
//SYSPRINT DD   SYSOUT=A
//ASM.SYSIN DD  *
       TITLE 'UPDATED UCS5 IMAGE TABLE'
 UCS5  CSECT
       IGGUCSIT NAME=RPQ1,                                       X
                VLENGTH=(50,50,50,50,50,50,50),                  X
                DESCR='RPQ BAND'
       IGGUCS5
       END
/*
//LKED.SYSLMOD DD DSN=SYS1.IMAGELIB(UCS5),DISP=OLD,
//                SPACE=    (OVERRIDE SECONDARY ALLOCATION)
```

Notes to Example 1:

1. The RENT and REUS linkage editor attributes are used for performance considerations in a paging environment. They may be omitted.

2. Executing the ASMFCL procedure does not actually generate executable code. The assembler/linkage editor is used to place the UCS image table entry into SYS1.IMAGELIB.

3. The SPACE parameter is overridden here because the ASMFCL cataloged procedure has secondary allocation specified. Elimination of the override causes the original secondary allocation amount to be used.

ADDING AN FCB IMAGE TO THE IMAGE LIBRARY

Two standard FCB images, STD1 and STD2, are included in SYS1.IMAGELIB during system generation for the following printers:

3203

3211

4245

STD1 sets line spacing at 6 lines per inch for an 8-1/2 inch form; STD2 is a default FCB image that sets line spacing at 6 lines per inch for an 11-inch form. Channels for both images are evenly spaced, with Channel 1 on the fourth line and Channel 9 on the last line. See Figure 44 and Figure 45 on page 178.2 for sample STD1 and STD2 images.

The 4245 printer loads a default FCB image into the buffer at power-on time. The 4245 default FCB image is an 11-inch form with 6 lines per inch and a Channel 1 on the first print line.

The IEBIMAGE utility should be used to create and modify FCB modules for the 3800 printing subsystem.

```
FCB2STD1  CSECT
          DC    X'80'                DEFAULT
          DC    AL1(51)              FCB IMAGE LENGTH = 51
          DC    X'000000'            LINE 1, 2, 3
          DC    X'01'                LINE 4, CHANNEL 1
          DC    X'000000'            LINE 5, 6, 7
          DC    X'02'                LINE 8, CHANNEL 2
          DC    X'000000'            LINE 9, 10, 11
          DC    X'03'                LINE 12, CHANNEL 3
          DC    X'000000'            LINE 13, 14, 15
          DC    X'04'                LINE 16, CHANNEL 4
          DC    X'000000'            LINE 17, 18, 19
          DC    X'05'                LINE 20, CHANNEL 5
          DC    X'000000'            LINE 21, 22, 23
          DC    X'06'                LINE 24, CHANNEL 6
          DC    X'000000'            LINE 25, 26, 27
          DC    X'07'                LINE 28, CHANNEL 7
          DC    X'000000'            LINE 29, 30, 31
          DC    X'08'                LINE 32, CHANNEL 8
          DC    X'000000'            LINE 33, 34, 35
          DC    X'0A'                LINE 36, CHANNEL 10
          DC    X'000000'            LINE 37, 38, 39
          DC    X'0B'                LINE 40, CHANNEL 11
          DC    X'0000000000000000'  LINE 41, 42, 43, 44, 45, 46, 47, 48
          DC    X'0C'                LINE 49, CHANNEL 12
          DC    X'00'                LINE 50
          DC    X'19'                LINE 51, CHANNEL 9-END OF FCB IMAGE
          END
```

Figure 44. Sample of the Standard FCB Image STD1

```
FCB2STD2  CSECT
          DC    X'80'                DEFAULT
          DC    AL1(66)              FCB IMAGE LENGTH = 66
          DC    X'000000'            LINE 1, 2, 3
          DC    X'01'                LINE 4, CHANNEL 1
          DC    X'0000000000'        LINE 5, 6, 7, 8, 9
          DC    X'02'                LINE 10, CHANNEL 2
          DC    X'0000000000'        LINE 11, 12, 13, 14, 15
          DC    X'03'                LINE 16, CHANNEL 3
          DC    X'0000000000'        LINE 17, 18, 19, 20, 21
          DC    X'04'                LINE 22, CHANNEL 4
          DC    X'0000000000'        LINE 23, 24, 25, 26, 27
          DC    X'05'                LINE 28, CHANNEL 5
          DC    X'0000000000'        LINE 29, 30, 31, 32, 33
          DC    X'06'                LINE 34, CHANNEL 6
          DC    X'0000000000'        LINE 35, 36, 37, 38, 39
          DC    X'07'                LINE 40, CHANNEL 7
          DC    X'0000000000'        LINE 41, 42, 43, 44, 45
          DC    X'08'                LINE 46, CHANNEL 8
          DC    X'0000000000'        LINE 47, 48, 49, 50, 51
          DC    X'0A'                LINE 52, CHANNEL 10
          DC    X'0000000000'        LINE 53, 54, 55, 56, 57
          DC    X'0B'                LINE 58, CHANNEL 11
          DC    X'0000000000'        LINE 59, 60, 61, 62, 63
          DC    X'0C'                LINE 64, CHANNEL 12
          DC    X'00'                LINE 65
          DC    X'19'                LINE 66, CHANNEL 9-END OF FCB IMAGE
          END
```

Figure 45. Sample of the Standard FCB Image STD2

You may add a 3211 format FCB image to those that reside in SYS1.IMAGELIB, using the assembler and linkage editor. No executable code is generated; the assembler prepares DCs, and the linkage editor puts them into SYS1.IMAGELIB. The new FCB image must be structured according to the following rules:

1. The member name cannot exceed 8 bytes. The first 4 characters of the name must be FCB2. The characters that follow identify the FCB image and are referred to as the "image identifier" (ID). Any combination of valid assembler language characters can be used, with the exception of a single "C" or "U", because these are used by the system to recognize special conditions. The image identifier must be specified in the FCB keyword of a DD statement or in the SETPRT macro to load the image into the FCB buffer.

2. The first byte of the FCB load module specifies whether the image is a default. (Default images may be used by the system for jobs that do not request a specific image.) Specify the following in the first byte:

   X'80' indicates a default image
   X'00' indicates a nondefault image

3. The second byte of the load module indicates the number of bytes to be transferred to the control unit to load the FCB image. This count includes the byte, if used, for the print position indexing feature.

4. The third byte of the load module (the first byte of the FCB image) is either the print position indexing byte, or the lines-per-inch byte. The print position indexing byte is optional and, when used, precedes the lines-per-inch byte. The 4245 printer accepts and discards the index byte if it is present, because the printer does not support the indexing feature. A description of the print position indexing feature and its use will be found in <u>IBM 3211 Printer, 3216 Interchangeable Train Cartridge, and 3811 Printer Control Unit Component Description and Operator's Guide</u>.

   The special index flag in the third byte contains X'80' plus a binary index value, from 1 to 32 (the default is 1). This index value sets the left margin: 1 indicates flush-left; any other value indicates a line indented that many spaces.

   The form image begins with the lines-per-inch (LPI) byte. The LPI byte defines the number of lines per inch (6 or 8) and also represents the first line of the page. It may or may not also contain a channel identifier.

   Typically, the length of an FCB image is consistent with the length of the form it represents. For example, an 8-1/2 inch form to be printed at 6 LPI has an FCB image that is 51 bytes long (8-1/2 inches times 6 LPI).

   The LPI byte appears as follows:

   X'1n' sets 8 LPI

   X'0n' sets 6 LPI

5. All remaining bytes (lines) must contain X'0n', except the last byte, which must be X'1n'. The letter n can be a hexadecimal value from 1 to C, representing a channel (one to 12), or it can be 0, which means no channel is indicated.

In Figure 46 on page 178.4, an FCB load module is assembled and added to SYS1.IMAGELIB. The image defines a print density of 8 lines per inch on an 11-inch form, with a right shift of 15 line character positions (1-1/2 inches).

```
//ADDFCB        JOB  MSGLEVEL=1
//STEP          EXEC PROC=ASMFCL,PARM.ASM='NODECK,LOAD',
//                   PARM.LKED='LIST,OL,REFR,RENT,XREF'
//ASM.SYSIN     DD   *
FCB2ID1         CSECT
*THIS EXAMPLE IS FOR A FORM LENGTH OF 11 INCHES WITH 8 LPI (88 LINES)
                DC   X'80'     THIS IS A DEFAULT IMAGE
                DC   AL1(89)   LENGTH OF FCB IMAGE AND INDEXING BYTE
                DC   X'8F'     OFFSET 15 CHARACTERS TO THE RIGHT
                DC   X'10'     8 LINES PER INCH-NO CHANNEL FOR LINE 1
                DC   XL4'0'    4 LINES NO CHANNEL
                DC   X'01'     CHANNEL 1 IN LINE 6
                DC   XL6'0'    6 LINES NO CHANNEL
                DC   X'02'     CHANNEL 2 IN LINE 13
                DC   XL6'0'    6 LINES NO CHANNEL
                DC   X'03'     CHANNEL 3 IN LINE 20
                DC   XL6'0'    6 LINES NO CHANNEL
                DC   X'04'     CHANNEL 4 IN LINE 27
                DC   XL6'0'    6 LINES NO CHANNEL
                DC   X'05'     CHANNEL 5 IN LINE 34
                DC   XL6'0'    6 LINES NO CHANNEL
                DC   X'06'     CHANNEL 6 IN LINE 41
                DC   XL6'0'    6 LINES NO CHANNEL
                DC   X'07'     CHANNEL 7 IN LINE 48
                DC   XL6'0'    6 LINES NO CHANNEL
                DC   X'08'     CHANNEL 8 IN LINE 55
                DC   XL6'0'    6 LINES NO CHANNEL
                DC   X'09'     CHANNEL 9 IN LINE 62
                DC   XL6'0'    6 LINES NO CHANNEL
                DC   X'0A'     CHANNEL 10 IN LINE 69
                DC   XL6'0'    6 LINES NO CHANNEL
                DC   X'0B'     CHANNEL 11 IN LINE 76
                DC   XL6'0'    6 LINES NO CHANNEL
                DC   X'0C'     CHANNEL 12 IN LINE 83
                DC   XL4'0'    4 LINES NO CHANNEL
                DC   X'10'     POSITION 88 LAST LINE IN IMAGE
                END
/*
//LKED.SYSLMOD  DD   DSNAME=SYS1.IMAGELIB(FCB2ID1),DISP=OLD,
//                   SPACE=    (OVERRIDE SECONDARY ALLOCATION)
```

Figure 46. Sample Code to Assemble and Add an FCB Load Module to SYS1.IMAGELIB

Notes to Figure 46:

1. The RENT and REFR linkage editor attributes are used for performance considerations in a paging environment. They may be omitted.

2. Executing the ASMFCL procedure does not actually generate executable code. The assembler/linkage editor is used to place the FCB image into SYS1.IMAGELIB.

3. The SPACE parameter is overridden here because the ASMFCL cataloged procedure has secondary allocation specified. Elimination of the override causes the original secondary allocation amount to be used.

RETRIEVING AN FCB IMAGE FROM SYS1.IMAGELIB

If you want to modify an FCB image in virtual storage before loading it into a forms control buffer, you can use this sequence of macro instructions to read the FCB image into virtual storage.

1. An IMGLIB macro instruction, with the OPEN parameter

2. A BLDL macro instruction, to determine whether the FCB image you want is in the image library

3. A LOAD macro instruction, to load the image into virtual storage

After the image has been read in, you should issue the IMGLIB macro instruction with the CLOSE parameter and the address of the DCB that was built by the first IMGLIB macro. A SETPRT macro instruction can be used to load the forms control buffer with the modified image. Printers other than the 3800 will require the use of an FCB entry in an exit list, as described in *Data Management Services*.

The format of the BLDL and SETPRT macros is given in *Data Management Macro Instructions*; the format of the LOAD macro is given in *Supervisor Services and Macro Instructions*.

The format of the IMGLIB macro is shown below:

<table>
<tr><td>[<i>symbol</i>]</td><td><b>IMGLIB</b></td><td><b>{OPEN|CLOSE,<i>addr</i>}</b></td></tr>
</table>

OPEN
specifies that a DCB is to be built for SYS1.IMAGELIB and that SYS1.IMAGELIB is to be opened. The address of the DCB is returned in register 1.

CLOSE
specifies that SYS1.IMAGELIB is to be closed.

addr
specifies the RX-type address of the word which points to the DCB. If coded in the form (*reg*), then the register in parentheses contains the address of the DCB, not the address of the fullword.

Return codes from the IMGLIB OPEN macro are shown below:

Return Code	**Meaning**
0 (00)	Operation successful.
4 (04)	Either the volume containing SYS1.IMAGELIB is not mounted or a required catalog volume is not mounted.
8 (08)	Either SYS1.IMAGELIB does not exist on the volume to which the catalog points, or SYS1.IMAGELIB is not cataloged.
12 (0C)	An error occurred in reading the catalog or VTOC.

BLDL and LOAD are the only macros that may refer to the DCB built by the IMGLIB macro.

UCS ALIAS NAMES

The system assigns an alias for each installation-standard print chain not actually defined on a given printer. This provides JES2 with flexibility in scheduling printers for SYSOUT data sets. For example, a request for the 1403 TN train would be assigned the T11 train, if the data set were printed on a 3211. The assigned alias names, which follow the naming conventions currently used in SYS1.IMAGELIB, are:

IMAGE	ALIAS
UCS1AN	UCS1A11
UCS1HN	UCS1H11
UCS1PN	UCS1P11
UCS1TN	UCS1T11
UCS2A11	UCS2AN
UCS2H11	UCS2HN
UCS2P11	UCS2PN,UCS2RN,UCS2QN
UCS2T11	UCS2TN

The image and alias names are included in SYS1.IMAGELIB at system generation.

Some trains, such as SN and G11, do not have aliases because neither has an equivalent train on the other printer. An installation can assign an alias, if it so chooses. (See Linkage Editor and Loader for details about the ALIAS statement.) If an alias is supplied, JES2 will use it. If an alias is not supplied, an installation-defined SYSOUT class or a printer routing code (specified via the DEST parameter) should be used to assign the data set to the correct printer. If a SYSOUT class or a printer routing code is not used, and JES2 is directed to print a data set on a printer for which the proper image is not supplied, JES2 notifies the operator. The operator can then print the data set with a valid train or redirect the data set to the proper printer via the '$E' command.

If an installation defines a new train, it can supply an alias name for that train, via the linkage editor ALIAS statement, when including the image in SYS1.IMAGELIB.

THE 3211 INDEXING FEATURE

JES2 supports the 3211 Indexing Feature in two ways:

1. Specification of the INDEX parameter on the /*OUTPUT card.

2. The extended FCB image:

   JES2 supplies two special FCBs: FCB26 for 6 lines per inch and FCB28 for 8 lines per inch (specified as FCB=6 and FCB=8, respectively). These FCBs contain a channel 1 indication in position 1, a special index flag in the third byte, and the number of lines per inch in the fourth byte of the image.

   The special index flag in the third byte of FCB26 and FCB28 contains X'80' plus a binary index value, in the range 1 to 32 (default=1). The index value sets the left margin (1 indicates flush-left position; other values cause indentation of the print line by N-1 positions).

If any other FCB images are to be used by JES2, they must specify channel 1 in position 1; otherwise JES2 incorrectly positions the forms in the printer. (STD1 and STD2 do not specify channel 1 in position 1 and therefore must not be specified, unless altered, for JES2.)

If the third byte of any other FCB image contains a data character (specifying the number of lines per inch) other than X'80', JES2 uses that specification and supplies an index value of 1.

IBM 3203 MODEL 5 PRINTER

The IBM 3203 Model 5 Printer is treated the same as a 3211 printer by JES2, except that the 3203 Model 5 does not support the 3211 indexing feature, and any indexing commands from JES2 are ignored by the 3203 Model 5. The 3203 Model 5 uses 3211 FCB images and its own unique UCS images. UCS images are listed in System Generation Reference.

CHAPTER 10. CATALOG, SCRATCH, AND RENAME DUMMY MODULES

The load modules for CATALOG (SVC 26), SCRATCH (SVC 29), and RENAME (SVC 30) contain as their entry points the dummy modules IGG026DU, IGG029DU, and IGG030DU, respectively. These dummy modules immediately pass control to the first processing module for their respective SVCs without performing any processing themselves. The CATALOG dummy module IGG026DU receives control from SVC 26 and immediately passes control to module IGC0002F. The SCRATCH dummy module IGG029DU receives control from SVC 29 and immediately passes control to module IGC0002I. The RENAME dummy module, IGG030DU, receives control from SVC 30 and immediately passes control to IGC00030.

The load module for SCRATCH(SVC29) also contains the dummy module IGG029DM. The SCRATCH dummy module IGG029DM receives control from IGG0290D when an error return code of 4 or 8 is indicated, and immediately passes control to the location pointed to by register 14.

If you require special processing either before or after SVC 26, 29, or 30, you replace the appropriate dummy module(s) with your own module(s). Your replacement modules must follow all the characteristics and programming conventions for SVC routines. For information on writing SVC routines, characteristics of SVC routines, programming conventions for SVC routines, and inserting SVC routines, see System Programming Library: Supervisor Services and Macro Instructions. Your modules may replace IGG026DU, IGG029DU, IGG029DM, and IGG030DU in SYS1.AOSD0 prior to system generation, or you may replace the dummy modules in SYS1.LPALIB after system generation. Information on how to replace the dummy modules with your modules can be obtained from the appropriate link-edit step of the STAGE I system generation output. You may also obtain link-edit information from the STAGE I system generation macro SGIEC4DM in SYS1.AGENLIB. You may apply PTFs to CATALOG, SCRATCH, or RENAME with SMP without modifying your own versions of IGG026DU, IGG029DU, IGG029DM,and IGG030DU.

The prolog of each of the dummy modules contains register conventions and other information about these modules.

CHAPTER 11. SPECIFYING BUFFER NUMBERS FOR DASD DATA SETS

The BUFNO keyword in the DCB macro and the BUFNO subparameter of the DCB keyword in the DD statement determine how many buffers are allocated when accessing a partitioned or sequential data set using QSAM. The NCP keyword in the DCB macro determines how many un-CHECKed READ or WRITE macro instructions are allowed when accessing a sequential or partitioned data set using BSAM; one buffer is used for each READ or WRITE macro instruction.

The sequential access method can construct a channel program to transfer up to 30 buffers or 240,000 bytes of data, whichever is less. If BUFNO or NCP is less than 30, no more than that number of buffers can be transferred with a single channel program.

BUFNO is defaulted in OPEN to 5 if it is not specified for a QSAM DCB; NCP is defaulted to 1 in OPEN if it is not specified. The QSAM access method manages buffers. The user program must manage buffers when it uses BSAM.

PERFORMANCE CONSIDERATIONS

Buffer number and block size influence the rate with which data can be transferred and the operating system overhead per block. The use of more buffers reduces (per block transferred) the EXCP and IOS overhead and the time waiting for the DASD device to seek to the requested cylinder and rotate to the requested record (device latency time). However, if more buffers are allocated than a program can effectively process, the virtual pages containing those buffers will be paged out, effectively adding to the system overhead for the job. A large number of buffers also cause a large amount of real storage to be allocated to the job while the data is being transferred.

A job in a low-performance group may get swapped out more frequently than a higher priority job. The number of buffers allocated for the job contributes to the number of pages which have to be swapped out.

Programs that access data sets with small block size (for example, 80) can easily make effective use of 30 buffers which fit in, at most, two 4096-byte pages. The advantage of 30 buffers over the default of five buffers is great: one channel program versus six channel programs to transfer 30 blocks.

At the other end of the spectrum, usage of data sets with large blocking factors such as full-track blocking on 3350 or half-track blocking on 3380 can still be effective when only 3 or 4 buffers, rather than 5 or more, are specified. The slightly lower DASD performance and small increase in EXCP and IOS instruction costs should be more than offset by a reduction in paging or swapping in a constrained environment.

It can be seen that proper selection of buffer number can have a positive effect on the elapsed time of a job and the system overhead associated with the job. The DCB OPEN installation exit can use installation criteria for a default buffer number for QSAM DCBs (see "DCB Open Installation Exit" on page 125 for a description of the open installation exit). The NCP field of the DCB must be set by the program for BSAM DCBs.

CVAFDIR MACRO

SYNTAX

<table>
<tr><td>[label]</td><td>CVAFDIR</td><td>ACCESS=READ|WRITE|RLSE
[,DSN=addr]
[,BUFLIST=addr]
[,VERIFY=YES|NO]
[,DEB=addr|UCB=addr]
[,IOAREA=KEEP|(KEEP,addr)NOKEEP|
(NOKEEP,addr)]
[,MAPRCDS=YES|(YES,addr)|NO|
(NO,addr)]
[,IXRCDS=KEEP|(KEEP,addr)|NOKEEP|
(NOKEEP,addr)]
[,BRANCH=YES[1]|NO|(YES,SUP)|(YES,PGM)]
[,MF=I|L|(E,addr)]</td></tr>
</table>

[1] The default is SUP if YES is coded.

ACCESS: READ OR WRITE A DSCB OR VIR(S), OR RELEASE BUFFER LISTS

When ACCESS is READ or WRITE, a single DSCB is accessed for an indexed or nonindexed VTOC, or one or more VIRs are accessed for an indexed VTOC.

ACCESS=READ

Specifies that a single DSCB or one or more VIR(s) are to be read into a buffer whose address is in a buffer list.

If the buffer list if for a DSCB, only one entry is used in the buffer list. The first entry with the skip bit set to zero and with a nonzero buffer address is used.

All VIR(s) whose buffer list entry has the skip bit off will be read into a buffer.

DSN and BUFLIST are required if ACCESS=READ for a DSCB buffer list.

ACCESS=WRITE

Specifies that a single DSCB or one or more VIRs are to be written from buffer(s) whose address is in a buffer list.

WRITE is permitted with BRANCH=NO only if the caller is authorized by APF.

DSN and BUFLIST are required if ACCESS=WRITE for a DSCB buffer list.

If any buffer list entry has its modified bit set, only those entries with the modified bit set will be written. If no modify bits are on, all VIRs will be written.

ACCESS=RLSE

Applies only to VIR buffer lists. It requests the release of one or more buffers in the VIR buffer list chain identified in the BUFLIST keyword, and the release of each buffer list for which all buffers are released.

DSN and BUFLIST are not required if ACCESS=RLSE.

Only buffers in the buffer list with the skip bit set to zero and with a nonzero buffer address are released. The buffer list is not released if any entry has the skip bit set to one.

For an indexed VTOC, if ACCESS=RLSE is coded, buffer lists and buffers pointed to by the BUFLIST keyword will be released, as well as buffer lists supplied in the CVAF parameter list CVMRCDS and CVIRCDS fields. If the CVMRCDS or the CVIRCDS buffers are supplied in the BUFLIST field, either directly or indirectly through chaining, the keyword MAPRCDS=YES, IXRCDS=KEEP, or MAPRCDS=(NO,0), IXRCDS=(NOKEEP,0) must be coded to prevent CVAF from freeing the buffers more than once. If buffers are released, the CVAF parameter list field pointing to the buffer list will be updated.

DSN: SPECIFY THE NAME OF THE DSCB

DSN=<u>addr</u>

DSN specifies the address of a 44-byte data set name of the DSCB to be accessed.

DSN is required if ACCESS=READ or WRITE and the request is to read or write a DSCB. If a 140-byte DSCB is specified, the validity of the storage location is checked but its contents are ignored.

BUFLIST: SPECIFY ONE OR MORE BUFFER LISTS

BUFLIST=<u>addr</u>

The BUFLIST keyword contains the address of a buffer list used to read or write a DSCB or VIRs.

VERIFY: VERIFY THAT A DSCB IS A FORMAT-0 DSCB

<u>**VERIFY=YES**</u>

CVAF will verify that the DSCB is a format-0 DSCB before writing the DSCB. The first four bytes of the key will be compared with binary zeros. If the key does not start with four bytes of zeros, the DSCB will not be written and an error code will be returned.

VERIFY=NO

CVAF will not test the key of the DSCB.

Note: VERIFY applies only when writing a 140-byte DSCB. VERIFY is ignored when a VIR is written.

BRANCH: SPECIFY THE ENTRY TO THE MACRO

BRANCH=(YES,SUP)

Requests that the branch entry to CVAFDIR be used. You must be in supervisor state. Protect key checking is bypassed.

An 18-word save area must be supplied if BRANCH=YES is coded. No lock may be held on entry to CVAF. SRB mode is not allowed.

BRANCH=YES

Equivalent to BRANCH=(YES,SUP), because SUP is the default when YES is coded. Protect key checking is bypassed.

BRANCH=(YES,PGM)

Requests the branch entry. You must be authorized by APF and be in problem state. Protect key checking is bypassed.

BRANCH=NO
Requests the SVC entry. You must be authorized by APF if any output operations are requested. Protect key checking is performed.

DEB|UCB: SPECIFY THE VTOC TO BE ACCESSED

DEB=addr
Supplies the address of a DEB opened to the VTOC to be accessed. CVAF will not allow output requests to the VTOC or VTOC index if DEB is supplied. No asynchronous requests may be performed by an unauthorized caller against the DEB (such as EXCP, CLOSE, EOV), because CVAF will remove the DEB from the DEB table for the duration of the CVAF call. An unauthorized caller (neither APF authorized nor in a system key) must supply a DEB and not a UCB to CVAF. The unauthorized caller's DEB must have been created under the current task by either SAM or EXCP.

UCB=addr
Supplies the address of the UCB for the unit whose VTOC is to be accessed. An unauthorized caller must not use this parameter.

If the address of a previously obtained I/O area is supplied through the IOAREA keyword, neither UCB nor DEB need be supplied. Otherwise, either a UCB or DEB must be supplied. If a UCB address is supplied, it will be overlaid in the CVPL by the UCB address present in the I/O area.

If DEB and UCB are supplied in the CVPL, the DEB address will be used and the UCB address will be overlaid in the CVPL by the UCB address in the DEB.

IOAREA: KEEP OR FREE THE I/O WORKAREA

IOAREA=KEEP
Specifies the CVAF I/O area associated with the CVAF parameter list is to be kept upon completion of the CVAF request. IOAREA=KEEP may be coded with BRANCH=NO only if the caller is authorized (APF, or system key).

If IOAREA=KEEP is coded, the caller must issue CVAF with IOAREA=NOKEEP specified at some future time, whether or not any further VTOC access is required: for example, the recovery routine of the caller of CVAF.

Coding IOAREA=KEEP allows subsequent CVAF requests to be more efficient, as certain initialization functions can be bypassed. Neither DEB nor UCB need be specified when a previously obtained CVAF I/O area is supplied; neither can they be changed.

When IOAREA=KEEP is first issued, CVAF returns the CVAF I/O area in the CVAF parameter list (CVIOAR). Subsequent calls of CVAF may use that same parameter list, and CVAF will obtain its I/O area from the CVIOAR.

When processing on the current volume is finished, release all areas that were kept.

IOAREA=(KEEP,addr)
Provides the address of a previously obtained I/O area. If a different CVAF parameter list is used, the previously obtained I/O area may be passed to CVAF by coding its address as the second parameter of the IOAREA keyword.

IOAREA=NOKEEP
Causes the work area to be freed upon completion of the CVAF request.

IOARECA=(NOKEEP,addr)

Causes a previously obtained work area to be freed upon completion of the CVAF request.

MF: SPECIFY THE FORM OF THE MACRO

This keyword specifies whether the list, execute, or normal form of the macro is requested.

MF=I

If I is coded, or neither L nor E is coded, the CVAF parameter list is generated and CVAF is called. This is the normal form of the macro.

MF=L

L indicates the list form of the macro. A parameter list is generated, but CVAF is not called.

MF=(E,addr)

E indicates the execute form of the macro. The CVAF parameter list whose address is in 'addr' can be modified by this form of the macro.

MAPRCDS: KEEP OR FREE MAPRCDS BUFFER LIST AND BUFFERS

This keyword applies to an indexed VTOC only and specifies the disposition of the MAPRCDS buffer list and buffers.

MAPRCDS=YES

Specifies that the buffer list and buffers are to be retained at the end of processing.

If no buffer list address is in the CVAF parameter list, CVAF will read the MAP VIRs into buffers it obtains. The buffer list that contains the address and RBAs of the VIRs can be accessed after processing from the CVAF parameter list field, CVMRCDS. The buffer list and VIR buffers are in your protect key: subpool 0 if you are not authorized; 229 if you are.

When processing on the current volume is finished, release all areas that were kept.

MAPRCDS=(YES,addr)

If YES is coded and the buffer list address (CVMRCDS in CVAF parameter list) is supplied, VIRs are not read.

The CVMRCDS buffer list used in CVAFDIR macro can be passed to another CVAF macro call through the MAPRCDS keyword.

If MAPRCDS=YES is coded for an unindexed VTOC, the function is performed, but an error code will be returned.

MAPRCDS=NO

If MAPRCDS=NO is coded, all the buffers without the skip bit on in the buffer list whose address is in the CVMRCDS field of the CVPL will be freed. If all the buffers are freed, the buffer list will also be freed.

MAPRCDS=(NO,addr)

Causes buffer lists and buffers previously obtained by CVAF to be freed.

You must free buffer lists and buffers obtained by CVAF. This can be done in three ways:

- By coding MAPRCDS=NO on the CVAFDIR macro that obtained the buffers
- By coding MAPRCDS=NO on a subsequent CVAF macro
- By coding CVAFDIR ACCESS=RLSE and providing the address of the buffer list in the BUFLIST keyword

Note: To maintain the integrity of MAP records read, you must enqueue the VTOC and reserve the unit.

IXRCDS: RETAIN VIERS IN VIRTUAL STORAGE

This keyword applies to indexed VTOCs only.

IXRCDS=KEEP
Specifies that VIERs read into storage are to be kept in virtual storage. The VIERs are retained even if processing cannot complete successfully. The CVAF parameter list in field CVIRCDS will have the address of a buffer list containing the VIR buffer addresses and RBAs of the VIERs read.

Index search function will dynamically update the buffer list and, when necessary, obtain additional buffer lists and chain them together.

If KEEP is specified and no buffer list is supplied to CVAF in the CVPL, CVAF will obtain a buffer list and buffers and read the first high-level VIER. The address of the buffer list is placed in the CVMICDS field of the CVPL. The first high-level VIER will be checked for the VXFHLV bit and to see if the VXVISE bit is off.

The buffer list and VIR buffers are in your protect key. The subpool is 0 if you are not authorized; subpool 229 if you are.

If IXRCDS=KEEP is coded for an nonindexed VTOC, a request to read or write a DSCB will be performed, but an error code will be returned.

When processing on the current volume is finished, release all areas that were kept.

IXRCDS=(KEEP,__addr__**)**
The index records buffer list address from one CVAF request is being passed to this CVAF parameter list by specifying its address as the second parameter in the IXRCDS keyword.

__**IXRCDS=NOKEEP**__
If NOKEEP is coded, the VIERs that are accessed (if any) are not retained. Furthermore, the buffer list supplied in the CVIRCDS field in the CVAF parameter list is released, as are all buffers found in the buffer list. If the skip bit is set in any entry in the buffer list, the buffer and buffer list will not be freed.

IXRCDS=(NOKEEP,__addr__**)**
Specifies that previously accessed VIERs are not to be retained.

You must free buffer lists and buffers obtained by CVAF. This can be done in three ways:

- By coding IXRCDS=NOKEEP on the CVAFDIR macro that obtained the buffers
- By coding IXRCDS=NOKEEP on a subsequent CVAF macro
- By coding CVAFDIR ACCESS=RLSE and providing the address of the buffer list in the BUFLIST keyword

Note: To maintain the integrity of the VIERs read, you must enqueue the VTOC and reserve the unit.

CVAFDSM MACRO

SYNTAX

[label]	CVAFDSM	ACCESS=MAPDATA ,MAP=INDEX|VOLUME|VTOC [,EXTENTS=addr] [,MAPRCDS=YES[1]|(YES,addr)|NO[2]| (NO,addr)] [,UCB=addr|DEB=addr] [,COUNT=YES|NO] [,CTAREA=addr] [,IOAREA=KEEP|(KEEP,addr)|NOKEEP| (NOKEEP,addr)] [,BRANCH=NO|YES[3]|(YES,SUP)|(YES,PGM)] [,MF=I|L|(E,addr)]

[1] Default if MF=I.

[2] Default if MF=L or MF=(E,addr).

[3] Default is SUP if YES is coded.

ACCESS=MAPDATA: REQUEST INFORMATION FROM THE INDEX SPACE MAPS

ACCESS=MAPDATA

Obtains data from the index space maps. Three kinds of data are available:

- The number of format-0 DSCBs (the data is obtained from the VTOC map of DSCBs)
- The number of unallocated VIRs in the index (the data is obtained from the VTOC index map)
- The number (and location) of extents of unallocated pack space (the data is obtained from the VTOC pack space map)

MAPRCDS: KEEP OR FREE MAPRCDS BUFFER LIST AND BUFFERS

MAPRCDS=YES

Specifies that the buffer list and buffers are to be retained at the end of the function.

If YES is specified and no buffer list is supplied through the CVAF parameter list, CVAF will read the MAP VIRs into buffers obtained by CVAF. The buffer list that contains the address and RBAs of the VIRs can be accessed after the CVAF call from the CVAF parameter list field, CVMRCDS. The buffer list and VIR buffers are in the caller's protect key: subpool 0 if the caller is not authorized; subpool 229, if the caller is authorized.

YES is the default if MF=I is specified or defaulted.

When processing on the current volume is finished, release all areas that were kept.

MAPRCDS=(YES,addr)

If YES is coded but the buffer list address (CVMRCDS in CVAF parameter list) is supplied, the VIRs are not read.

The CVMRCDS buffer list from one CVAF call can be passed to another CVAF macro call through the MAPRCDS keyword.

MAPRCDS=NO
If MAPRCDS=NO is coded, the MAP records buffers and buffer list will be freed upon completion of the CVAFDSM function.

NO is the default if MF=L is specified.

MAPRCDS=(NO,addr)
Causes buffer lists and buffers previously obtained by CVAF to be freed.

Buffer lists and buffers obtained by CVAF must be freed by the caller. This can be done in three ways:

- By coding MAPRCDS=NO on the call that obtained the buffers
- By coding MAPRCDS=NO on a subsequent CVAF call
- By calling CVAFDIR ACCESS=RLSE and providing the buffer list in the BUFLIST keyword

If MF=(E,addr) is coded and MAPRCDS is not coded, the parameter list value of MAPRCDS is not changed.

Note: To maintain the integrity of the MAP records read, you must enqueue the VTOC and reserve the unit.

MAP: IDENTIFY THE MAP TO BE ACCESSED

MAP=INDEX
Specifies the VTOC index map (VIXM) is to be accessed and a count of unallocated VIRs returned. COUNT=YES must also be coded.

MAP=VOLUME
Specifies the VTOC pack space map (VPSM) is to be accessed and information on unallocated extents of pack space returned. EXTENTS=addr and COUNT=NO must also be coded.

MAP=VTOC
Specifies the VTOC map of DSCBs (VMDS) is to be accessed and a count of format-0 DSCBs returned. COUNT=YES must also be coded.

EXTENTS: IDENTIFY WHERE EXTENTS FROM THE VPSM ARE RETURNED

EXTENTS=addr
If one or more extents from the VPSM are requested, EXTENTS is the address of a 1-byte count field containing the number of 5-byte extents that follow. In the first two bytes of the first 5-byte extent, you must supply the relative track address (RTA) at which CVAF should start the VPSM search. The first extent area is updated with information on the next free extent found that has a higher starting RTA than that supplied. Each subsequent extent area is filled in with information on free space extents (in ascending track address order).

Information on free extents has the format, XXYYZ, where:

- XX is the relative track address of the first track of the extent.
- YY is the number of whole cylinders in the extent.
- Z is the number of additional tracks in the extent.

Only XX is supplied by the caller in the first extent area. CVAF will start searching the VPSM at relative track address XX.

If all the unallocated extents in the VPSM are provided before filling in all the supplied extent areas, the remaining extent areas are set to zero. Register 15 is set to 4 on return, with the CVSTAT field in the CVPL set to X'20' to indicate end of data.

DEB|UCB: SPECIFY THE VTOC TO BE ACCESSED

UCB=_addr_
Supplies the address of the UCB for the unit whose VTOC is to be accessed. An unauthorized caller may not supply a UCB to CVAF.

DEB=_addr_
Supplies the address of a DEB opened to the VTOC to be accessed. CVAF will not allow output requests to the VTOC or VTOC index if DEB is supplied. No asynchronous requests may be performed by an unauthorized caller against the DEB (such as EXCP, CLOSE, EOV), because CVAF will remove the DEB from the DEB table for the duration of the CVAF call. An unauthorized caller (neither APF authorized nor in a system key) must supply a DEB and not a UCB to CVAF. The unauthorized caller's DEB must have been created under the current task by either SAM or EXCP.

If a previously obtained CVAF I/O area is supplied through the IOAREA keyword, neither UCB nor DEB need be supplied. Otherwise, either a UCB or DEB must be supplied. If a UCB address is supplied, it will be overlaid in the CVPL with the UCB address in the I/O area.

If DEB and UCB are supplied in the CVPL, the DEB will be used, and the UCB address supplied will be overlaid in the CVPL with the UCB address obtained from the DEB.

COUNT: OBTAIN A COUNT OF UNALLOCATED DSCBS OR VIRS

COUNT=YES
Indicates a count of unallocated DSCBs or VIRs in the designated space map is requested. MAP=VTOC or MAP=INDEX must be specified if COUNT=YES is coded.

COUNT=NO
Indicates a count of unallocated DSCBs or VIRs is not desired but, rather, information on free space on the pack is desired. MAP=VOLUME must be coded if COUNT=NO is coded or defaulted.

CTAREA: SUPPLY A FIELD TO CONTAIN THE NUMBER OF FORMAT-0 DSCBS

CTAREA=_addr_
Gives the address of a 4-byte field to contain the number of format-0 DSCBs when COUNT=YES, MAP=VTOC is specified; or the number of unallocated VIRs in the VTOC index when COUNT=YES, MAP=INDEX is specified.

IOAREA: KEEP OR FREE THE I/O WORK AREA

IOAREA=KEEP
Specifies the CVAF I/O area associated with the CVAF parameter list is to be kept upon completion of the CVAF request. IOAREA=KEEP may be coded with BRANCH=NO only if the caller is authorized (APF, or system key).

If IOAREA=KEEP is coded, the caller must issue CVAF with IOAREA=NOKEEP specified at some future time, whether or not any further VTOC access is required: for example, the recovery routine of the caller of CVAF.

Coding IOAREA=KEEP allows subsequent CVAF requests to be more efficient, as certain initialization functions can be bypassed. Neither DEB nor UCB need be specified when a previously obtained CVAF I/O area is supplied; neither can they be changed.

When IOAREA=KEEP is first issued, CVAF returns the CVAF I/O area in the CVAF parameter list (CVIOAR). Subsequent calls of CVAF may use that same parameter list, and CVAF will obtain its I/O area from the CVIOAR.

When processing on the current volume is finished, release all areas that were kept.

IOAREA=(KEEP,<u>addr</u>)
Provides the address of a previously obtained I/O area. If a different CVAF parameter list is used, the previously obtained CVAF I/O area may be passed to CVAF by coding its address as the second parameter of the IOAREA keyword.

<u>IOAREA=NOKEEP</u>
Causes the work area to be freed upon completion of the CVAF request.

IOAREA=(NOKEEP,<u>addr</u>)
Causes a previously obtained work area to be freed upon completion of the CVAF request.

BRANCH: SPECIFY THE ENTRY TO THE MACRO

BRANCH=(YES,SUP)
Requests that the branch entry to CVAFDIR be used. The caller must be in supervisor state. Protect key checking is bypassed.

An 18-word save area must be supplied if BRANCH=YES is coded. No lock may be held on entry to CVAF. SRB mode is not allowed.

BRANCH=YES
Equivalent to BRANCH=(YES,SUP), because SUP is the default when YES is coded. Protect key checking is bypassed.

BRANCH=(YES,PGM)
Requests the branch entry. The caller must be APF authorized and in problem state. Protect key checking is bypassed.

<u>BRANCH=NO</u>
Requests the SVC entry. The caller must be APF authorized if any output operations are requested. Protect key checking is performed.

MF: SPECIFY THE FORM OF THE MACRO

This keyword specifies whether the list, execute, or normal form of the macro is requested.

<u>MF=I</u>
If I is coded, or neither L nor E is coded, the CVAF parameter list is generated as is code to call CVAF. This is the normal form of the macro.

MF=L
L indicates the list form of the macro. A parameter list is generated, but code to call CVAF is not generated.

MF=(E,<u>addr</u>)
E indicates the execute form of the macro. The remote CVAF parameter list supplied as 'addr' is used in, and can be modified by, the execute form of the macro.

CVAFSEQ MACRO

SYNTAX

[label]	CVAFSEQ	ACCESS=GT|GTEQ [,BUFLIST=addr] [,DSN=addr] [,UCB=addr|DEB=addr] [,DSNONLY=NO|YES] [,ARG=addr] [,IOAREA=KEEP|(KEEP,addr)|NOKEEP| (NOKEEP,addr)] [,IXRCDS=KEEP|(KEEPaddr)|NOKEEP| (NOKEEP,addr)] [,BRANCH=NO|YES[1]|(YES,SUP)|(YES,PGM)] [,MF=I|L|(E,addr)]

[1] If YES, default is SUP.

ACCESS: SPECIFY RELATIONSHIP BETWEEN SUPPLIED AND RETURNED DSN

ACCESS=GT
Specifies that the DSN or argument value is to be used to return a DSCB whose DSN or argument is greater than that supplied.

ACCESS=GTEQ
Specifies that the DSN or argument value is to be used to return a DSCB whose DSN or argument is greater than or equal to that supplied.

Note: A CVAF call specifying ACCESS=GTEQ should be followed by an ACCESS=GT request, or the same DSCB or name will be returned.

BUFLIST: SPECIFY ONE OR MORE BUFFER LISTS

BUFLIST=addr
The BUFLIST keyword supplies the address of a buffer list used to read or write DSCBs and VIRs.

DSN: SPECIFY ACCESS BY DSN ORDER OR BY PHYSICAL-SEQUENTIAL ORDER

DSN=addr
Specifies that access of an indexed VTOC is by DSN order. BUFLIST is required if DSNONLY=NO is coded or defaulted.

DSN omitted
If you omit the DSN keyword, access of an indexed or nonindexed VTOC is by physical-sequential order. BUFLIST is required.

UCB|DEB: SPECIFY THE VTOC TO BE ACCESSED

UCB=addr
Supplies the address of the UCB for the unit whose VTOC is to be accessed. An unauthorized caller may not supply a UCB to CVAF.

DEB=addr
Supplies the address of a DEB opened to the VTOC to be accessed. CVAF will not allow output requests to the VTOC or VTOC index if DEB is supplied. No asynchronous requests may be performed by an unauthorized caller against the DEB (such as EXCP, CLOSE, EOV) because CVAF will remove the DEB from the DEB table for the duration of the CVAF call. An

unauthorized caller (neither APF authorized nor in a system key) must supply a DEB and not a UCB to CVAF. The unauthorized caller's DEB must have been created under the current task by either SAM or EXCP.

If a previously obtained CVAF I/O area is supplied through the IOAREA keyword, neither UCB nor DEB need be supplied.

Otherwise, either a UCB or DEB must be supplied. If a UCB address is supplied, it will be overlaid in the CVPL with the UCB address in the I/O area.

If DEB and UCB are supplied in the CVPL, the DEB will be used, and the UCB address supplied will be overlaid in the CVPL with the UCB address obtained from the DEB.

DSNONLY: SPECIFY THAT ONLY THE DATA SET NAME BE READ

This keyword is applicable only to accessing an indexed VTOC in DSN order.

DSNONLY=NO
Requests that the data set name be obtained from the VTOC index and the DSCB be read into a buffer supplied through the BUFLIST keyword. BUFLIST is required.

DSNONLY=YES
Requests that only the data set name be obtained from the VTOC index. If the ARG keyword is coded, the argument of the DSCB is returned.

ARG: SPECIFY WHERE THE ARGUMENT OF THE DSCB IS TO BE RETURNED

This keyword is applicable only to accessing an indexed VTOC in DSN order with DSNONLY=YES coded.

ARG=*addr*
Provides the address of the 5-byte area at which the CCHHR of each data set name in the VTOC index is returned when DSNONLY=YES is coded.

IOAREA: KEEP OR FREE THE I/O WORK AREA

IOAREA=KEEP
Specifies the CVAF I/O area associated with the CVAF parameter list is to be kept upon completion of the CVAF request. IOAREA=KEEP may be coded with BRANCH=NO only if the caller is authorized (APF, or system key).

If IOAREA=KEEP is coded, the caller must issue CVAF with IOAREA=NOKEEP specified at some future time, whether or not any further VTOC access is required: for example, the recovery routine of the caller of CVAF.

Coding IOAREA=KEEP allows subsequent CVAF requests to be more efficient, as certain initialization functions can be bypassed. Neither DEB nor UCB need be specified when a previously obtained CVAF I/O area is supplied; neither can they be changed.

When IOAREA=KEEP is first issued, CVAF returns the CVAF I/O area in the CVAF parameter list (CVIOAR). Subsequent calls of CVAF may use that same parameter list, and CVAF will obtain its I/O area from the CVIOAR.

When processing on the current volume is finished, release all areas that were kept.

IOARENA=(KEEP,addr)

Provides the address of a previously obtained I/O area. If a different CVAF parameter list is used, the previously obtained CVAF I/O area may be passed to CVAF by coding its address as the second parameter of the IOAREA keyword.

IOAREA=NOKEEP
Causes the work area to be freed upon completion of the CVAF request.

IOAREA=(NOKEEP,addr)
Causes a previously obtained work area to be freed upon completion of the CVAF request.

IXRCDS: RETAIN VIERS IN VIRTUAL STORAGE

This keyword applies to an indexed VTOC only.

IXRCDS=KEEP
Specifies that the VIERs read into storage during the CVAF function are to be kept in virtual storage. The VIERs are retained even if the index function is unsuccessful. The VIERs are accessed from the CVAF parameter list (CVIRCDS). CVIRCDS is the address of a buffer list containing the VIR buffer addresses and RBAs of the VIERs read.

Index search function will dynamically update the buffer list and, when necessary, obtain additional buffer lists and chain them together.

If KEEP is specified and no buffer list is supplied to CVAF in the CVPL, CVAF will obtain a buffer list and buffers and read the first high-level VIER. The address of the buffer list is placed in the CVIRCDS field of the CVPL. The first high-level VIER will be checked for the VXFHLV bit and to see if the VXVISE bit is off.

The buffer list and VIR buffers are in the caller's protect key. The subpool is 0 if the caller is not authorized; subpool 229, if the caller is authorized.

If IXRCDS=KEEP for an nonindexed VTOC, a request to read a DSCB may be performed, but an error code will be returned.

When processing on the current volume is finished, release all areas that were kept.

IXRCDS=(KEEP,addr)
The CVIRCDS from one CVAF call can be passed to another CVAF parameter list by specifying the address as the second parameter in the IXRCDS keyword.

IXRCDS=NOKEEP
If NOKEEP is coded, the VIERs which are accessed (if any) are not retained. Furthermore, the buffer list supplied in the CVIRCDS field in the CVAF parameter list is released, as are all buffers found in the buffer list. If the skip bit is set in any entry in the buffer list, the buffer and buffer list will not be freed.

IXRCDS=(NOKEEP,addr)
Specifies that previously accessed VIERs are not to be retained.

You must free buffer lists and buffers obtained by CVAF. This can be done in three ways:

- By coding IXRCDS=NOKEEP on the CVAFSEQ macro that obtained the buffers
- By coding IXRCDS=NOKEEP on a subsequent CVAF macro
- By coding CVAFDIR ACCESS=RLSE and providing the address of the buffer list in the BUFLIST keyword

Note: To maintain the integrity of the VIERs read, you must enqueue the VTOC and reserve the unit.

BRANCH: SPECIFY THE ENTRY TO THE MACRO

BRANCH=(YES,SUP)
Requests that the branch entry to CVAFDIR be used. The caller must be in supervisor state. Protect key checking is bypassed.

An 18-word save area must be supplied if BRANCH=YES is coded. No lock may be held on entry to CVAF. SRB mode is not allowed.

BRANCH=YES
Equivalent to BRANCH=(YES,SUP), because SUP is the default when YES is coded. Protect key checking is bypassed.

BRANCH=(YES,PGM)
Requests the branch entry. The caller must be APF authorized and in problem state. Protect key checking is bypassed.

BRANCH=NO
Requests the SVC entry. The caller must be APF authorized if any output operations are requested. Protect key checking is performed.

MF: SPECIFY THE FORM OF THE MACRO

This keyword specifies whether the list or execute or normal form of the macro is requested.

MF=I
If I is coded, or neither L nor E is coded, the CVAF parameter list is generated as is code to call CVAF. This is the normal form of the macro.

MF=L
L indicates the list form of the macro. A parameter list is generated, but code to call CVAF is not generated.

MF=(E,addr)
E indicates the execute form of the macro. The remote CVAF parameter list supplied as 'addr' is used in and can be modified by the execute form of the macro.

CVAFTST MACRO

SYNTAX

[label]	CVAFTST	UCB=addr

UCB: SPECIFY THE VTOC TO BE TESTED

UCB=addr
Supplies the address of the UCB for the volume whose VTOC is to be tested.

APPENDIX B. EXAMPLES OF VTOC ACCESS MACROS

The examples that follow are partial assembly listings which include expansions of each VTOC access macro. The expansions are provided to show how the VTOC macros can be substituted for existing procedures.

EXAMPLE 1: USING THE CVAFDIR MACRO WITH AN INDEXED OR NONINDEXED VTOC

This example uses the CVAFDIR macro to read a DSCB of a given data set name and determines whether the DSCB is for a partitioned data set. The address of the 44-byte data set name is supplied to the program in register 5 (labeled RDSN in the example). The address of a DEB open to the VTOC is supplied to the program in register 4 (labeled RDEB in the example).

The buffer list is in the program and is generated by the ICVAFBFL macro. The DSCB buffer is in the program and is generated by the IECSDSL1 macro.

```
EXAMPLE1 CSECT
         STM   14,12,12(RSAVE)
         BALR  12,0
         USING *,12
         ST    RSAVE,SAVEAREA+4
         LA    RWORK,SAVEAREA
         ST    RWORK,8(,RSAVE)
         LR    RSAVE,RWORK
************************************************************
*
*        REGISTERS
*
************************************************************
REG1     EQU   1                  REGISTER 1
RWORK    EQU   3                  WORK REGISTER
RDEB     EQU   4                  DEB ADDRESS
RDSN     EQU   5                  ADDRESS OF DATA SET NAME
RSAVE    EQU   13                 SAVE AREA ADDRESS
REG15    EQU   15                 RETURN CODE REGISTER 15
************************************************************
*
*        RETURN CODES
*
************************************************************
PDSRTN   EQU   0                  DATA SET A PDS RETURN CODE
NOTFND   EQU   4                  DATA SET NOT FOUND RETURN CODE
NOTPDS   EQU   8                  DATA SET NOT A PDS RETURN CODE
UNEXPECD EQU   12                 UNEXPECTED ERROR RETURN CODE
************************************************************
*
*        READ DSCB INTO DS1FMTID.
*        DATASET NAME ADDRESS SUPPLIED IN RDSN.
*        ADDRESS OF DEB OPEN TO VTOC SUPPLIED IN RDEB.
*        DETERMINE IF DATA SET IS A PARTITIONED DATA SET.
*        THIS PROGRAM IS NEITHER REENTRANT NOR REUSABLE.
*
************************************************************
         XC   BUFLIST(BFLHLN+BFLELN),BUFLIST ZERO BUFFER LIST
         OI   BFLHFL,BFLHDSCB      DSCBS TO BE READ WITH BUFFER LIST
         MVI  BFLHNOE,1            ONE BUFFER LIST ENTRY
         LA   RWORK,DS1FMTID       ADDRESS OF DSCB BUFFER
         ST   RWORK,BFLEBUF        PLACE IN BUFFER LIST
         OI   BFLEFL,BFLECHR       CCHHR OF DSCB RETURNED BY CVAF
         MVI  BFLELTH,DSCBLTH      DATA PORTION OF DSCB READ - DSN     *
                                   SUPPLIED IN CVPL
         MVC  DS1DSNAM,0(RDSN)     MOVE IN DATA SET NAME TO WORKAREA
         CVAFDIR ACCESS=READ,DSN=DS1DSNAM,BUFLIST=BUFLIST,DEB=(RDEB)
```

```
+         CNOP  0,4
+         BAL   1,ICV1E                  LOAD CVPL LIST ADDRESS
+ICV1S    EQU   *                        START OF CVPL
+         DC    CL4'CVPL'                EBCDIC 'CVPL'
+         DC    AL2(ICV1E-ICV1S)         LENGTH OF CVPL
+         DC    XL1'01'                  FUNCTION CODE
+         DC    XL1'00'                  STATUS INFORMATION
+         DC    B'00000000'              FIRST FLAG BYTE
+         DC    B'00000000'              SECOND FLAG BYTE
+         DC    H'0'                     RESERVED
+         DC    A(0)                     UCB ADDRESS
+         DC    A(DS1DSNAM)              DATA SET NAME ADDRESS
+         DC    A(BUFLIST)               BUFFER LIST ADDRESS
+         DC    A(0)                     INDEX VIR'S BUFFER LIST ADDRESS
+         DC    A(0)                     MAP VIR'S BUFFER LIST ADDRESS
+         DC    A(0)                     I/O AREA ADDRESS
+         DC    A(0)                     DEB ADDRESS
+         DC    A(0)                     ARGUMENT ADDRESS
+         DC    A(0)                     SPACE PARAMETER LIST ADDRESS
+         DC    A(0)                     EXTENT TABLE ADDRESS
+         DC    A(0)                     NEW VRF VIXM BUFFER LIST ADDR
+         DC    A(0)                     VRF DATA ADDRESS
+         DC    A(0)                     COUNT AREA ADDRESS
+ICV1E    EQU   *                        END OF CVPL
+         ST    RDEB,36(,1)              STORE DEB PTR IN PARM LIST
+         SVC   139
          USING CVPL,REG1           ADDRESSABILITY TO CVPL
          LTR   REG15,REG15         ANY ERROR
          BZ    NOERROR             BRANCH IF NOT
 *************************************************************
 *
 *        DETERMINE WHAT ERROR IS
 *
 *************************************************************
          C     REG15,ERROR4        IS RETURN CODE 4
          BNE   OTHERERR            BRANCH IF NOT 4
          CLI   CVSTAT,STAT001      IS IT DATA SET NAME NOT FOUND?
          BNE   OTHERERR            BRANCH IF NOT
          DROP  REG1                ADDRESSABILITY TO CVPL NOT NEEDED
 *************************************************************
 *
 *        DATA SET NAME NOT FOUND
 *
 *************************************************************
          L     RSAVE,4(,RSAVE)
          RETURN (14,12),RC=NOTFND  SET UP DATA SET NOT FOUND ERROR
+         LM    14,12,12(13)                      RESTORE THE REGISTERS
+         LA    15,NOTFND(0,0)                    LOAD RETURN CODE
+         BR    14                                RETURN
 NOERROR  EQU   *                   DSCB READ
          MVC   F1CCHHR,BFLECHR     MOVE CCHHR OF FORMAT 1/4 DSCB TO    *
                                    WORKAREA
          CLI   DS1FMTID,C'4'       IS DSCB A FORMAT 4 DSCB
          BE    NOTF1               BRANCH IF YES. NOT A FORMAT 1
          TM    DS1DSORG,DS1DSGPO   IS FORMAT 1 DSCB FOR PARTITIONED    *
                                    DATA SET
          BO    PDS                 BRANCH IF PDS
 NOTF1    EQU   *                   DSCB IS NOT A PDS
          L     RSAVE,4(,RSAVE)
          RETURN (14,12),RC=NOTPDS  SET UP NOT PDS RETURN CODE
+         LM    14,12,12(13)                      RESTORE THE REGISTERS
+         LA    15,NOTPDS(0,0)                    LOAD RETURN CODE
+         BR    14                                RETURN
 PDS      EQU   *                   DATA SET IS PARTITIONED
          L     RSAVE,4(,RSAVE)
          RETURN (14,12),RC=PDSRTN  SET UP PDS RETURN CODE
+         LM    14,12,12(13)                      RESTORE THE REGISTERS
+         LA    15,PDSRTN(0,0)                    LOAD RETURN CODE
+         BR    14                                RETURN
 OTHERERR EQU   *                   UNEXPECTED ERROR
          L     RSAVE,4(,RSAVE)
          RETURN (14,12),RC=UNEXPECD
+         LM    14,12,12(13)                      RESTORE THE REGISTERS
+         LA    15,UNEXPECD(0,0)                  LOAD RETURN CODE
+         BR    14                                RETURN
 ERROR4   DC   F'4'                  ERROR RETURN CODE 4
 BUFLIST  ICVAFBFL DSECT=NO          BUFFER LIST
```

```
+***********************************************************************
+*        BUFFER LIST HEADER
+***********************************************************************

+BUFLIST  DS    0F                      BUFFER LIST HEADER
+BFLHNOE  DS    XL1                     NUMBER OF ENTRIES
+BFLHFL   DS    XL1                     KEY AND FLAG BYTE
+         ORG   BFLHFL
+BFLHKEY  DS    XL1                     PROTECT KEY (FIRST 4 BITS)
+BFLHVIR  EQU   X'08'                   BUF. LIST ENTRIES DESCRIBE VIRS
+BFLHDSCB EQU   X'04'                   BUF. LIST ENTRIES DESCRIBE DSCBS
+         DS    XL1                     RESERVED
+BFLHSP   DS    XL1                     SUBPOOL OF BUF. LIST/BUFFERS
+BFLHFCHN DS    A                       FORWARD CHAIN PTR TO NEXT BUF.
+*                                      LIST
+BFLHLN   EQU   *-BUFLIST               LENGTH OF BUFFER LIST HEADER

+***********************************************************************
+*        BUFFER LIST ENTRY
+***********************************************************************

+BFLE     DS    0F                      BUFFER LIST ENTRY
+BFLEFL   DS    XL1                     BUFFER LIST ENTRY FLAG
+BFLERBA  EQU   X'80'                   ARGUMENT IS RBA
+BFLECHR  EQU   X'40'                   ARGUMENT IS CCHHR
+BFLETTR  EQU   X'20'                   ARGUMENT IS TTR
+BFLEAUPD EQU   X'10'                   CVAF UPDATED ARGUMENT FIELD
+BFLEMOD  EQU   X'08'                   DATA IN BUF. HAS BEEN MODIFIED
+BFLESKIP EQU   X'04'                   SKIP THIS ENTRY
+BFLEIOER EQU   X'02'                   I/O ERROR
+         DS    XL1                     RESERVED
+BFLELTH  DS    XL1                     LENGTH OF DSCB BUFFER OR
+*                                      LENGTH OF VIR DIVIDED BY 256
+BFLEARG  DS    XL5                     ARGUMENT OF VIR OR DSCB (CCHHR)
+         ORG   BFLEARG+1
+BFLEATTR DS    XL3                     'TTR' OF ARGUMENT
+         ORG   BFLEARG+1
+BFLEARBA DS    XL4                     'RBA'  OF ARGUMENT
+BFLEBUF  DS    A                       BUFFER ADDRESS
+BFLELN   EQU   *-BFLE                  LENGTH OF A BUFFER LIST ENTRY
          IECSDSL1 (1)              FORMAT 1 DSCB DATASET NAME AND      *
                                    BUFFER
+IECSDSL1 EQU   *                   FORMAT 1 DSCB
+IECSDSF1 EQU   IECSDSL1
+DS1DSNAM DS    CL44                DATA SET NAME
+DS1FMTID DS    CL1                 FORMAT IDENTIFIER
+DS1DSSN  DS    CL6                 DATA SET SERIAL NUMBER
+DS1VOLSQ DS    XL2                 VOLUME SEQUENCE NUMBER
+DS1CREDT DS    XL3                 CREATION DATE
+DS1EXPDT DS    XL3                 EXPIRATION DATE
+DS1NOEPV DS    XL1                 NUMBER OF EXTENTS ON VOLUME
+DS1NOBDB DS    XL1                 NUMBER OF BYTES USED IN LAST
+*                                      DIRECTORY BLOCK
+         DS    XL1                 RESERVED
+DS1SYSCD DS    CL13                SYSTEM CODE
+         DS    XL7                 RESERVED
+DS1DSORG DS    XL2                 DATA SET ORGANIZATION
+*                     FIRST BYTE OF DS1DSORG
+DS1DSGIS EQU   X'80'               IS - INDEXED SEQUENTIAL         @01A
+*                                  ORGANIZATION
+DS1DSGPS EQU   X'40'               PS - PHYSICAL SEQUENTIAL        @01A
+*                                  ORGANIZATION
+DS1DSGDA EQU   X'20'               DA - DIRECT ORGANIZATION        @01A
+DS1DSGCX EQU   X'10'               CX - BTAM OR QTAM LINE GROUP    @01A
+*        EQU   X'08'               RESERVED                        @01A
+*        EQU   X'04'               RESERVED                        @01A
+DS1DSGPO EQU   X'02'               PO - PARTITIONED ORGANIZATION   @01A
+DS1DSGU  EQU   X'01'               U - UNMOVABLE, THE DATA         @01A
+*                                  CONTAINS LOCATION DEPENDENT
+*                                  INFORMATION
+*
```

```
+*                     SECOND BYTE OF DS1DSORG
+DS1DSGGS EQU   X'80'               GS - GRAPHICS ORGANIZATION      @01A
+DS1DSGTX EQU   X'40'               TX - TCAM LINE GROUP            @01A
+DS1DSGTQ EQU   X'20'               TQ - TCAM MESSAGE QUEUE         @01A
+*        EQU   X'10'               RESERVED                        @01A
+DS1ACBM  EQU   X'08'               ACCESS METHOD CONTROL BLOCK     @01A
+DS1DSGTR EQU   X'04'               TR - TCAM 3705                  @01A
+*        EQU   X'02'               RESERVED                        @01A
+*        EQU   X'01'               RESERVED                        @01A
+DS1RECFM DS    XL1                 RECORD FORMAT
+DS1OPTCD DS    XL1                 OPTION CODE
+DS1BLKL  DS    XL2                 BLOCK LENGTH
+DS1LRECL DS    XL2                 RECORD LENGTH
+DS1KEYL  DS    XL1                 KEY LENGTH
+DS1RKP   DS    XL2                 RELATIVE KEY POSITION
+DS1DSIND DS    XL1                 DATA SET INDICATORS
+DS1SCALO DS    XL4                 SECONDARY ALLOCATION
+DS1LSTAR DS    XL3                 LAST USED TRACK AND BLOCK ON TRACK
+DS1TRBAL DS    XL2                 BYTES REMAINING ON LAST TRACK USED
+         DS    XL2                 RESERVED
+DS1EXT1  DS    XL10                FIRST EXTENT DESCRIPTION
+*        FIRST BYTE                EXTENT TYPE INDICATOR
+*        SECOND BYTE               EXTENT SEQUENCE NUMBER
+*        THIRD - SIXTH BYTES       LOWER LIMIT
+*        SEVENTH - TENTH BYTES     UPPER LIMIT
+DS1EXT2  DS    XL10                SECOND EXTENT DESCRIPTION
+DS1EXT3  DS    XL10                THIRD EXTENT DESCRIPTION
+DS1PTRDS DS    XL5                 POSSIBLE PTR TO A FORMAT 2 OR 3 DSCB
+DS1END   EQU   *
 DSCBLTH  EQU   *-IECSDSL1-L'DS1DSNAM LENGTH OF DATA PORTION OF DSCB
 F1CCHHR  DS   XL5                  CCHHR OF DSCB
 SAVEAREA DS   18F                  SAVE AREA
 CVPL     ICVAFPL ,                 CVPL MAPPING MACRO

+************************************************************************
+*        CVAF PARAMETER LIST
+************************************************************************

+CVPL     DSECT                         CVAF PARAMETER LIST
+         DS    0F
+CVLBL    DS    CL4                     EBCDIC 'CVPL'
+CVLTH    DS    H                       LENGTH OF CVPL
+CVFCTN   DS    XL1                     FUNCTION BYTE
+CVDIRD   EQU   X'01'                   CVAFDIR ACCESS=READ
+CVDIWR   EQU   X'02'                   CVAFDIR ACCESS=WRITE
+CVDIRLS  EQU   X'03'                   CVAFDIR ACCESS=RLSE
+CVSEQGT  EQU   X'04'                   CVAFSEQ ACCESS=GT
+CVSEQGTE EQU   X'05'                   CVAFSEQ ACCESS=GTEQ
+CVDMIXA  EQU   X'06'                   CVAFDSM ACCESS=IXADD
+CVDMIXD  EQU   X'07'                   CVAFDSM ACCESS=IXDLT
+CVDMALC  EQU   X'08'                   CVAFDSM ACCESS=ALLOC
+CVDMRLS  EQU   X'09'                   CVAFDSM ACCESS=RLSE
+CVDMMAP  EQU   X'0A'                   CVAFDSM ACCESS=MAPDATA
+CVVOL    EQU   X'0B'                   CVAFVOL ACCESS=VIBBLD
+CVVRFRD  EQU   X'0C'                   CVAFVRF ACCESS=READ
+CVVRFWR  EQU   X'0D'                   CVAFVRF ACCESS=WRITE
+CVSTAT   DS    XL1                     STATUS INFORMATION (SEE LIST    *
+                                       BELOW)
+CVFL1    DS    XL1                     FIRST FLAG BYTE
+CV1IVT   EQU   X'80'                   INDEXED VTOC ACCESSED
+CV1IOAR  EQU   X'40'                   IOAREA=KEEP
+CV1PGM   EQU   X'20'                   BRANCH=(YES,PGM)
+CV1MRCDS EQU   X'10'                   MAPRCDS=YES
+CV1IRCDS EQU   X'08'                   IXRCDS=KEEP
+CV1MAPIX EQU   X'04'                   MAP=INDEX
+CV1MAPVT EQU   X'02'                   MAP=VTOC
+CV1MAPVL EQU   X'01'                   MAP=VOLUME
+CVFL2    DS    XL1                     SECOND FLAG BYTE
+CV2HIVIE EQU   X'80'                   HIVIER=YES
+CV2VRF   EQU   X'40'                   VRF DATA EXISTS
+CV2CNT   EQU   X'20'                   COUNT=YES
+CV2RCVR  EQU   X'10'                   RECOVER=YES
```

```
+CV2SRCH  EQU   X'08'                   SEARCH=YES
+CV2DSNLY EQU   X'04'                   DSNONLY=YES
+CV2VER   EQU   X'02'                   VERIFY=YES
+CV2NLEVL EQU   X'01'                   OUTPUT-NEW HIGHEST LEVEL VIER
+*                                      CREATED
+         DS    H                       RESERVED
+CVUCB    DS    A                       UCB ADDRESS
+CVDSN    DS    A                       DATA SET NAME ADDRESS
+CVBUFL   DS    A                       BUFFER LIST ADDRESS
+CVIRCDS  DS    A                       INDEX VIR'S BUFFER LIST ADDRESS
+CVMRCDS  DS    A                       MAP VIR'S BUFFER LIST ADDRESS
+CVIOAR   DS    A                       I/O AREA ADDRESS
+CVDEB    DS    A                       DEB ADDRESS
+CVARG    DS    A                       ARGUMENT ADDRESS
+CVSPACE  DS    A                       SPACE PARAMETER LIST ADDRESS
+CVEXTS   DS    A                       EXTENT TABLE ADDRESS
+CVBUFL2  DS    A                       NEW VRF VIXM BUFFER LIST ADDR
+CVVRFDA  DS    A                       VRF DATA ADDRESS
+CVCTAR   DS    A                       COUNT AREA ADDRESS
+CVPLNGTH EQU   *-CVPL

+*              VALUES OF CVSTAT
+*(THIS PART OF THE ICVAFPL MACRO EXPANSION IS NOT SHOWN)
          END
```

EXAMPLE 2: USING THE CVAFDIR MACRO WITH AN INDEXED VTOC

This example uses the CVAFDIR macro to read one or more DSCBs on a VTOC. The UCB is supplied to the program in register 4 (labeled RUCB). The TTR of each DSCB read is to be returned to the caller. This program must be APF authorized.

The address of a parameter list is supplied to the program in register 5 (labeled RLIST). The parameter list contains one or more 3-word entries. The format of each 3-word entry is mapped by the LISTMAP DSECT. The first word contains the address of the data set name of the DSCB to be read. The second word contains the address of the 96-byte buffer into which the DSCB is to be read. The third word contains the address of the 3-byte TTR of the DSCB read.

The CVPL is generated by a list form of the CVAFDIR macro at label CVPL. The BUFLIST, IXRCDS, IOAREA, and BRANCH keywords are coded on the list form of the macro. IXRCDS=KEEP and IOAREA=KEEP are coded to avoid overhead if two or more DSCBs are to be read. BRANCH=(YES,PGM) is coded in the list form of the CVAFDIR macro to cause the CVPL to have the CV1PGM bit set to one; this will indicate to CVAF that the caller is authorized by APF and not in supervisor state. The execute forms of the CVAFDIR macro then specify BRANCH=YES, and not BRANCH=(YES,PGM), because the CV1PGM bit is set in the list form of the macro.

The CVAFDIR macro with ACCESS=RLSE is coded before the program exits in order to release the CVAF I/O area and the index records buffer list. BUFLIST=0 is coded because no user-supplied buffer list is to be released; BUFLIST was coded on the list form of the CVAFDIR macro and, therefore, is in the CVBUFL field of the CVPL. This field must be set to zero for the release.

```
EXAMPLE2 CSECT
         STM   14,12,12(13)
         BALR  12,0
         USING *,12
         ST    13,SAVEAREA+4
         LA    RWORK,SAVEAREA
         ST    RWORK,8(,13)
         LR    13,RWORK
*************************************************************
*
*        REGISTERS
*
*************************************************************
RWORK    EQU   3                  WORK REGISTER
RUCB     EQU   4                  UCB ADDRESS SUPPLIED BY CALLER
RLIST    EQU   5                  ADDRESS OF PARAMETER LIST
RDSN     EQU   6                  ADDRESS OF DATA SET NAME
RTTR     EQU   7                  ADDRESS OF TTR
REG15    EQU   15                 RETURN CODE REGISTER 15
*************************************************************
*
*        READ DSCB OF DATA SET NAME SUPPLIED. RETURN TTR OF DSCB.
*        UCB ADDRESS SUPPLIED IN RUCB.
*        ADDRESS OF PARAMETER LIST IN RLIST.
*          WORD 1 OF PARAMETER LIST = ADDRESS OF DATA SET NAME
*          WORD 2 OF PARAMETER LIST = ADDRESS OF DSCB TO BE RETURNED
*          WORD 3 OF PARAMETER LIST = ADDRESS OF TTR TO BE RETURNED
*                    WORDS 1-3 DUPLICATED WITH THE HIGH ORDER BIT OF
*                    WORD 3 SET TO ONE FOR LAST ENTRY.
*
*************************************************************
         USING LISTMAP,RLIST       ADDRESSABILITY TO PARMLIST
TOPLOOP  EQU  *                    LOOP FOR EACH DSCB
         XC   BUFLIST(BFLHLN+BFLELN),BUFLIST ZERO BUFFER LIST
         OI   BFLHFL,BFLHDSCB      DSCBS TO BE READ WITH BUFFER LIST
         MVI  BFLHNOE,1            ONE BUFFER LIST ENTRY
         LA   RWORK,LISTDSCB       ADDRESS OF DSCB BUFFER
         ST   RWORK,BFLEBUF        PLACE IN BUFFER LIST
         OI   BFLEFL,BFLETTR       TTR OF DSCB RETURNED BY CVAF
         MVI  BFLELTH,DSCBLTH      DATA PORTION OF DSCB READ - DSN     *
                                   SUPPLIED IN CVPL
         L    RDSN,LISTDSN         ADDRESS OF DATA SET NAME
         CVAFDIR DSN=(RDSN),UCB=(RUCB),MF=(E,CVPL),BRANCH=YES
+        LA    1,CVPL                                   LOAD PARAMETER REG 1
+        ST    RUCB,12(,1)              STORE UCB PTR IN PARM LIST
+        ST    RDSN,16(,1)              STORE DSN PTR IN PARM LIST
+        L     15,16                    LOAD THE CVT
+        L     15,328(,15)              LOAD VS1/VS2 COMMON EXTENSION2
+        L     15,12(,15)               LOAD THE CVT CVAF TABLE
+        L     15,0(,15)                LOAD THE CVAF ADDRESS
+        BALR  14,15                    BRANCH AND LINK TO CVAF
         L    RTTR,LISTTTR         ADDRESS OF TTR TO BE RETURNED
         USING TTRMAP,RTTR         MAP OF TTR
         LTR  REG15,REG15          ANY ERROR
         BZ   NOERROR              BRANCH IF NOT
         XC   TTR,TTR              ZERO TTR INDICATING NO DSCB
         B    RELOOP               GET NEXT ENTRY
NOERROR  EQU  *                    DSCB READ
         MVC  TTR,BFLEATTR         RETURN TTR OF DSCB
RELOOP   EQU  *                    GET NEXT ENTRY
         TM   LASTLIST,LASTBIT     IS IT LAST ENTRY IN LIST?
         LA   RLIST,NEXTLIST       GET NEXT ENTRY
         BZ   TOPLOOP              PROCESS NEXT LIST
         CVAFDIR ACCESS=RLSE,      RELEASE CVAF OBTAINED AREAS         *
               IOAREA=NOKEEP,      RELEASE IOAREA                      *
               IXRCDS=NOKEEP,      RELEASE VIER BUFFER LIST            *
               BUFLIST=0,          NO USER BUFFER LIST SUPPLIED TO RLSE*
               BRANCH=YES,         BRANCH ENTER CVAF                   *
               MF=(E,CVPL)
+        LA    1,CVPL                                   LOAD PARAMETER REG 1
+        MVI   6(1),X'03'               SET FUNCTION CODE
+        NI    8(1),B'10110111'         RESET CVAF FLAGS OFF
+        LA    15,0                     GET BUFLIST ADDRESS AND
+         ST    15,20(,1)                STORE BUFLIST PTR IN PARM LIST
+         L     15,16                    LOAD THE CVT
+         L     15,328(,15)              LOAD VS1/VS2 COMMON EXTENSION2
+         L     15,12(,15)               LOAD THE CVT CVAF TABLE
+         L     15,0(,15)                LOAD THE CVAF ADDRESS
+         BALR  14,15                    BRANCH AND LINK TO CVAF
          L     13,SAVEAREA+4
          RETURN (14,12)
+         LM    14,12,12(13)                      RESTORE THE REGISTERS
+         BR    14                                RETURN

 BUFLIST  ICVAFBFL DSECT=NO         BUFFER LIST

+***********************************************************************
+*        BUFFER LIST HEADER
+***********************************************************************

+BUFLIST  DS    0F                      BUFFER LIST HEADER
+BFLHNOE  DS    XL1                     NUMBER OF ENTRIES
+BFLHFL   DS    XL1                     KEY AND FLAG BYTE
+         ORG   BFLHFL
+BFLHKEY  DS    XL1                     PROTECT KEY (FIRST 4 BITS)
+BFLHVIR  EQU   X'08'                   BUF. LIST ENTRIES DESCRIBE VIRS
+BFLHDSCB EQU   X'04'                   BUF. LIST ENTRIES DESCRIBE DSCBS
+         DS    XL1                     RESERVED
+BFLHSP   DS    XL1                     SUBPOOL OF BUF. LIST/BUFFERS
+BFLHFCHN DS    A                       FORWARD CHAIN PTR TO NEXT BUF.
+*                                      LIST
+BFLHLN   EQU   *-BUFLIST               LENGTH OF BUFFER LIST HEADER

+***********************************************************************
+*        BUFFER LIST ENTRY
+***********************************************************************

+BFLE     DS    0F                      BUFFER LIST ENTRY
+BFLEFL   DS    XL1                     BUFFER LIST ENTRY FLAG
+BFLERBA  EQU   X'80'                   ARGUMENT IS RBA
+BFLECHR  EQU   X'40'                   ARGUMENT IS CCHHR
+BFLETTR  EQU   X'20'                   ARGUMENT IS TTR
+BFLEAUPD EQU   X'10'                   CVAF UPDATED ARGUMENT FIELD
+BFLEMOD  EQU   X'08'                   DATA IN BUF. HAS BEEN MODIFIED
+BFLESKIP EQU   X'04'                   SKIP THIS ENTRY
+BFLEIOER EQU   X'02'                   I/O ERROR
+         DS    XL1                     RESERVED
+BFLELTH  DS    XL1                     LENGTH OF DSCB BUFFER OR
+*                                      LENGTH OF VIR DIVIDED BY 256
+BFLEARG  DS    XL5                     ARGUMENT OF VIR OR DSCB (CCHHR)
+         ORG   BFLEARG+1
+BFLEATTR DS    XL3                     'TTR' OF ARGUMENT
+         ORG   BFLEARG+1
+BFLEARBA DS    XL4                     'RBA'  OF ARGUMENT
+BFLEBUF  DS    A                       BUFFER ADDRESS
+BFLELN   EQU   *-BFLE                  LENGTH OF A BUFFER LIST ENTRY
 SAVEAREA DS    18F                 REGISTER SAVE AREA
 LISTMAP  DSECT
 LISTDSN  DS    F                   ADDRESS OF DATA SET NAME
 LISTDSCB DS    F                   ADDRESS OF BUFFER FOR DSCB TO BE    *
                                    RETURNED
 LISTTTR  DS    0F                  ADDRESS OF TTR OF DSCB TO BE        *
                                    RETURNED
 LASTLIST DS    X                   FIRST BYTE
 LASTBIT  EQU   X'80'               LAST ENTRY IN LIST
          DS    XL3                 REMAINDER OF TTR ADDRESS
 NEXTLIST EQU   *                   NEXT LIST
 DSCB     DSECT
          IECSDSL1 (1)
+IECSDSL1 EQU   *                   FORMAT 1 DSCB
+IECSDSF1 EQU   IECSDSL1
+DS1DSNAM DS    CL44                DATA SET NAME
+DS1FMTID DS    CL1                 FORMAT IDENTIFIER
+DS1DSSN  DS    CL6                 DATA SET SERIAL NUMBER
```

```
+DS1VOLSQ DS    XL2                 VOLUME SEQUENCE NUMBER
+DS1CREDT DS    XL3                 CREATION DATE
+DS1EXPDT DS    XL3                 EXPIRATION DATE
+DS1NOEPV DS    XL1                 NUMBER OF EXTENTS ON VOLUME
+DS1NOBDB DS    XL1                 NUMBER OF BYTES USED IN LAST
+*                                     DIRECTORY BLOCK
+         DS    XL1                 RESERVED
+DS1SYSCD DS    CL13                SYSTEM CODE
+         DS    XL7                 RESERVED
+DS1DSORG DS    XL2                 DATA SET ORGANIZATION
+*                    FIRST BYTE OF DS1DSORG
+DS1DSGIS EQU   X'80'               IS - INDEXED SEQUENTIAL         @01A
+*                                  ORGANIZATION
+DS1DSGPS EQU   X'40'               PS - PHYSICAL SEQUENTIAL        @01A
+*                                  ORGANIZATION
+DS1DSGDA EQU   X'20'               DA - DIRECT ORGANIZATION        @01A
+DS1DSGCX EQU   X'10'               CX - BTAM OR QTAM LINE GROUP    @01A
+*        EQU   X'08'               RESERVED                        @01A
+*        EQU   X'04'               RESERVED                        @01A
+DS1DSGPO EQU   X'02'               PO - PARTITIONED ORGANIZATION   @01A
+DS1DSGU  EQU   X'01'               U - UNMOVABLE, THE DATA         @01A
+*                                  CONTAINS LOCATION DEPENDENT
+*                                  INFORMATION
+*
+*                     SECOND BYTE OF DS1DSORG
+DS1DSGGS EQU   X'80'               GS - GRAPHICS ORGANIZATION      @01A
+DS1DSGTX EQU   X'40'               TX - TCAM LINE GROUP            @01A
+DS1DSGTQ EQU   X'20'               TQ - TCAM MESSAGE QUEUE         @01A
+*        EQU   X'10'               RESERVED                        @01A
+DS1ACBM  EQU   X'08'               ACCESS METHOD CONTROL BLOCK     @01A
+DS1DSGTR EQU   X'04'               TR - TCAM 3705                  @01A
+*        EQU   X'02'               RESERVED                        @01A
+*        EQU   X'01'               RESERVED                        @01A
+DS1RECFM DS    XL1                 RECORD FORMAT
+DS1OPTCD DS    XL1                 OPTION CODE
+DS1BLKL  DS    XL2                 BLOCK LENGTH
+DS1LRECL DS    XL2                 RECORD LENGTH
+DS1KEYL  DS    XL1                 KEY LENGTH
+DS1RKP   DS    XL2                 RELATIVE KEY POSITION
+DS1DSIND DS    XL1                 DATA SET INDICATORS
+DS1SCALO DS    XL4                 SECONDARY ALLOCATION
+DS1LSTAR DS    XL3                 LAST USED TRACK AND BLOCK ON TRACK
+DS1TRBAL DS    XL2                 BYTES REMAINING ON LAST TRACK USED
+         DS    XL2                 RESERVED
+DS1EXT1  DS    XL10                FIRST EXTENT DESCRIPTION
+*        FIRST BYTE                EXTENT TYPE INDICATOR
+*        SECOND BYTE               EXTENT SEQUENCE NUMBER
+*        THIRD - SIXTH BYTES       LOWER LIMIT
+*        SEVENTH - TENTH BYTES     UPPER LIMIT
+DS1EXT2  DS    XL10                SECOND EXTENT DESCRIPTION
+DS1EXT3  DS    XL10                THIRD EXTENT DESCRIPTION
+DS1PTRDS DS    XL5                 POSSIBLE PTR TO A FORMAT 2 OR 3 DSCB
+DS1END   EQU   *
 DSCBLTH  EQU   *-DSCB-L'DS1DSNAM   LENGTH OF DATA PORTION OF DSCB
 TTRMAP   DSECT
 TTR      DS    XL3                 TTR TO BE RETURNED
 EXAMPLE2 CSECT
 CVPL     CVAFDIR ACCESS=READ,BUFLIST=BUFLIST,MF=L,                    *
                IOAREA=KEEP,        KEEP IOAREA TO AVOID OVERHEAD      *
                IXRCDS=KEEP     KEEP VIERS FOR 2ND AND SUBSEQUENT CALLS*
                                    CALLED IN PROGRAM STATE BUT APF    *
                                    AUTHORIZED SO UCB IS SUPPLIED
+         CNOP  0,4
+CVPL     EQU   *
+         DC    CL4'CVPL'               EBCDIC 'CVPL'
+         DC    AL2(ICV8E-CVPL)         LENGTH OF CVPL
+         DC    XL1'01'                 FUNCTION CODE
+         DC    XL1'00'                 STATUS INFORMATION
+         DC    B'01001000'             FIRST FLAG BYTE
+         DC    B'00000000'             SECOND FLAG BYTE
+         DC   H'0'                      RESERVED
+         DC   A(0)                      UCB ADDRESS
+         DC   A(0)                      DATA SET NAME ADDRESS
+         DC   A(BUFLIST)                BUFFER LIST ADDRESS
+         DC   A(0)                      INDEX VIR'S BUFFER LIST ADDRESS
+         DC   A(0)                      MAP VIR'S BUFFER LIST ADDRESS
+         DC   A(0)                      I/O AREA ADDRESS
+         DC   A(0)                      DEB ADDRESS
+         DC   A(0)                      ARGUMENT ADDRESS
+         DC   A(0)                      SPACE PARAMETER LIST ADDRESS
+         DC   A(0)                      EXTENT TABLE ADDRESS
+         DC   A(0)                      NEW VRF VIXM BUFFER LIST ADDR
+         DC   A(0)                      VRF DATA ADDRESS
+         DC   A(0)                      COUNT AREA ADDRESS
+ICV8E    EQU  *                         END OF CVPL
          ORG  CVPL                 OVERLAY CVPL WITH EXPANSION OF MAP
 CVPLMAP  ICVAFPL DSECT=NO

+***********************************************************************
+*        CVAF PARAMETER LIST
+***********************************************************************

+CVPLMAP  DS   0F                       CVAF PARAMETER LIST
+CVLBL    DS   CL4                      EBCDIC 'CVPL'
+CVLTH    DS   H                        LENGTH OF CVPL
+CVFCTN   DS   XL1                      FUNCTION BYTE
+CVDIRD   EQU  X'01'                    CVAFDIR ACCESS=READ
+CVDIWR   EQU  X'02'                    CVAFDIR ACCESS=WRITE
+CVDIRLS  EQU  X'03'                    CVAFDIR ACCESS=RLSE
+CVSEQGT  EQU  X'04'                    CVAFSEQ ACCESS=GT
+CVSEQGTE EQU  X'05'                    CVAFSEQ ACCESS=GTEQ
+CVDMIXA  EQU  X'06'                    CVAFDSM ACCESS=IXADD
+CVDMIXD  EQU  X'07'                    CVAFDSM ACCESS=IXDLT
+CVDMALC  EQU  X'08'                    CVAFDSM ACCESS=ALLOC
+CVDMRLS  EQU  X'09'                    CVAFDSM ACCESS=RLSE
+CVDMMAP  EQU  X'0A'                    CVAFDSM ACCESS=MAPDATA
+CVVOL    EQU  X'0B'                    CVAFVOL ACCESS=VIBBLD
+CVVRFRD  EQU  X'0C'                    CVAFVRF ACCESS=READ
+CVVRFWR  EQU  X'0D'                    CVAFVRF ACCESS=WRITE
+CVSTAT   DS   XL1                      STATUS INFORMATION (SEE LIST    *
+                                       BELOW)
+CVFL1    DS   XL1                      FIRST FLAG BYTE
+CV1IVT   EQU  X'80'                    INDEXED VTOC ACCESSED
+CV1IOAR  EQU  X'40'                    IOAREA=KEEP
+CV1PGM   EQU  X'20'                    BRANCH=(YES,PGM)
+CV1MRCDS EQU  X'10'                    MAPRCDS=YES
+CV1IRCDS EQU  X'08'                    IXRCDS=KEEP
+CV1MAPIX EQU  X'04'                    MAP=INDEX
+CV1MAPVT EQU  X'02'                    MAP=VTOC
+CV1MAPVL EQU  X'01'                    MAP=VOLUME
+CVFL2    DS   XL1                      SECOND FLAG BYTE
+CV2HIVIE EQU  X'80'                    HIVIER=YES
+CV2VRF   EQU  X'40'                    VRF DATA EXISTS
+CV2CNT   EQU  X'20'                    COUNT=YES
+CV2RCVR  EQU  X'10'                    RECOVER=YES
+CV2SRCH  EQU  X'08'                    SEARCH=YES
+CV2DSNLY EQU  X'04'                    DSNONLY=YES
+CV2VER   EQU  X'02'                    VERIFY=YES
+CV2NLEVL EQU  X'01'                    OUTPUT-NEW HIGHEST LEVEL VIER
+*                                      CREATED
+         DS   H                        RESERVED
+CVUCB    DS   A                        UCB ADDRESS
+CVDSN    DS   A                        DATA SET NAME ADDRESS
+CVBUFL   DS   A                        BUFFER LIST ADDRESS
+CVIRCDS  DS   A                        INDEX VIR'S BUFFER LIST ADDRESS
+CVMRCDS  DS   A                        MAP VIR'S BUFFER LIST ADDRESS
+CVIOAR   DS   A                        I/O AREA ADDRESS
+CVDEB    DS   A                        DEB ADDRESS
+CVARG    DS   A                        ARGUMENT ADDRESS
+CVSPACE  DS   A                        SPACE PARAMETER LIST ADDRESS
```

```
+CVEXTS   DS    A                       EXTENT TABLE ADDRESS
+CVBUFL2  DS    A                       NEW VRF VIXM BUFFER LIST ADDR
+CVVRFDA  DS    A                       VRF DATA ADDRESS
+CVCTAR   DS    A                       COUNT AREA ADDRESS
+CVPLNGTH EQU   *-CVPLMAP

+*              VALUES OF CVSTAT
+*(THIS PART OF THE ICVAFPL MACRO EXPANSION IS NOT SHOWN)
          END
```

EXAMPLE 3: USING THE CVAFSEQ MACRO WITH AN INDEXED VTOC

This example uses the CVAFSEQ to count the number of ISAM data sets whose data set names are within the range defined by two supplied data set names. The addresses of the two data set names are supplied to the program in registers 6 and 7, labeled RDSN1 and RDSN2, respectively. The address of a DEB open to the VTOC is supplied in register 4, labeled RDEB.

The CVAF parameter list is expanded by a list form of the CVAFSEQ macro. ACCESS=GTEQ is specified on the list form of the macro and is, therefore, not coded in the first execution of the CVPL. Subsequent executions of the CVPL (at label RELOOP) specify ACCESS=GT.

End of data is tested by comparing the CVSTAT field to the value STAT032, which is an equate in the ICVAFPL mapping macro.

The count of ISAM DSCBs matching the data set name criterion is returned in register 15 unless an error is encountered, in which case a negative one is returned in register 15.

```
 EXAMPLE3 CSECT
          STM   14,12,12(13)
          BALR  12,0
          USING *,12
          ST    13,SAVEAREA+4
          LA    RWORK,SAVEAREA
          ST    RWORK,8(,13)
          LR    13,RWORK
 ************************************************************
 *
 *        REGISTERS
 *
 ************************************************************
 REG1     EQU   1                   REGISTER 1
 RWORK    EQU   3                   WORK REGISTER
 RDEB     EQU   4                   DEB ADDRESS
 RDSN1    EQU   6                   ADDRESS OF DATA SET NAME 1
 RDSN2    EQU   7                   ADDRESS OF DATA SET NAME 2
 REG15    EQU   15                  RETURN CODE REGISTER 15
 ************************************************************
 *
 *        COUNT THE NUMBER OF ISAM DATA SETS WHOSE DATA SET NAMES ARE
 *           BETWEEN DSN1 AND DSN2 INCLUSIVELY.
 *           RDSN1 CONTAINS ADDRESS OF DSN1.
 *           RDSN2 CONTAINS ADDRESS OF DSN2.
 *        ADDRESS OF DEB OPEN TO VTOC SUPPLIED IN RDEB.
 *
 ************************************************************
          XC   BUFLIST(BFLHLN+BFLELN),BUFLIST ZERO BUFFER LIST
          OI   BFLHFL,BFLHDSCB       DSCBS TO BE READ WITH BUFFER LIST
          MVI  BFLHNOE,1             ONE BUFFER LIST ENTRY
          LA   RWORK,DS1FMTID        ADDRESS OF DSCB BUFFER
          ST   RWORK,BFLEBUF         PLACE IN BUFFER LIST

          MVI  BFLELTH,DSCBLTH       DATA PORTION OF DSCB READ - DSN     *
                                     SUPPLIED IN CVPL
          MVC  DS1DSNAM,0(RDSN1)    MOVE IN STARTING DATA SET NAME TO   *
                                    WORKAREA
          XR   RWORK,RWORK          ZERO COUNT
          CVAFSEQ DEB=(RDEB),       FIND FIRST DATA SET WHOSE DATA SET  *
                BUFLIST=BUFLIST,    NAME IS GREATER THAN OR EQUAL TO    *
                MF=(E,CVPL)         THAT OF DSN1
+         LA    1,CVPL                            LOAD PARAMETER REG 1
+         ST    RDEB,36(,1)              STORE DEB PTR IN PARM LIST
+         SVC   139
 LOOP     EQU  *                    LOOP UNTIL END OF DATA OR DATA SET  *
                                    NAME GREATER THAN DSN2
          USING CVPL,REG1           ADDRESSABILITY TO CVPL
          LTR  REG15,REG15          ANY ERROR
          BZ   TESTDSN              BRANCH IF NOT-CHECK DSN LIMIT
 ************************************************************
 *
 *        DETERMINE WHAT ERROR IS
 *
 ************************************************************
          C    REG15,ERROR4         IS RETURN CODE 4
          BNE  OTHERERR             BRANCH IF NOT 4
          CLI  CVSTAT,STAT032       IS IT END OF DATA?
          BNE  OTHERERR             BRANCH IF NOT
          DROP REG1                 ADDRESSABILITY TO CVPL NOT NEEDED
 ************************************************************
 *
 *        END OF DATA
 *
 ************************************************************
          B    RELEASE              RELEASE CVAF RESOURCES AND RETURN
 TESTDSN  EQU  *                    IS DATA SET NAME GREATER THAN DSN2
          CLI  DS1FMTID,C'1'        IS THIS A FORMAT 1 DSCB?
          BNE  CKLAST               BRANCH IF NO. CAN NOT BE ISAM.
          CLC  DS1DSNAM,0(RDSN2)    HAS LIMIT BEEN REACHED?
          BNH  TESTIS               BRANCH IF NO-TEST FOR ISAM
          B    RELEASE              RELEASE CVAF RESOURCES AND RETURN
 TESTIS   EQU  *                    ONLY COUNT ISAM
          TM   DS1DSORG,DS1DSGIS    IS DATA SET ISAM
          BZ   CKLAST               BRANCH IF NO-DO NOT COUNT IT
          LA   RWORK,1(,RWORK)      INCREMENT COUNT BY ONE
 CKLAST   EQU  *                    CHECK IF LAST DATA SET NAME (DSN2)
          CLC  DS1DSNAM,0(RDSN2)    HAS LIMIT BEEN REACHED?
          BNH  RELOOP               BRANCH IF NO-READ NEXT ONE
          B    RELEASE              RELEASE CVAF RESOURCES AND RETURN
 RELOOP   EQU  *                    READ NEXT DSCB
          CVAFSEQ ACCESS=GT,MF=(E,CVPL) GET DSCB WITH DATA SET NAME     *
                                    GREATER THAN THE ONE LAST READ
+         LA    1,CVPL                            LOAD PARAMETER REG 1
+         MVI   6(1),X'04'               SET FUNCTION CODE
+         SVC   139
          B    LOOP                 CHECK RESULTS OF CVAFSEQ
 OTHERERR EQU  *                    UNEXPECTED ERROR
 ************************************************************
 *
 *        UNEXPECTED ERROR PROCESSING
 *
 ************************************************************
          LA   RWORK,1(0,0)         ONE IN RWORK
          LNR  RWORK,RWORK          SET NEGATIVE COUNT INDICATING ERROR
 RELEASE  CVAFDIR ACCESS=RLSE,      RELEASE CVAF BUFFERS/IOAREA         *
                BUFLIST=0,          DO NOT RELEASE USER BUFFER LIST     *
                IXRCDS=NOKEEP,      RELEASE CVAF VIER BUFFERS           *
                MF=(E,CVPL)         RELEASE CVAF I/O AREA
+RELEASE  EQU   *
+         LA    1,CVPL                            LOAD PARAMETER REG 1
+         MVI   6(1),X'03'               SET FUNCTION CODE
+         NI    8(1),B'11110111'         RESET CVAF FLAGS OFF
+         LA    15,0                     GET BUFLIST ADDRESS AND
+         ST    15,20(,1)                STORE BUFLIST PTR IN PARM LIST
+         SVC   139
          LR   REG15,RWORK          CURRENT COUNT IS RETURN CODE
          L    13,SAVEAREA+4
          RETURN (14,12),RC=(15)    RETURN CURRENT COUNT
+         L     14,12(13,0)                       RESTORE REGISTER 14
+         LM    0,12,20(13)                       RESTORE THE REGISTERS
+         BR    14                                RETURN
 ERROR4   DC    F'4'                ERROR RETURN CODE 4
 BUFLIST  ICVAFBFL DSECT=NO         BUFFER LIST

+*************************************************************
+*        BUFFER LIST HEADER
+*************************************************************

+BUFLIST  DS    0F                      BUFFER LIST HEADER
+BFLHNOE  DS    XL1                     NUMBER OF ENTRIES
+BFLHFL   DS    XL1                     KEY AND FLAG BYTE
+         ORG   BFLHFL
+BFLHKEY  DS    XL1                     PROTECT KEY (FIRST 4 BITS)
+BFLHVIR  EQU   X'08'                   BUF. LIST ENTRIES DESCRIBE VIRS
+BFLHDSCB EQU   X'04'                   BUF. LIST ENTRIES DESCRIBE DSCBS
+         DS    XL1                     RESERVED
+BFLHSP   DS    XL1                     SUBPOOL OF BUF. LIST/BUFFERS
+BFLHFCHN DS    A                       FORWARD CHAIN PTR TO NEXT BUF.
+*                                      LIST
+BFLHLN   EQU   *-BUFLIST               LENGTH OF BUFFER LIST HEADER

+***********************************************************************
+*        BUFFER LIST ENTRY
+***********************************************************************

+BFLE     DS    0F                      BUFFER LIST ENTRY
+BFLEFL   DS    XL1                     BUFFER LIST ENTRY FLAG
+BFLERBA  EQU   X'80'                   ARGUMENT IS RBA
+BFLECHR  EQU   X'40'                   ARGUMENT IS CCHHR
+BFLETTR  EQU   X'20'                   ARGUMENT IS TTR
+BFLEAUPD EQU   X'10'                   CVAF UPDATED ARGUMENT FIELD
+BFLEMOD  EQU   X'08'                   DATA IN BUF. HAS BEEN MODIFIED
+BFLESKIP EQU   X'04'                   SKIP THIS ENTRY
+BFLEIOER EQU   X'02'                   I/O ERROR
+         DS    XL1                     RESERVED
+BFLELTH  DS    XL1                     LENGTH OF DSCB BUFFER OR
+*                                      LENGTH OF VIR DIVIDED BY 256
+BFLEARG  DS    XL5                     ARGUMENT OF VIR OR DSCB (CCHHR)
+         ORG   BFLEARG+1
+BFLEATTR DS    XL3                     'TTR' OF ARGUMENT
+         ORG   BFLEARG+1
+BFLEARBA DS    XL4                     'RBA'  OF ARGUMENT
+BFLEBUF  DS    A                       BUFFER ADDRESS
+BFLELN   EQU   *-BFLE                  LENGTH OF A BUFFER LIST ENTRY
          IECSDSL1 (1)              FORMAT 1 DSCB DATASET NAME AND      *
                                    BUFFER
+IECSDSL1 EQU   *                   FORMAT 1 DSCB
+IECSDSF1 EQU   IECSDSL1
+DS1DSNAM DS    CL44                DATA SET NAME
+DS1FMTID DS    CL1                 FORMAT IDENTIFIER
+DS1DSSN  DS    CL6                 DATA SET SERIAL NUMBER
+DS1VOLSQ DS    XL2                 VOLUME SEQUENCE NUMBER
+DS1CREDT DS    XL3                 CREATION DATE
+DS1EXPDT DS    XL3                 EXPIRATION DATE
+DS1NOEPV DS    XL1                 NUMBER OF EXTENTS ON VOLUME
+DS1NOBDB DS    XL1                 NUMBER OF BYTES USED IN LAST
+*                                      DIRECTORY BLOCK
+         DS    XL1                 RESERVED
+DS1SYSCD DS    CL13                SYSTEM CODE
+         DS    XL7                 RESERVED
+DS1DSORG DS    XL2                 DATA SET ORGANIZATION
+*                    FIRST BYTE OF DS1DSORG
+DS1DSGIS EQU   X'80'               IS - INDEXED SEQUENTIAL         @01A
+*                                  ORGANIZATION
+DS1DSGPS EQU   X'40'               PS - PHYSICAL SEQUENTIAL        @01A
+*                                  ORGANIZATION
+DS1DSGDA EQU   X'20'               DA - DIRECT ORGANIZATION        @01A
```

```
+DS1DSGCX EQU   X'10'                CX - BTAM OR QTAM LINE GROUP    @01A
+*        EQU   X'08'                RESERVED                        @01A
+*        EQU   X'04'                RESERVED                        @01A
+DS1DSGPO EQU   X'02'                PO - PARTITIONED ORGANIZATION   @01A
```

```
+DS1DSGU  EQU   X'01'               U - UNMOVABLE, THE DATA         @01A
+*                                  CONTAINS LOCATION DEPENDENT
+*                                  INFORMATION
+*
+*                     SECOND BYTE OF DS1DSORG
+DS1DSGGS EQU   X'80'               GS - GRAPHICS ORGANIZATION      @01A
+DS1DSGTX EQU   X'40'               TX - TCAM LINE GROUP            @01A
+DS1DSGTQ EQU   X'20'               TQ - TCAM MESSAGE QUEUE         @01A
+*        EQU   X'10'               RESERVED                        @01A
+DS1ACBM  EQU   X'08'               ACCESS METHOD CONTROL BLOCK     @01A
+DS1DSGTR EQU   X'04'               TR - TCAM 3705                  @01A
+*        EQU   X'02'               RESERVED                        @01A
+*        EQU   X'01'               RESERVED                        @01A
+DS1RECFM DS    XL1                 RECORD FORMAT
+DS1OPTCD DS    XL1                 OPTION CODE
+DS1BLKL  DS    XL2                 BLOCK LENGTH
+DS1LRECL DS    XL2                 RECORD LENGTH
+DS1KEYL  DS    XL1                 KEY LENGTH
+DS1RKP   DS    XL2                 RELATIVE KEY POSITION
+DS1DSIND DS    XL1                 DATA SET INDICATORS
+DS1SCALO DS    XL4                 SECONDARY ALLOCATION
+DS1LSTAR DS    XL3                 LAST USED TRACK AND BLOCK ON TRACK
+DS1TRBAL DS    XL2                 BYTES REMAINING ON LAST TRACK USED
+         DS    XL2                 RESERVED
+DS1EXT1  DS    XL10                FIRST EXTENT DESCRIPTION
+*        FIRST BYTE                EXTENT TYPE INDICATOR
+*        SECOND BYTE               EXTENT SEQUENCE NUMBER
+*        THIRD - SIXTH BYTES       LOWER LIMIT
+*        SEVENTH - TENTH BYTES     UPPER LIMIT
+DS1EXT2  DS    XL10                SECOND EXTENT DESCRIPTION
+DS1EXT3  DS    XL10                THIRD EXTENT DESCRIPTION
+DS1PTRDS DS    XL5                 POSSIBLE PTR TO A FORMAT 2 OR 3 DSCB
+DS1END   EQU   *
 DSCBLTH  EQU   *-IECSDSL1-L'DS1DSNAM LENGTH OF DATA PORTION OF DSCB
 SAVEAREA DS    18F                 SAVE AREA
 CVPL     CVAFSEQ ACCESS=GTEQ,      READ DSCB WITH DSN >= SUPPLIED DSN  *
                IXRCDS=KEEP,        KEEP VIERS IN STORAGE DURING CALLS  *
                DSN=DS1DSNAM,       SUPPLIED DATA SET NAME              *
                BUFLIST=BUFLIST,
                MF=L
+         CNOP  0,4
+CVPL     EQU   *
+         DC    CL4'CVPL'                EBCDIC 'CVPL'
+         DC    AL2(ICV10E-CVPL)         LENGTH OF CVPL
+         DC    XL1'05'                  FUNCTION CODE
+         DC    XL1'00'                  STATUS INFORMATION
+         DC    B'00001000'              FIRST FLAG BYTE
+         DC    B'00000000'              SECOND FLAG BYTE
+         DC    H'0'                     RESERVED
+         DC    A(0)                     UCB ADDRESS
+         DC    A(DS1DSNAM)              DATA SET NAME ADDRESS
+         DC    A(0)                     BUFFER LIST ADDRESS
+         DC    A(0)                     INDEX VIR'S BUFFER LIST ADDRESS
+         DC    A(0)                     MAP VIR'S BUFFER LIST ADDRESS
+         DC    A(0)                     I/O AREA ADDRESS
+         DC    A(0)                     DEB ADDRESS
+         DC    A(0)                     ARGUMENT ADDRESS
+         DC    A(0)                     SPACE PARAMETER LIST ADDRESS
+         DC    A(0)                     EXTENT TABLE ADDRESS
+         DC    A(0)                     NEW VRF VIXM BUFFER LIST ADDR
+         DC    A(0)                     VRF DATA ADDRESS
+         DC    A(0)                     COUNT AREA ADDRESS
+ICV10E   EQU   *                        END OF CVPL
          ORG   CVPL                EXPAND MAP OVER LIST
 CVPLMAP  ICVAFPL DSECT=NO          CVPL MAP

+***********************************************************************
+*        CVAF PARAMETER LIST
+***********************************************************************
```

```
+CVPLMAP  DS    0F                      CVAF PARAMETER LIST
+CVLBL    DS    CL4                     EBCDIC 'CVPL'
+CVLTH    DS    H                       LENGTH OF CVPL
+CVFCTN   DS    XL1                     FUNCTION BYTE
+CVDIRD   EQU   X'01'                   CVAFDIR ACCESS=READ
+CVDIWR   EQU   X'02'                   CVAFDIR ACCESS=WRITE
+CVDIRLS  EQU   X'03'                   CVAFDIR ACCESS=RLSE
+CVSEQGT  EQU   X'04'                   CVAFSEQ ACCESS=GT
+CVSEQGTE EQU   X'05'                   CVAFSEQ ACCESS=GTEQ
+CVDMIXA  EQU   X'06'                   CVAFDSM ACCESS=IXADD
+CVDMIXD  EQU   X'07'                   CVAFDSM ACCESS=IXDLT
+CVDMALC  EQU   X'08'                   CVAFDSM ACCESS=ALLOC
+CVDMRLS  EQU   X'09'                   CVAFDSM ACCESS=RLSE
+CVDMMAP  EQU   X'0A'                   CVAFDSM ACCESS=MAPDATA
+CVVOL    EQU   X'0B'                   CVAFVOL ACCESS=VIBBLD
+CVVRFRD  EQU   X'0C'                   CVAFVRF ACCESS=READ
+CVVRFWR  EQU   X'0D'                   CVAFVRF ACCESS=WRITE
+CVSTAT   DS    XL1                     STATUS INFORMATION (SEE LIST    *
+                                       BELOW)
+CVFL1    DS    XL1                     FIRST FLAG BYTE
+CV1IVT   EQU   X'80'                   INDEXED VTOC ACCESSED
+CV1IOAR  EQU   X'40'                   IOAREA=KEEP
+CV1PGM   EQU   X'20'                   BRANCH=(YES,PGM)
+CV1MRCDS EQU   X'10'                   MAPRCDS=YES
+CV1IRCDS EQU   X'08'                   IXRCDS=KEEP
+CV1MAPIX EQU   X'04'                   MAP=INDEX
+CV1MAPVT EQU   X'02'                   MAP=VTOC
+CV1MAPVL EQU   X'01'                   MAP=VOLUME
+CVFL2    DS    XL1                     SECOND FLAG BYTE
+CV2HIVIE EQU   X'80'                   HIVIER=YES
+CV2VRF   EQU   X'40'                   VRF DATA EXISTS
+CV2CNT   EQU   X'20'                   COUNT=YES
+CV2RCVR  EQU   X'10'                   RECOVER=YES
+CV2SRCH  EQU   X'08'                   SEARCH=YES
+CV2DSNLY EQU   X'04'                   DSNONLY=YES
+CV2VER   EQU   X'02'                   VERIFY=YES
+CV2NLEVL EQU   X'01'                   OUTPUT-NEW HIGHEST LEVEL VIER
+*                                      CREATED
+         DS    H                       RESERVED
+CVUCB    DS    A                       UCB ADDRESS
+CVDSN    DS    A                       DATA SET NAME ADDRESS
+CVBUFL   DS    A                       BUFFER LIST ADDRESS
+CVIRCDS  DS    A                       INDEX VIR'S BUFFER LIST ADDRESS
+CVMRCDS  DS    A                       MAP VIR'S BUFFER LIST ADDRESS
+CVIOAR   DS    A                       I/O AREA ADDRESS
+CVDEB    DS    A                       DEB ADDRESS
+CVARG    DS    A                       ARGUMENT ADDRESS
+CVSPACE  DS    A                       SPACE PARAMETER LIST ADDRESS
+CVEXTS   DS    A                       EXTENT TABLE ADDRESS
+CVBUFL2  DS    A                       NEW VRF VIXM BUFFER LIST ADDR
+CVVRFDA  DS    A                       VRF DATA ADDRESS
+CVCTAR   DS    A                       COUNT AREA ADDRESS
+CVPLNGTH EQU   *-CVPLMAP

+*              VALUES OF CVSTAT
+*(THIS PART OF THE ICVAFPL MACRO EXAPNSION IS NOT SHOWN)
          END
```

EXAMPLE 4: USING THE CVAFSEQ MACRO WITH A NONINDEXED VTOC

This example reads up to five DSCBs in physical-sequential order. The address of the UCB is supplied to the program in register 5 (labeled RUCB). The address of a parameter list is supplied in register 4 (labeled RLIST). The first word of the parameter list contains the address of a 5-byte field. On entry, this field is set to zero if no previous DSCBs have been read; otherwise, the field is set to the CCHHR of the last DSCB read. This 5-byte field is supplied by the caller of this program and is not modified by this program.

The remainder of the parameter list consists of one or more 2-word entries, up to a maximum of five 2-word entries. The first word of each entry contains the address of a 140-byte DSCB buffer. The second word contains the address of a 5-byte field that is to contain the CCHHR of the DSCB.

A buffer list with five buffer list entries is contained in the program. The ICVAFBFL macro generates the buffer list header and one buffer list entry. The remaining buffer list entries are generated following the ICVAFBFL macro.

The CVAFSEQ macro is used once in the program to read as many DSCBs as there are 2-word entries in the parameter list. The buffer list header field BFLHNOE is initialized with the number of buffer list entries that CVAFSEQ is to process. The number matches the number of 2-word entries in the parameter list supplied to this program.

After the CVAFSEQ call, the CCHHR for each DSCB read is moved from the buffer list entry field BFLEARG to the field whose address is supplied by the caller of the program. If the BFLEARG field is zero, the previous DSCB read was the last in the VTOC.

The BFLEARG in the first buffer list entry is initialized with the CCHHR supplied by the caller: its address is the third word in the parameter list. This CCHHR serves as the starting place for the CVAFSEQ call. DSCBs with a CCHHR greater than the supplied CCHHR are read.

This program must be APF authorized.

```
EXAMPLE4 CSECT
         STM   14,12,12(13)
         BALR  12,0
         USING *,12
         ST    13,SAVEAREA+4
         LA    RWORK,SAVEAREA
         ST    RWORK,8(,13)
         LR    13,RWORK
************************************************************
*
*        REGISTERS
*
************************************************************
REG1     EQU   1                  REGISTER 1
RWORK    EQU   3                  WORK REGISTER
RLIST    EQU   4                  ADDRESS OF PARM LIST
RUCB     EQU   5                  UCB ADDRESS
RCURRENT EQU   6                  CURRENT ENTRY IN PARM LIST
RBLE     EQU   7                  CURRENT BUFFER LIST ENTRY
RCOUNT   EQU   8                  COUNT OF ENTRIES IN BUFFER LIST
REG15    EQU   15                 RETURN CODE REGISTER 15
************************************************************
*
*        READ UP TO 5 DSCBS.
*        RUCB CONTAINS ADDRESS OF UCB.
*        RLIST CONTAINS ADDRESS OF PARAMETER LIST.
*          WORD 0 = ADDRESS OF CCHHR OF LAST DSCB READ. THIS DSCB IS
*                   NOT TO BE READ
*          WORD 1 = ADDRESS OF DSCB BUFFER.
*          WORD 2 = ADDRESS OF CCHHR OF DSCB READ.
*            WORD1 AND WORD2 REPEATED UP TO 4 TIMES.
*            HIGH ORDER BIT OF WORD 2 SET TO ONE FOR LAST ENTRY.
*
************************************************************
         USING LIST,RLIST          ADDRESSABILITY TO PARM LIST
         XC   BFLHDR(BFLHLN+5*BFLELN),BFLHDR ZERO BUFFER LIST WITH     *
                                   5 BUFFER LIST ENTRIES
```

```
          OI   BFLHFL,BFLHDSCB      DSCBS TO BE READ WITH BUFFER LIST
          LA   RCURRENT,LISTPRMS    FIRST DOUBLEWORD ENTRY IN PARM LIST
          USING LISTPRMS,RCURRENT   USING ON DOUBLEWORDS
          LA   RBLE,BFLE            FIRST BUFFER LIST ENTRY
          USING BFLE,RBLE
          L    RWORK,LISTSTRT       ADDRESS OF STARTING CCHHR
          MVC  BFLEARG,0(RWORK)     MOVE STARTING CCHHR INTO FIRST      *
                                    BUFFER LIST ENTRY
          XR   RCOUNT,RCOUNT        ZERO COUNT
 BUFLOOP  EQU  *                    PUT BUFFER ADDRESSES IN BUFFER LIST *
                                    ENTRIES
          LA   RCOUNT,1(,RCOUNT)    INCREMENT COUNT
          L    RWORK,LISTBUF        ADDRESS OF DSCB BUFFER
          ST   RWORK,BFLEBUF-BFLE(,RBLE) PLACE IN BUFFER LIST
          MVI  BFLELTH-BFLE(RBLE),DSCBLTH FULL DSCB READ
          TM   LISTLAST,LASTBIT     IS IT LAST ENTRY IN LIST
          LA   RCURRENT,LISTNEXT    INCREMENT TO NEXT ENTRY IN LIST
          LA   RBLE,BFLELN(,RBLE)   INCREMENT TO NEXT BUFFER LIST ENTRY
          BZ   BUFLOOP              LOOP TO PUT NEXT BUFFER IN BFLE
          STC  RCOUNT,BFLHNOE       SET NUMBER OF ENTRIES IN BUFFER     *
                                    LIST HEADER
          DROP RCURRENT,RBLE
 ************************************************************
 *
 *        READ UP TO 5 DSCBS WHOSE CCHHR IS GREATER THAN THE CCHHR IN
 *        THE FIRST BUFFER LIST ENTRY
 *
 ************************************************************
          CVAFSEQ UCB=(RUCB),       CALL CVAF                           *
                BRANCH=YES,         BRANCH ENTER                        *
                MF=(E,CVPL)
+         LA    1,CVPL                            LOAD PARAMETER REG 1

+         ST    RUCB,12(,1)              STORE UCB PTR IN PARM LIST
+         L     15,16                    LOAD THE CVT
+         L     15,328(,15)              LOAD VS1/VS2 COMMON EXTENSION2
+         L     15,12(,15)               LOAD THE CVAF TABLE ADDRESS
+         L     15,0(,15)                LOAD THE CVAF ADDRESS
+         BALR  14,15                    BRANCH AND LINK TO CVAF
          USING CVPL,REG1           ADDRESSABILITY TO CVPL
          LTR  REG15,REG15          ANY ERROR
          BZ   MOVECHR              BRANCH IF MOVE IN CCHHRS
 ************************************************************
 *
 *        DETERMINE WHAT ERROR IS
 *
 ************************************************************
          C    REG15,ERROR4         IS RETURN CODE 4
          BNE  OTHERERR             BRANCH IF NOT 4
          CLI  CVSTAT,STAT032       IS IT END OF DATA?
          BNE  OTHERERR             BRANCH IF NOT
          DROP REG1                 ADDRESSABILITY TO CVPL NOT NEEDED
 ************************************************************
 *
 *        DETERMINE IF ANY DSCBS HAVE BEEN READ. BFLEARG IS NON-ZERO
 *        IN EACH BUFFER LIST ENTRY FOR WHICH A DSCB HAS BEEN READ
 *
 ************************************************************
 MOVECHR  EQU  *                    IS DATA SET NAME GREATER THAN DSN2
          LA   RCURRENT,LISTPRMS    FIRST ENTRY IN PARM LIST
          USING LISTPRMS,RCURRENT
          LA   RBLE,BFLE            FIRST BUFFER LIST ENTRY
          USING BFLE,RBLE
 CHRLOOP  EQU  *                    MOVE CCHHR ARGUMENT TO CALLER AREA
          L    RWORK,LISTCHR        ADDRESS OF CCHHR OF CALLER
          XC   0(L'BFLEARG,RWORK),0(RWORK) ZERO CALLER CCHHR AREA
          NC   BFLEARG,BFLEARG      IS CCHHR ZERO
          BZ   EXIT                 BRANCH IF YES-NO MORE DSCBS
          MVC  0(L'BFLEARG,RWORK),BFLEARG MOVE CCHHR TO CALLER AREA
          TM   LISTLAST,LASTBIT     LAST ENTRY IN PARM LIST?
          BO   EXIT                 BRANCH IF YES
          LA   RCURRENT,LISTNEXT    NEXT ENTRY IN LIST
          LA   RBLE,BFLELN(,RBLE)   NEXT BUFFER LIST ENTRY
          B    CHRLOOP              TEST NEXT BFLE
 EXIT     EQU  *                    RETURN TO CALLER
          L    13,SAVEAREA+4
          RETURN (14,12)
+         LM    14,12,12(13)                      RESTORE THE REGISTERS
+         BR    14                                RETURN
```

```
 OTHERERR EQU  *                    ERROR PROCESSING
 *
 *
 *
          B    EXIT                 RETURN
 ERROR4   DC   F'4'                 RETURN CODE 4
          ICVAFBFL DSECT=NO         BUFFER LIST WITH ONE BUFFER LIST    *
                                    ENTRY

+***********************************************************************
+*        BUFFER LIST HEADER
+***********************************************************************

+BFLHDR   DS    0F                      BUFFER LIST HEADER
+BFLHNOE  DS    XL1                     NUMBER OF ENTRIES
+BFLHFL   DS    XL1                     KEY AND FLAG BYTE
+         ORG   BFLHFL
+BFLHKEY  DS    XL1                     PROTECT KEY (FIRST 4 BITS)
+BFLHVIR  EQU   X'08'                   BUF. LIST ENTRIES DESCRIBE VIRS
+BFLHDSCB EQU   X'04'                   BUF. LIST ENTRIES DESCRIBE DSCBS
+         DS    XL1                     RESERVED
+BFLHSP   DS    XL1                     SUBPOOL OF BUF. LIST/BUFFERS
+BFLHFCHN DS    A                       FORWARD CHAIN PTR TO NEXT BUF.
+*                                      LIST
+BFLHLN   EQU   *-BFLHDR                LENGTH OF BUFFER LIST HEADER

+***********************************************************************
+*        BUFFER LIST ENTRY
+***********************************************************************

+BFLE     DS    0F                      BUFFER LIST ENTRY
+BFLEFL   DS    XL1                     BUFFER LIST ENTRY FLAG
+BFLERBA  EQU   X'80'                   ARGUMENT IS RBA
+BFLECHR  EQU   X'40'                   ARGUMENT IS CCHHR
+BFLETTR  EQU   X'20'                   ARGUMENT IS TTR
+BFLEAUPD EQU   X'10'                   CVAF UPDATED ARGUMENT FIELD
+BFLEMOD  EQU   X'08'                   DATA IN BUF. HAS BEEN MODIFIED
+BFLESKIP EQU   X'04'                   SKIP THIS ENTRY
+BFLEIOER EQU   X'02'                   I/O ERROR
+         DS    XL1                     RESERVED
+BFLELTH  DS    XL1                     LENGTH OF DSCB BUFFER OR
+*                                      LENGTH OF VIR DIVIDED BY 256
+BFLEARG  DS    XL5                     ARGUMENT OF VIR OR DSCB (CCHHR)
+         ORG   BFLEARG+1
+BFLEATTR DS    XL3                     'TTR' OF ARGUMENT
+         ORG   BFLEARG+1
+BFLEARBA DS    XL4                     'RBA'  OF ARGUMENT
+BFLEBUF  DS    A                       BUFFER ADDRESS
+BFLELN   EQU   *-BFLE                  LENGTH OF A BUFFER LIST ENTRY
          DS   CL(4*BFLELN)         FOUR BUFFER LIST ENTRIES
 SAVEAREA DS   18F                  SAVE AREA
 DSCB     DSECT
          IECSDSL1 (1)              FORMAT 1 DSCB DATASET NAME AND      *
                                    DATA
+IECSDSL1 EQU   *                   FORMAT 1 DSCB
+IECSDSF1 EQU   IECSDSL1
+DS1DSNAM DS    CL44                DATA SET NAME
+DS1FMTID DS    CL1                 FORMAT IDENTIFIER
+DS1DSSN  DS    CL6                 DATA SET SERIAL NUMBER
+DS1VOLSQ DS    XL2                 VOLUME SEQUENCE NUMBER
+DS1CREDT DS    XL3                 CREATION DATE
+DS1EXPDT DS    XL3                 EXPIRATION DATE
+DS1NOEPV DS    XL1                 NUMBER OF EXTENTS ON VOLUME
+DS1NOBDB DS    XL1                 NUMBER OF BYTES USED IN LAST
+*                                      DIRECTORY BLOCK
+         DS    XL1                 RESERVED
+DS1SYSCD DS    CL13                SYSTEM CODE
+         DS    XL7                 RESERVED
+DS1DSORG DS    XL2                 DATA SET ORGANIZATION
```

```
+*                    FIRST BYTE OF DS1DSORG
+DS1DSGIS EQU   X'80'               IS - INDEXED SEQUENTIAL         @01A
+*                                  ORGANIZATION
+DS1DSGPS EQU   X'40'               PS - PHYSICAL SEQUENTIAL        @01A
+*                                  ORGANIZATION
+DS1DSGDA EQU   X'20'               DA - DIRECT ORGANIZATION        @01A
+DS1DSGCX EQU   X'10'               CX - BTAM OR QTAM LINE GROUP    @01A
+*        EQU   X'08'               RESERVED                        @01A
+*        EQU   X'04'               RESERVED                        @01A
+DS1DSGPO EQU   X'02'               PO - PARTITIONED ORGANIZATION   @01A
+DS1DSGU  EQU   X'01'               U - UNMOVABLE, THE DATA         @01A
+*                                  CONTAINS LOCATION DEPENDENT
+*                                  INFORMATION
+*
+*                     SECOND BYTE OF DS1DSORG
+DS1DSGGS EQU   X'80'               GS - GRAPHICS ORGANIZATION      @01A
+DS1DSGTX EQU   X'40'               TX - TCAM LINE GROUP            @01A
+DS1DSGTQ EQU   X'20'               TQ - TCAM MESSAGE QUEUE         @01A
+*        EQU   X'10'               RESERVED                        @01A
+DS1ACBM  EQU   X'08'               ACCESS METHOD CONTROL BLOCK     @01A
+DS1DSGTR EQU   X'04'               TR - TCAM 3705                  @01A
+*        EQU   X'02'               RESERVED                        @01A
+*        EQU   X'01'               RESERVED                        @01A
+DS1RECFM DS    XL1                 RECORD FORMAT
+DS1OPTCD DS    XL1                 OPTION CODE
+DS1BLKL  DS    XL2                 BLOCK LENGTH
+DS1LRECL DS    XL2                 RECORD LENGTH
+DS1KEYL  DS    XL1                 KEY LENGTH
+DS1RKP   DS    XL2                 RELATIVE KEY POSITION
+DS1DSIND DS    XL1                 DATA SET INDICATORS
+DS1SCALO DS    XL4                 SECONDARY ALLOCATION
+DS1LSTAR DS    XL3                 LAST USED TRACK AND BLOCK ON TRACK
+DS1TRBAL DS    XL2                 BYTES REMAINING ON LAST TRACK USED
+         DS    XL2                 RESERVED
+DS1EXT1  DS    XL10                FIRST EXTENT DESCRIPTION
+*        FIRST BYTE                EXTENT TYPE INDICATOR
+*        SECOND BYTE               EXTENT SEQUENCE NUMBER
+*        THIRD - SIXTH BYTES       LOWER LIMIT
+*        SEVENTH - TENTH BYTES     UPPER LIMIT
+DS1EXT2  DS    XL10                SECOND EXTENT DESCRIPTION
+DS1EXT3  DS    XL10                THIRD EXTENT DESCRIPTION
+DS1PTRDS DS    XL5                 POSSIBLE PTR TO A FORMAT 2 OR 3 DSCB
+DS1END   EQU   *
 DSCBLTH  EQU  *-IECSDSL1           LENGTH OF DSCB
 LIST     DSECT                     PARAMETER LIST
 LISTSTRT DS    F                   ADDRESS OF CCHHR TO START SEARCH
 LISTPRMS EQU   *
 LISTBUF  DS    F                   BUFFER ADDRESS
 LISTCHR  DS    0F                  ADDRESS OF CCHHR FIELD
 LISTLAST DS    X                   BYTE
 LASTBIT  EQU   X'80'               LAST DOUBLE WORD
          DS    AL3                 3 BYTE ADDRESS OF CCHHR
 LISTNEXT EQU   *                   NEXT DOUBLEWORD
 EXAMPLE4 CSECT
 ************************************************************
 *
 *        READ DSCBS WITH CCHHR GREATER THAN THE CCHHR IN THE FIRST
 *        BUFFER LIST ENTRY.
 *
 ************************************************************
 CVPL     CVAFSEQ ACCESS=GT,                                            *
                BUFLIST=BFLHDR,     ADDRESS OF BUFFER LIST              *
                MF=L
+         CNOP  0,4
+CVPL     EQU   *
+         DC    CL4'CVPL'                EBCDIC 'CVPL'
+         DC    AL2(ICV6E-CVPL)          LENGTH OF CVPL
+         DC    XL1'04'                  FUNCTION CODE
+         DC    XL1'00'                  STATUS INFORMATION
+         DC    B'00100000'              FIRST FLAG BYTE
+         DC    B'00000000'              SECOND FLAG BYTE
+         DC    A(0)                     RESERVED
```

```
+         DC    H'0'                     RESERVED
+         DC    A(0)                     UCB ADDRESS
+         DC    A(0)                     DATA SET NAME ADDRESS
+         DC    A(BFLHDR)                BUFFER LIST ADDRESS
+         DC    A(0)                     INDEX VIR'S BUFFER LIST ADDRESS
+         DC    A(0)                     MAP VIR'S BUFFER LIST ADDRESS
+         DC    A(0)                     I/O AREA ADDRESS
+         DC    A(0)                     DEB ADDRESS
+         DC    A(0)                     ARGUMENT ADDRESS
+         DC    A(0)                     SPACE PARAMETER LIST ADDRESS
+         DC    A(0)                     EXTENT TABLE ADDRESS
+         DC    A(0)                     NEW VRF VIXM BUFFER LIST ADDR
+         DC    A(0)                     VRF DATA ADDRESS
+         DC    A(0)                     COUNT AREA ADDRESS
+ICV6E    EQU   *                        END OF CVPL
          ORG   CVPL                EXPAND MAP OVER LIST
 CVPLMAP  ICVAFPL DSECT=NO          CVPL MAP

+***********************************************************************
+*        CVAF PARAMETER LIST
+***********************************************************************

+CVPLMAP  DS    0F                      CVAF PARAMETER LIST
+CVLBL    DS    CL4                     EBCDIC 'CVPL'
+CVLTH    DS    H                       LENGTH OF CVPL
+CVFCTN   DS    XL1                     FUNCTION BYTE
+CVDIRD   EQU   X'01'                   CVAFDIR ACCESS=READ
+CVDIWR   EQU   X'02'                   CVAFDIR ACCESS=WRITE
+CVDIRLS  EQU   X'03'                   CVAFDIR ACCESS=RLSE
+CVSEQGT  EQU   X'04'                   CVAFSEQ ACCESS=GT
+CVSEQGTE EQU   X'05'                   CVAFSEQ ACCESS=GTEQ
+CVDMIXA  EQU   X'06'                   CVAFDSM ACCESS=IXADD
+CVDMIXD  EQU   X'07'                   CVAFDSM ACCESS=IXDLT
+CVDMALC  EQU   X'08'                   CVAFDSM ACCESS=ALLOC
+CVDMRLS  EQU   X'09'                   CVAFDSM ACCESS=RLSE
+CVDMMAP  EQU   X'0A'                   CVAFDSM ACCESS=MAPDATA
+CVVOL    EQU   X'0B'                   CVAFVOL ACCESS=VIBBLD
+CVVRFRD  EQU   X'0C'                   CVAFVRF ACCESS=READ
+CVVRFWR  EQU   X'0D'                   CVAFVRF ACCESS=WRITE
+CVSTAT   DS    XL1                     STATUS INFORMATION (SEE LIST    *
+                                       BELOW)
+CVFL1    DS    XL1                     FIRST FLAG BYTE
+CV1IVT   EQU   X'80'                   INDEXED VTOC ACCESSED
+CV1IOAR  EQU   X'40'                   IOAREA=KEEP
+CV1PGM   EQU   X'20'                   BRANCH=(YES,PGM)
+CV1MRCDS EQU   X'10'                   MAPRCDS=YES
+CV1IRCDS EQU   X'08'                   IXRCDS=KEEP
+CV1MAPIX EQU   X'04'                   MAP=INDEX
+CV1MAPVT EQU   X'02'                   MAP=VTOC
+CV1MAPVL EQU   X'01'                   MAP=VOLUME
+CVFL2    DS    XL1                     SECOND FLAG BYTE
+CV2HIVIE EQU   X'80'                   HIVIER=YES
+CV2VRF   EQU   X'40'                   VRF DATA EXISTS
+CV2CNT   EQU   X'20'                   COUNT=YES
+CV2RCVR  EQU   X'10'                   RECOVER=YES
+CV2SRCH  EQU   X'08'                   SEARCH=YES
+CV2DSNLY EQU   X'04'                   DSNONLY=YES
+CV2VER   EQU   X'02'                   VERIFY=YES
+CV2NLEVL EQU   X'01'                   OUTPUT-NEW HIGHEST LEVEL VIER
+*                                      CREATED
+         DS    H                       RESERVED
+CVUCB    DS    A                       UCB ADDRESS
+CVDSN    DS    A                       DATA SET NAME ADDRESS
+CVBUFL   DS    A                       BUFFER LIST ADDRESS
+CVIRCDS  DS    A                       INDEX VIR'S BUFFER LIST ADDRESS
+CVMRCDS  DS    A                       MAP VIR'S BUFFER LIST ADDRESS
+CVIOAR   DS    A                       I/O AREA ADDRESS
+CVDEB    DS    A                       DEB ADDRESS
+CVARG    DS    A                       ARGUMENT ADDRESS
+CVSPACE  DS    A                       SPACE PARAMETER LIST ADDRESS
```

```
+CVEXTS   DS    A                          EXTENT TABLE ADDRESS
+CVBUFL2  DS    A                          NEW VRF VIXM BUFFER LIST ADDR
+CVVRFDA  DS    A                          VRF DATA ADDRESS
+CVCTAR   DS    A                          COUNT AREA ADDRESS
+CVPLNGTH EQU   *-CVPLMAP

+*              VALUES OF CVSTAT
+*(THIS PART OF THE ICVAFPL MACRO EXPANSION IS NOT SHOWN)
          END
```

EXAMPLE 5: USING THE CVAFTST AND CVAFDSM MACROS

This example returns a format-5 DSCB to the caller. The format-5 DSCB is constructed by this program if the volume contains an indexed VTOC. The format-5 DSCB is read by another program, F5RTN (not described in the example), if the volume contains a nonindexed VTOC.

The CVAFTST macro is used to determine if a nonindexed VTOC is on the volume.

If the CVAFTST return code is neither 0 nor 4 (a nonindexed VTOC is on the volume), the CVAFDSM macro is issued to obtain up to 27 extents from the VPSM in the VTOC index. The program does not determine whether the CVAFTST return code is 8 (volume contains indexed VTOC) or 12 (it cannot be determined what type of VTOC is on the volume). In either case, the CVAFDSM macro is issued. If the CVAFTST return code is 12, the CVAFDSM macro call will cause CVAF to determine whether an indexed or a nonindexed VTOC is on the volume, and the CV1IVT bit will be set to one or zero, accordingly.

The extent table (at label EXTABL) is initialized to request 27 extents from the CVAFDSM macro, which is one more than the number of extents that fit in a format-5 DSCB. The format-5 DSCB is constructed from the first 26 extents returned from the CVAFDSM call.

The first extent in the extent table is initialized from the last extent in the format-5 DSCB area supplied by the caller of the program. If this is the first call, the program assumes that the format-5 area is initialized to zero. Thus, the first extent in the extent table has a value of zero to serve as the starting place for the extent search. If this is the second or subsequent call, the last extent in the format-5 area would be the last extent obtained from the previous CVAFDSM call.

The format-5 chain pointer field (DS5PTRDS) is set to a nonzero value if CVAFDSM returned a 27th extent. In this case, the program will be called again to obtain another format-5 DSCB.

The program's return code is 0 if no errors were encountered and 4 if an error was encountered.

This program must be APF authorized.

```
 EXAMPLE5 CSECT
          STM   14,12,12(13)
          BALR  12,0
          USING *,12
          ST    13,SAVEAREA+4
          LA    RWORK,SAVEAREA
          ST    RWORK,8(,13)
          LR    13,RWORK
 ************************************************************
 *
 *        REGISTERS
 *
 ************************************************************
 RDEB     EQU   3                  DEB ADDRESS SUPPLIED BY CALLER
 RUCB     EQU   4                  UCB ADDRESS SUPPLIED BY CALLER
 RF5      EQU   5                  ADDRESS OF FORMAT 5 BUFFER SUPPLIED   *
                                   BY CALLER
 RWORK    EQU   6                  WORK REGISTER
 REG15    EQU   15                 RETURN CODE REGISTER 15
 *
 KF5      EQU   26                 NUMBER OF FORMAT 5 EXTENTS
 ************************************************************
 *
 *        READ FORMAT 5 DSCB OR BUILD A FORMAT 5 DSCB IF
 *          AN INDEXED VTOC
 *        UCB ADDRESS SUPPLIED IN RUCB.
 *        RF5 CONTAINS THE ADDRESS OF THE FORMAT 5 DSCB BUFFER. IT
 *          CONTAINS THE LAST FORMAT 5 DSCB READ OR BUILT. THE FORMAT 5
 *          BUFFER IS ZERO IF THIS IS THE FIRST CALL
 *        IF THE FORMAT 5 DSCB BUFFER RETURNED TO THE CALLER HAS A
 *          NONZERO VALUE IN DS5PTRDS, THIS ROUTINE WILL BE CALLED
 *          AGAIN TO OBTAIN THE NEXT FORMAT 5 DSCB.
 *
 ************************************************************
          USING IECSDSF5,RF5        ADDRESSABILITY TO FORMAT 5 BUFFER
          CVAFTST UCB=(RUCB)        TEST VTOC
+         CNOP  0,4                      START OF CVAFTST MACRO
+         LR    1,RUCB                            LOAD PARAMETER REG 1
+         L     15,16                    LOAD THE CVT
+         L     15,328(,15)              LOAD VS1/VS2 COMMON EXTENSION2
+         L     15,12(,15)               LOAD THE CVAF TABLE ADDRESS
+         LTR   15,15                    TEST FOR ZERO VALUE
+         BZ    ICV1E                    CVAF IS NOT ON THE SYSTEM
+         L     15,4(,15)                LOAD POINTER TO CVAF TEST E.P.
+         BALR  14,15                    BRANCH AND LINK TO CVAF TEST
+ICV1E    EQU   *                        END OF CVAFTST
          LTR  REG15,REG15
          BZ   UNINDXD              READ NEXT FORMAT 5
          C    REG15,NOTIXRC        UNINDEXED VTOC?
          BE   UNINDXD              READ NEXT FORMAT 5
 ************************************************************
 *
 *        ASSUME INDEXED VTOC UNLESS CVAFDSM CALL INDICATES UNINDEXED
 *
 ************************************************************
          MVC  EXTS(L'DS5AVEXT),DS5MAVET+L'DS5MAVET-L'DS5AVEXT MOVE THE *
                                    LAST EXTENT FROM FORMAT 5 TO FIRST   *
                                    ENTRY IN THE EXTENT TABLE
          CVAFDSM MF=(E,CVPL),      GET 27 EXTENTS FROM CVPL             *
                UCB=(RUCB),         RUCB ADDRESS REQUIRED                *
                DEB=(RDEB),         RDEB ADDRESS REQUIRED BY             *
                                     UNAUTHORIZED PROGRAMS CALLING CVAF  *
                 BRANCH=YES         BRANCH ENTRY CALL                    *
+         LA    1,CVPL                            LOAD PARAMETER REG 1
+         L     15,16                    LOAD THE CVT
+         L     15,328(,15)              LOAD VS1/VS2 COMMON EXTENSION2
+         L     15,12(,15)               LOAD THE CVAF TABLE ADDRESS
+         L     15,0(,15)                LOAD THE CVAF ADDRESS
+         BALR  14,15                    BRANCH AND LINK TO CVAF
          TM   CVFL1,CV1IVT         IS THIS INDEXED VTOC
          BZ   UNINDXD              READ FORMAT 5 IF NOT
          LTR  REG15,REG15          ANY ERROR
          BZ   NOERROR
          C    REG15,RC04
          BNE  OTHERERR             UNEXPECTED ERROR
          CLI  CVSTAT,STAT032       END OF DATA
          BNE  OTHERERR             UNEXPECTED ERROR
 NOERROR  EQU  *                    BUILD FORMAT 5
          MVC  DS5KEYID,F5ID



          MVC  DS5AVEXT(L'DS5AVEXT+L'DS5EXTAV),EXTS MOVE IN EXTENTS     *
                                    TO DS5FMTID
          MVI  DS5FMTID,C'5'
          MVC  DS5MAVET,EXTS+L'DS5AVEXT+L'DS5EXTAV MOVE REMAINING      *
                                     EXTENTS
          XR   REG15,REG15           RETURN CODE ZERO
          XC   DS5PTRDS,DS5PTRDS     ZERO CHAIN POINTER
          NC   EXTS+L'EXTS-L'DS5AVEXT(L'DS5AVEXT),EXTS+L'EXTS-L'DS5AVEXT*
                                     IS LAST(27TH) EXTENT FROM CVAF     *
                                     ZERO?
          BZ   RETURN                BRANCH IF YES-LEAVE DS5PTRDS ZERO
          MVI  DS5PTRDS+L'DS5PTRDS-1,1 SET DS5PTRDS NONZERO TO SIMULATE *
                                     THERE BEING ANOTHER FORMAT 5
          B    RETURN
 UNINDXD  EQU  *                     CALL ROUTINE TO READ NEXT FORMAT 5
          LINK EP=F5RTN              LINK TO FORMAT 5 ROUTINE. RETURN   *
                                     CODE PASSED BACK IN REG15
+         CNOP  0,4
+         BAL   15,*+20                           LOAD SUP.PARAMLIST ADR
+         DC    A(*+8)                            ADDR OF EP PARAMETER
+         DC    A(0)                DCB ADDRESS PARAMETER       LCOA
+         DC    CL8'F5RTN'                        EP PARAMETER
+         SVC   6                                 ISSUE LINK SVC
 RETURN   EQU  *                     RETURN TO CALLER
          L    13,SAVEAREA+4
          RETURN (14,12),RC=(15)
+         L     14,12(13,0)                       RESTORE REGISTER 14
+         LM    0,12,20(13)                       RESTORE THE REGISTERS
+         BR    14                                RETURN
 OTHERERR EQU  *                     ERROR
          L    REG15,RC04            ERROR RETURN CODE
          B    RETURN
 DSCB     DSECT
          IECSDSL1 (5)
+IECSDSL5 EQU   *                   FORMAT 5 DSCB
+IECSDSF5 EQU   IECSDSL5
+DS5KEYID DS    XL4                 KEY IDENTIFIER
+DS5AVEXT DS    XL5                 AVAILABLE EXTENT
+*        BYTES 1 - 2     RELATIVE TRACK ADDRESS OF THE FIRST TRACK
+*                        IN THE EXTENT
+*        BYTES 3 - 4     NUMBER OF UNUSED CYLINDERS IN THE EXTENT
+*        BYTE  5         NUMBER OF ADDITIONAL UNUSED TRACKS
+DS5EXTAV DS    XL35                SEVEN AVAILABLE EXTENTS
+DS5FMTID DS    CL1                 FORMAT IDENTIFIER
+DS5MAVET DS    XL90                EIGHTEEN AVAILABLE EXTENTS
+DS5PTRDS DS    XL5                 POINTER TO NEXT FORMAT 5 DSCB
+DS5END   EQU   *
 EXAMPLE5 CSECT
 NOTIXRC  DC   F'4'                 CVAFTST RETURN CODE-UNINDEXED
 RC04     DC   F'4'                 RETURN CODE 4
 F5ID     DC   XL4'0505050505'      FORMAT 5 FIELD, DS5KEYID
 SAVEAREA DS   18F                  REGISTER SAVE AREA
 EXTABL   DS   0CL(1+(KF5+1)*L'DS5AVEXT) EXTENT TABLE
 EXTNO    DC   AL1(KF5+1)           NUMBER OF EXTENTS IN TABLE
 EXTS     DS   CL((KF5+1)*L'DS5AVEXT) EXTENTS
 CVPL     CVAFDSM ACCESS=MAPDATA,                                       *
                COUNT=NO,           DO NOT COUNT EXTENTS                *
                MAP=VOLUME,         ACCESS VOLUME SPACE MAP             *
                EXTENTS=EXTABL,     EXTENT TABLE ADDRESS                *
                MF=L                LIST FORM OF MACRO
+         CNOP  0,4
```

```
+CVPL     EQU   *
+         DC    CL4'CVPL'                EBCDIC 'CVPL'
+         DC    AL2(ICV9E-CVPL)          LENGTH OF CVPL
+         DC    XL1'0A'                  FUNCTION CODE
+         DC    XL1'00'                  STATUS INFORMATION
+         DC    B'00100001'              FIRST FLAG BYTE
+         DC    B'00000000'              SECOND FLAG BYTE
+         DC    H'0'                     RESERVED
+         DC    A(0)                     UCB ADDRESS
+         DC    A(0)                     DATA SET NAME ADDRESS
+         DC    A(0)                     BUFFER LIST ADDRESS
+         DC    A(0)                     INDEX VIR'S BUFFER LIST ADDRESS
+         DC    A(0)                     MAP VIR'S BUFFER LIST ADDRESS
+         DC    A(0)                     I/O AREA ADDRESS
+         DC    A(0)                     DEB ADDRESS
+         DC    A(0)                     ARGUMENT ADDRESS
+         DC    A(0)                     SPACE PARAMETER LIST ADDRESS
+         DC    A(EXTABL)                EXTENTS TABLE ADDRESS
+         DC    A(0)                     NEW VRF VIXM BUFFER LIST ADDR
+         DC    A(0)                     VRF DATA ADDRESS
+         DC    A(0)                     COUNT AREA ADDRESS
+ICV9E    EQU   *                        END OF CVPL
          ORG   CVPL                OVERLAY CVPL WITH EXPANSION OF MAP
 CVPLMAP  ICVAFPL DSECT=NO
+**********************************************************************
+*        CVAF PARAMETER LIST
+**********************************************************************
+CVPLMAP  DS    0F                      CVAF PARAMETER LIST
+CVLBL    DS    CL4                     EBCDIC 'CVPL'
+CVLTH    DS    H                       LENGTH OF CVPL
+CVFCTN   DS    XL1                     FUNCTION BYTE
+CVDIRD   EQU   X'01'                   CVAFDIR ACCESS=READ
+CVDIWR   EQU   X'02'                   CVAFDIR ACCESS=WRITE
+CVDIRLS  EQU   X'03'                   CVAFDIR ACCESS=RLSE
+CVSEQGT  EQU   X'04'                   CVAFSEQ ACCESS=GT
+CVSEQGTE EQU   X'05'                   CVAFSEQ ACCESS=GTEQ
+CVDMIXA  EQU   X'06'                   CVAFDSM ACCESS=IXADD
+CVDMIXD  EQU   X'07'                   CVAFDSM ACCESS=IXDLT
+CVDMALC  EQU   X'08'                   CVAFDSM ACCESS=ALLOC
+CVDMRLS  EQU   X'09'                   CVAFDSM ACCESS=RLSE
+CVDMMAP  EQU   X'0A'                   CVAFDSM ACCESS=MAPDATA
+CVVOL    EQU   X'0B'                   CVAFVOL ACCESS=VIBBLD
+CVVRFRD  EQU   X'0C'                   CVAFVRF ACCESS=READ
+CVVRFWR  EQU   X'0D'                   CVAFVRF ACCESS=WRITE
+CVSTAT   DS    XL1                     STATUS INFORMATION (SEE LIST    X
+                                       BELOW)
+CVFL1    DS    XL1                     FIRST FLAG BYTE
+CV1IVT   EQU   X'80'                   INDEXED VTOC ACCESSED
+CV1IOAR  EQU   X'40'                   IOAREA=KEEP
+CV1PGM   EQU   X'20'                   BRANCH=(YES,PGM)
+CV1MRCDS EQU   X'10'                   MAPRCDS=YES
+CV1IRCDS EQU   X'08'                   IXRCDS=KEEP
+CV1MAPIX EQU   X'04'                   MAP=INDEX
+CV1MAPVT EQU   X'02'                   MAP=VTOC
+CV1MAPVL EQU   X'01'                   MAP=VOLUME
+CVFL2    DS    XL1                     SECOND FLAG BYTE
+CV2HIVIE EQU   X'80'                   HIVIER=YES
+CV2VRF   EQU   X'40'                   VRF DATA EXISTS
+CV2CNT   EQU   X'20'                   COUNT=YES
+CV2RCVR  EQU   X'10'                   RECOVER=YES
+CV2SRCH  EQU   X'08'                   SEARCH=YES
+CV2DSNLY EQU   X'04'                   DSNONLY=YES
+CV2VER   EQU   X'02'                   VERIFY=YES
+CV2NLEVL EQU   X'01'                   OUTPUT-NEW HIGHEST LEVEL VIER
+*                                      CREATED
+         DS    H                       RESERVED
+CVUCB    DS    A                       UCB ADDRESS
+CVDSN    DS    A                       DATA SET NAME ADDRESS
+CVBUFL   DS    A                       BUFFER LIST ADDRESS
+CVIRCDS  DS    A                       INDEX VIR'S BUFFER LIST ADDRESS
```

```
+CVMRCDS  DS    A                       MAP VIR'S BUFFER LIST ADDRESS
+CVIOAR   DS    A                       I/O AREA ADDRESS
+CVDEB    DS    A                       DEB ADDRESS
+CVARG    DS    A                       ARGUMENT ADDRESS
+CVSPACE  DS    A                       SPACE PARAMETER LIST ADDRESS
+CVEXTS   DS    A                       EXTENT TABLE ADDRESS
+CVBUFL2  DS    A                       NEW VRF VIXM BUFFER LIST ADDR
+CVVRFDA  DS    A                       VRF DATA ADDRESS
+CVCTAR   DS    A                       COUNT AREA ADDRESS
+CVPLNGTH EQU   *-CVPLMAP
+*              VALUES OF CVSTAT
+*(THIS PART OF THE ICVAFPL MACRO EXPANSION IS NOT SHOWN)
          END
```

APPENDIX C. RETURN CODES FROM VTOC ACCESS MACROS

RETURN CODES FROM THE CVAFDIR MACRO

On return from CVAF, register 1 contains the address of the CVAF parameter list (CVPL), and register 15 contains one of the following return codes:

Code **Meaning**

0(00) The request was successful. However, if the CVAFDIR request is to read or write a DSCB and a VTOC index structure error is encountered, the CVSTAT field indicates the structure error encountered. (CVSTAT code descriptions are in Appendix B.)

4(04) An error occurred. The CVSTAT field in the CVPL contains an indication of the cause of the error. (CVSTAT code descriptions are in Appendix B.)

8(08) Invalid VTOC index structure while processing a request to read or write a VTOC index record. The CVSTAT field in the CVPL contains an indication of the cause of the error. (CVSTAT code descriptions are in Appendix B.)

12(0C) The CVAF parameter list is not in your protect key, or is invalid (the ID is invalid, or the length field is incorrect, or the CVFCTN field is invalid). The CVPL has not been modified.

16(10) An I/O error was encountered.

RETURN CODES FROM THE CVAFDSM MACRO

On return from CVAF, register 1 contains the address of the CVAF parameter list (CVPL), and register 15 contains one of the following return codes:

Code **Meaning**

0(00) The request was successful.

4(04) End of data (CVSTAT is set to decimal 32), or an error was encountered. The CVSTAT field in the CVPL contains an indication of the cause of the error. (CVSTAT code descriptions are in Appendix B.)

8(08) Invalid VTOC index structure. CVSTAT contains an indication of the cause of the error. (CVSTAT code descriptions are in Appendix B.)

12(0C) The CVAF parameter list is not in your protect key, or is invalid (the ID is invalid, or the length field is incorrect, or the CVFCTN field is invalid). The CVPL has not been modified.

16(10) An I/O error was encountered.

RETURN CODES FROM THE CVAFSEQ MACRO

On return from CVAF, register 1 contains the address of the CVAF parameter list (CVPL), and register 15 contains one of the following return codes:

Code **Meaning**

0(00) The request was successful.

4(04) End of data (CVSTAT is set to decimal 32), or an error was encountered. The CVSTAT field in the CVPL contains an indication of the cause of the error. Error descriptions are in Appendix D.

8(08) Invalid VTOC index structure. CVSTAT contains an indication of the cause of the error. Error descriptions are in Appendix D.

12(0C) The CVAF parameter list is not in your protect key, or is invalid (the ID is invalid, or the length field is incorrect, or the CVFCTN field is invalid). The CVPL has not been modified.

16(10) An I/O error was encountered.

RETURN CODES FROM THE CVAFTST MACRO

On return from CVAF, register 15 contains one of the following return codes:

Code **Meaning**

0(00) The system does not support an indexed VTOC. The volume should be considered to have an nonindexed VTOC. The UCB was not inspected to determine its validity or status.

4(04) The system supports an indexed VTOC, but the volume has an nonindexed VTOC.

8(08) The system supports an indexed VTOC and the volume has an indexed VTOC.

12(0C) The system supports an indexed VTOC, but the volume is not mounted or the VIB is not initialized for it, so the status (indexed or nonindexed) of the VTOC can not be determined.

16(10) The system supports an indexed VTOC, but the unit is not a DASD or has a VIO UCB, or the UCB address is invalid.

APPENDIX D. VTOC ERROR MESSAGE AND ASSOCIATED CODES

ERROR MESSAGE

When CVAF finds an error in a VTOC index, it issues this message:

```
IEC606I VTOC INDEX DISABLED ON dev,volser,
 code,[rba[,secno,offset]]
```

In addition, CVAF puts a return code in the CVSTAT field of the CVPL.

EXPLANATION

The Common VTOC Access Facility (CVAF) detected a VTOC index error on the device 'dev' with volume serial number 'volser'. 'code' is a number that represents the kind of VTOC index error encountered. These codes and their meanings are in Appendix C. 'rba' is the RBA of the VIR in the VTOC index that contains a structure error indicated by 'code'. If the VIR is a VIER, the section number in the VIER containing the VTOC index entry is supplied in 'secno', and the offset into the section of that VTOC index entry is supplied in 'offset'.

SYSTEM ACTION

The VTOC index is disabled by zeroing the index bit in the format-4 DSCB and setting the bit in the first high-level VIER which indicates invalid VTOC index structure. The VTOC will be converted to nonindexed format when DADSM next allocates space on the volume. A system dump is written to the SYS1.DUMP data set, and an entry is made in the SYS1.LOGREC data set. The message IEC604I (which indicates that the VTOC convert routines have been used) will be issued later.

PROGRAMMER RESPONSE

Examine the system dump and a print of the VTOC index, and use the information in message IEC606I to determine the cause of the VTOC index structure error.

ROUTING AND DESCRIPTOR CODES

The routing codes are 4 (direct access pool) and 10 (system/error maintenance), and the descriptor code is 4 (system status).

CODES PUT IN THE CVSTAT FIELD

Code	Meaning
0(00)	No error.
1(01)	Data set name not found.
2(02)	Argument is outside VTOC extents or RBA range of VTOC index.
4(04)	Invalid parameter supplied (wrong key).
5(05)	DSN keyword omitted.

Code	Meaning
6(06)	Not authorized to perform this function.
7(07)	Buffer list omitted.
8(08)	DEB invalid or omitted or not open to VTOC.
9(09)	IOAREA=KEEP and user not authorized, or I/O area supplied and user not authorized
10(0A)	Function not supported on indexed VTOC.
11(0B)	DSCB is not format-0 DSCB and VERIFY=YES.
12(0C)	MAPRCDS=YES and/or IXRCDS=KEEP but VTOC is nonindexed.
13(0D)	IXRCDS=KEEP not specified for CVAFDSM ACCESS=IXADD or IXDLT.
14(0E)	CTAREA keyword omitted.
15(0F)	UCB invalid, volume not mounted; VIO unit, not DASD.
17(11)	DSCB length invalid for the function requested: 96 bytes for CVAFDIR ACCESS=WRITE,VERIFY=YES; 96 bytes for CVAFSEQ reading in data-set-name sequence; 140 bytes for CVAFSEQ reading in physical sequence.
19(13)	UCB omitted and CVAF I/O area not supplied.
22(16)	Data set name already supplied.
23(17)	Invalid DSN supplied (44X'FF' is a reserved data set name).
24(18)	ARG keyword not supplied.
25(19)	Conflicting or incomplete information specified in the space table for a CVAFDSM ACCESS=ALLOC, MAP=VOLUME request.
27(1B)	VTOC index full. No free VIRs available and a VIER split is required.
28(1C)	Space keyword omitted (CVSPACE field zero in CVPL).
29(1D)	CVAFDSM ACCESS=ALLOC: No format 0 DSCB available (MAP=VTOC), or VTOC index full (MAP=INDEX), or volume space not available (MAP=VOLUME).
30(1E)	CVAFDSM ACCESS=ALLOC: CCHHR (MAP=VTOC) or RBA MAP=INDEX or volume space extent (MAP=VOLUME) already allocated.
31(1F)	CVAFDSM ACCESS=ALLOC: CCHHR supplied outside VTOC extents (MAP=VTOC), or RBA outside VTOC index extents (MAP=INDEX), or volume space extent invalid or outside volume (MAP=VOLUME).
32(20)	End of data. CVAFDSM ACCESS=MAPDATA: no more free extents in VPSM. CVAFSEQ: no more names in index or DSCBs in VTOC. For indexed access, no DSN in VTOC index with higher or higher-or-equal key than that supplied. For physical-sequential access, no DSCB in the VTOC has a higher argument than that supplied. For a multiple DSCB request, the last DSCB in the VTOC was read and more DSCBs were requested.
33(21)	EXTENTS keyword omitted, or supplied number of extents is zero.

Code	Meaning
34(22)	CVAFDSM ACCESS=RLSE1 format 0 DSCB already free (MAP=VTOC), or VIER already unallocated (MAP=INDEX) or volume space extent already unallocated (MAP=VOLUME).
42(2A)	VRF data supplied for write too long.
43(2B)	Buffer list is for VIRs, but a DSCB buffer list is required.
44(2C)	No buffer list entry found.
45(2D)	Invalid DSCB buffer length (neither 96 nor 140) in buffer list entry, or VIR buffer length not equal to VIB VIR size.
46(2E)	Neither TTR nor CCHHR bits set in buffer list entry to be used in writing a 140-byte DSCB.
47(2F)	More than one of the TTR, CCHHR, and RBA bits set in the buffer list entry.
48(30)	Both the DSCB and VIR bits set in the buffer list header.
49(31)	RBA bit set in a buffer list entry for a DSCB buffer list.
50(32)	TTR or CCHHR bit set in buffer list entry but buffer list header indicates buffer list is for a VIR.
52(34)	Combination of MAP and COUNT not supported.
53(35)	MAP omitted.
54(36)	Buffer list for a VIR chained to or from a buffer list for a DSCB.
55(37)	Unauthorized caller and VIB not initialized.
56(38)	MAPRCDS=YES not specified but required.
57(39)	Buffer list for a DSCB supplied but buffer list for a VIR required (in MAPRCDS or IXRCDS buffer list address in CVAF parameter list).
58(3A)	Neither the VIR nor DSCB bit set in a buffer list header.
60(3C)	Invalid or conflicting setting of allocate option byte in space parameter
127(7F)	I/O error occurred.
128(80)	Reserved.
129(81)	The first high-level VIER as indicated in the VIXM does not have the flag bit set indicating it is the first high-level VIER.
130(82)	A horizontal or vertical VIER pointer is outside the RBA range of the VTOC index.
131(83)	A vertical VIER pointer points to a VIR which is not a VIER (invalid ID in header).
132(84)	A level n vertical index entry pointer points to a VIER which is not at level n-1.

Code	Meaning
133(85)	Level n horizontal index entry pointer points to VIER which is not at level n.
134(86)	Horizontal VIER/map pointer points to a VIR which is not a VIER/map (invalid ID in header).
135(87)	Horizontal map pointer points to VIR which is not one of the first n VTOC index records (n is recorded in VIXM field VIMRCDS), or the first record in the VTOC index is not a VIXM.
136(88)	A level-1 index entry contains a CCHHR pointer which is outside the VTOC extent.
137(89)	The first high-level VIER, as indicated in the VIB, does not have the flag bit set indicating it is the first high-level VIER. (This error is either recovered from by updating the VIB from the VIXM, or the error is changed to 129.)
138(8A)	The RBA of the VTOC index VIR does not match the RBA recorded in the header of the record.
139(8B)	The first record of a map (VIXM, VPSM, or VMDS) is not one of the first n VTOC index records (n is recorded in the VIXM field, VIMRCDS).
140(8C)	The data set name in a level n+1 VIER entry is lower than the high key of the level n VIER that the level n+1 VIER entry points to.
141(8D)	First high-level VIER structure error bit is on.
142(8E)	I/O error indicating the VTOC index is not formatted correctly.
143(8F)	Either the index bit is zero or the DOS bit is zero in the format-4 DSCB of a VTOC previously found to be an indexed VTOC.
144(90)	No SYS1.VTOCIX.nnn data set name in a VTOC whose format-4 DSCB has the index bit on, indicating the VTOC has an index.
145(91)	The data set name in a level n+1 VIER entry is higher than the high key of the level n VIER that the level n+1 VIER entry points to
146(92)	Four or more high-level VIERs were encountered.
147(93)	Too many levels in the VTOC index. The length of the search list was exceeded.
148(94)	VIER invalid, because offset to last section is invalid.
149(95)	VIER invalid, because offset to last entry in a section is invalid.
150(96)	Media Manager initialization failed.
151(97)	Level-2 or higher VIER contains fewer than two entries.
152(98)	RECOVER=YES specified but the static text module (ICVIXSTO) indicates recovery is not permitted.

Code **Meaning**

153(99) The format-4 DSCB on an indexed VTOC is written with either the index- or DOS-bit zeroed on an indexed VTOC.

154(9A) A space map extends over more than 10 VTOC index records.

155(9B) Data set name not found in section with key greater than or equal to the name being searched for. The VIER section containing the name is invalid.

156(9C) Invalid VIER horizontal pointer. Horizontal pointer of VIER1 points to VIER2 whose high key is lower than or equal to the high key of VIER1.

157(9D) Could not find entry in level-2 or higher VIER that matches the high key of the VIER.

158(9E) Invalid section length or invalid number of sections in a VIER header.

159(9F) The first high-level VIER pointed to by the VIB has an invalid ID in the header.

APPENDIX E. EXAMPLE OF AN OPEN EXIT MODULE

PROCESSING IN IFG0EX0B

The following program listing is a sample of IFG0EX0B. The four subroutines (BUFNO, SCREEN, RLSE, and SQTY) show examples of the kind of processing that can be done in your installation's version of IFG0EX0B.

The BUFNO subroutine defaults the number of buffers for QSAM DCBs (DCBBUFNO) if the value is zero when the exit is given control. The block size in the DCB (DCBBLKSI) is used, together with a fixed amount of storage (64K bytes in the example) to determine a buffer number. A buffer number is limited to a fixed value (32 in the example). Storage quantity and maximum buffer number are contained in two tables, DAMAX and TPMAX, which are used for DASD devices and tape devices, respectively. Storage quantity is expressed in units of 1024 (1K) bytes. The values in the DAMAX and TPMAX tables can be altered by your installation.

The SCREEN subroutine determines those cases in which the succeeding subroutines, RLSE and SQTY, should be executed. DASD sequential and partitioned data sets being processed by BSAM or QSAM and opened for OUTPUT or OUTIN are selected. The VTOC data set and data sets starting with 'SYS1.' (system data sets) are excluded. An installation may want to make further selection tests.

REQUESTING PARTIAL RELEASE

The RLSE subroutine sets on the partial release indicators in the JFCB if the number of extents in the data set is less than a fixed value (8 in the example). It sets off the partial release indicators in the JFCB if the number of extents in the data set is equal or greater than a fixed value (8 in the example). Partitioned data sets are not processed, because they may be opened many times to write one new member for each OPEN/CLOSE.

UPDATING THE SECONDARY SPACE DATA

The SQTY subroutine provides a default secondary space quantity if none is specified. The default is one half of the primary space quantity if it is greater than one. If the primary quantity is zero, secondary is set to a fixed default number of tracks (5 in the example). If the primary quantity is one, secondary is set to the same fixed default (5); note that, in this case, the secondary quantity is in units of tracks, cylinders, or average blocks, depending on the unit of the primary quantity.

If the secondary space quantity is not zero, the SQTY subroutine tests the number of extents in the data set. If the number of extents is equal to or greater than a fixed value (10 in the example), then the secondary quantity is increased by 50% if it is greater than 1. It is set to a default quantity (5 in the example) if the secondary quantity is one; note that, in this case, the secondary quantity is in units of tracks, cylinders, or average blocks, depending on that of the primary quantity.

```
IFG0EX0B CSECT
************************************************************************
*                                                                      *
* FUNCTION =                                                           *
*        FOUR SAMPLE ROUTINES ARE SUPPLIED.                            *
*                                                                      *
*    BUFNO - DEFAULT DCBBUFNO                                          *
*        DCBBUFNO (NUMBER OF BUFFERS) IS DEFAULTED FOR                 *
*        OPENS TO PHYSICAL SEQUENTIAL AND PARTITIONED DATA SETS        *
*        ON DASD AND TAPE USING QSAM, FOR WHICH DCBBUFNO IS ZERO.      *
*        DCBBUFNO FOR SYSIN, SYSOUT, TERMINAL, AND DUMMY DATA SETS     *
*        IS SET TO THE EQUATE, INOUTBNO, OR THE VALUE IN THE           *
*        FULLWORD, INOUTBN.                                            *
*                                                                      *
*        DCBBUFNO IS SET TO THE NUMBER OF DCBBLKSZ BUFFERS WHICH       *
*        FIT IN A GIVEN AMOUNT OF STORAGE. THE AMOUNT OF STORAGE IS    *
*        DEFINED BY THE EQUATES, DAMXK AND TPMXK (OR THE FULLWORDS     *
*        AT LABELS, DAMAXK AND TPMAXK), FOR DASD AND                   *
*        TAPE, RESPECTIVELY. THE EQUATES DEFINE THE AMOUNT OF          *
*        STORAGE FOR BUFFERS IN UNITS OF 1024 (IF DAMXK IS 32, THEN    *
*        THE AMOUNT OF STORAGE IS 32K, OR 32768).                      *
*        DAMXK OR TPMXK TIMES 1024 IS DIVIDED BY DCBBLKSI TO           *
*        DETERMINE THE NUMBER OF BUFFERS TO DEFAULT.                   *
*                                                                      *
*        THE EQUATES, DAMXBNO AND TPMXBNO, OR THE FULLWORDS            *
*        AT LABELS, DAMAXBNO AND TPMAXBNO,                             *
*        DEFINE THE MAXIMUM NUMBER OF BUFFERS TO BE                    *
*        DEFAULTED FOR DASD AND TAPE IF THE CALCULATION, ABOVE,        *
*        RESULTS IN A LARGER NUMBER.                                   *
*                                                                      *
*    SCREEN - SCREEN OUT CASES FOR RLSE, SQTY                          *
*                                                                      *
*    RLSE - SET OR ZERO PARTIAL RELEASE                                *
*        THIS ROUTINE SETS PARTIAL RELEASE FOR DASD PS (NOT PO) DATA   *
*        SETS BEING OPENED FOR OUTPUT OR OUTIN.                        *
*                                                                      *
*        PARTIAL RELEASE IS SET ON IF THE NUMBER OF EXTENTS IS LESS    *
*        THAN A QUANTITY DEFINED BY THE EQUATE, RLSE1, OR THE BYTE,    *
*        EXTRLSE1.                                                     *
*                                                                      *
*        PARTIAL RELEASE IS SET OFF IF THE NUMBER OF EXTENTS IS NOT    *
*        LESS THAN A QUANTITY DEFINED BY THE EQUATE, RLSE0, OR THE     *
*        BYTE, EXTRLSE0.                                               *
*                                                                      *
*    SQTY - SET OR UPDATE SECONDARY SPACE QUANTITY                     *
*        THIS ROUTINE UPDATES THE SECONDARY SPACE                      *
*        QUANTITY FOR DASD PS OR PO DATA SETS BEING                    *
*        OPENED FOR OUTPUT OR OUTIN.                                   *
*                                                                      *
*        IF THE SECONDARY QUANTITY IS NOT ZERO,                        *
*        AND IF THE NUMBER OF EXTENTS IN THE DATA SET IS               *
*        AT LEAST EQUAL TO THE QUANTITY IN THE EQUATE, EXTSQT (OR      *
*        THE BYTE AT LABEL, EXTSQTY), THEN:                            *
*        1. IF THE SECONDARY QUANTITY IS GREATER THAN ONE,             *
*        SECONDARY QUANTITY IS INCREASED BY ONE HALF                   *
*        (50%).                                                        *
*        2. IF THE SECONDARY QUANTITY IS ONE,                          *
*        SECONDARY QUANTITY IS SET TO THE VALUE IN THE FULLWORD        *
*        AT LABEL, SQTYDFLT (EQUAL TO THE EQUATE, SQTYDFL).            *
*                                                                      *
*        IF THE SECONDARY QUANTITY IS NOT ZERO,                        *
*        AND IF THE NUMBER OF EXTENTS IN THE DATA SET IS               *
*        LESS THAN THE QUANTITY IN THE EQUATE, EXTSQT (OR              *
*        THE BYTE AT LABEL, EXTSQTY), SECONDARY QUANTITY               *
*        IS LEFT UNCHANGED.                                            *
*                                                                      *
*       IF SECONDARY QUANTITY IS ZERO, IT IS SET TO ONE HALF          *
*       OF PRIMARY QUANTITY IF PRIMARY IS NOT ZERO OR ONE.            *
*       IF PRIMARY QUANTITY IS ZERO, THE SPACE TYPE IS SET TO TRACKS,*
*       AND SECONDARY QUANTITY IS SET TO THE VALUE IN THE FULLWORD   *
*       AT LABEL SQTYDFLT (EQUAL TO THE EQUATE, SQTYDFL).             *
*       IF PRIMARY QUANTITY IS ONE, SECONDARY QUANTITY IS SET TO      *
*       VALUE IN THE FULLWORD AT LABEL SQTYDFLT (EQUAL TO THE         *
*       EQUATE, SQTYDFL).                                              *
*                                                                      *
* NOTES = SEE BELOW                                                    *
*                                                                      *
*    DEPENDENCIES =                                                    *
*            CLASS ONE CHARACTER CODE.  THE EBCDIC CHARACTER CODE      *
*            WAS USED FOR ASSEMBLY.  THE MODULE MUST BE REASSEMBLED    *
*            IF A DIFFERENT CHARACTER SET IS USED FOR EXECUTION.       *
*                                                                      *
*    RESTRICTIONS = NONE                                               *
*                                                                      *
*    REGISTER CONVENTIONS =                                            *
*            R1  OIEXL ADDRESS                                         *
*            R2  DCB ADDRESS                                           *
*            R3  UCB ADDRESS                                           *
*            R4  DCB BLOCK SIZE                                        *
*            R5  ADDRESS OF TPMAX OR DAMAX TABLES                      *
*            R6  EVEN REGISTER OF EVEN/ODD PAIR                        *
*            R7  ODD REGISTER OF EVEN/ODD PAIR                         *
*            R8  TIOT ENTRY ADDRESS                                    *
*            R8  JFCB ADDRESS                                          *
*            R10 FORMAT 1 DSCB ADDRESS                                 *
*            R11 SAVE RETURN CODE                                      *
*            R13 SAVE AREA ADDRESS                                     *
*            R14 RETURN ADDRESS                                        *
*            R15 BASE REGISTER                                         *
*                                                                      *
*    PATCH LABEL = PATCH                                               *
*                                                                      *
* MODULE TYPE = CONTROL (OPEN, CLOSE, EOV DATA MANAGEMENT)             *
*                                                                      *
*    PROCESSOR = ASSEMBLER XF                                          *
*                                                                      *
*    MODULE SIZE = SEE EXTERNAL SYMBOL DICTIONARY                      *
*                                                                      *
*    ATTRIBUTES = REENTRANT, REFRESHABLE,READ-ONLY, ENABLED,           *
*                  PRIVILEGED, SUPERVISOR STATE, KEY ZERO,             *
*                  LINK PACK AREA RESIDENT/PAGEABLE                    *
*                                                                      *
* ENTRY POINT = IFG0EX0B                                               *
*                                                                      *
*    PURPOSE = SEE FUNCTION                                            *
*                                                                      *
*    LINKAGE =                                                         *
*       FROM IFG0196L:                                                 *
*          BALR 14,15                                                  *
*                                                                      *
* INPUT = STANDARD LINKAGE CONVENTIONS                                 *
*                                                                      *
* OUTPUT =   DCBBUFNO DEFAULTED                                        *
*            PARTIAL RELEASE SET OR RESET                              *
*            CONTIGUOUS FLAG SET TO ZERO                               *
*            SECONDARY SPACE REQUEST MODIFIED                          *
*          RETURN CODE IN REGISTER 15                                  *
*            0 IF JFCB NOT MODIFIED                                    *
*            4 IF JFCB MODIFIED                                        *
```

```
 *                                                                     *
 * EXIT-NORMAL =                                                       *
 *          BR 14                                                      *
 *                                                                     *
 * EXIT-ERROR =                                                        *
 *        NONE                                                         *
 *                                                                     *
 * EXTERNAL REFERENCES = SEE BELOW                                     *
 *                                                                     *
 *    ROUTINES = NONE                                                  *
 *                                                                     *
 *    DATA AREAS = NONE                                                *
 *                                                                     *
 *    CONTROL BLOCK = NONE                                             *
 *                                                                     *
 * TABLES = NONE                                                       *
 *                                                                     *
 * MACROS = MODESET, IECOIEXL, IHADCB, IEFUCBOB, IEFTIOT1, IEFJFCBN,    *
 *          IECSDSL1                                                   *
 *                                                                     *
 ***********************************************************************
 ***********************************************************************
 *
 *        REGISTER EQUATES
 *
 ***********************************************************************
 R1       EQU   1                  OIEXL PARAMETER LIST ADDRESS
 RDCB     EQU   2                  DCB ADDRESS
 RUCB     EQU   3                  UCB ADDRESS
 RBKSIZ   EQU   4                  DCB BLOCK SIZE
 RMAX     EQU   5                  ADDRESS OF TPMAX OR DAMAX
 REVEN    EQU   6                  EVEN REGISTER OF EVEN/ODD PAIR
 RODD     EQU   7                  ODD REGISTER OF EVEN/ODD PAIR. HAS   *
                                   DCBBUFNO DEFAULT
 RTIOT    EQU   8                  TIOT ENTRY ADDRESS
 RJFCB    EQU   9                  JFCB ADDRESS
 RDSCB    EQU   10                 FORMAT 1 DSCB ADDRESS
 RINCODE  EQU   11                 INTERNAL RETURN CODE
 R12      EQU   12
 RSAVE    EQU   13                 SAVE AREA ADDRESS
 RET      EQU   14                 RETURN ADDRESS
 RCODE    EQU   15                 BASE REGISTER/RETURN CODE ON EXIT
 ***********************************************************************
 *
 *        RETURN CODE
 *
 ***********************************************************************
 MODJFCB  EQU   4                  RETURN CODE IF JFCB MODIFIED
          USING IFG0EX0B,RCODE
 ***********************************************************************
 *
 *        START OF SAMPLE PROGRAM
 *
 ***********************************************************************
          B     AFTRID1
          DC    C'IFG0EX0B JDM1137 &SYSDATE'
+         DC    C'IFG0EX0B JDM1137 05/01/81'
 AFTRID1  SAVE  (14,12)            SAVE REGISTERS
+AFTRID1  DS    0H
+         STM   14,12,12(13)                      SAVE REGISTERS
          XR    RINCODE,RINCODE    ZERO RETURN CODE
          USING OIEXL,R1           PARAMETER LIST
          BAL   RET,BUFNO          DEFAULT BUFNO
          BAL   RET,SCREEN         SCREEN OUT CASES WHERE RLSE, *
                                   AND SQTY SHOULD NOT BE CALLED
          BAL   RET,RLSE           SET PARTIAL RELEASE
          BAL   RET,SQTY           SET SECONDARY QUANTITY
 EXIT     EQU   *                  RETURN TO CALLER
```

```
***********************************************************************
*        RETURN TO CALLER
***********************************************************************
         LR    RCODE,RINCODE
         RETURN (14,12),RC=(15)   RESTORE REGISTER
+        L     14,12(13,0)                      RESTORE REGISTER 14
+        LM    0,12,20(13)                      RESTORE THE REGISTERS
+        BR    14                               RETURN
BUFNO    EQU   *                  DEFAULT DCB BUFNO
***********************************************************************
*
*        DEFINE DEFAULT VALUES
*          DAMXK   = NUMBER OF K (1024) OF BUFFERS FOR DASD
*          TPMXK   = NUMBER OF K (1024) OF BUFFERS FOR TAPE
*          DAMXBNO = MAXIMUM NUMBER OF BUFFERS FOR DASD
*          TPMXBNO = MAXIMUM NUMBER OF BUFFERS FOR TAPE
*        NOTE THAT DAMXBNO AND TPMXBNO MUST NOT BE GREATER THAN 255
*
***********************************************************************
DAMXK    EQU   64                 64K BUFFERS FOR DASD
TPMXK    EQU   64                 64K BUFFERS FOR TAPE
DAMXBNO  EQU   32                 32 BUFFERS MAXIMUM FOR DASD
TPMXBNO  EQU   32                 32 BUFFERS MAXIMUM FOR TAPE
INOUTBNO EQU   1                  DCBBUFNO DEFAULT FOR SYSIN, SYSOUT,  *
                                  AND DD DUMMY
ONEK     EQU   10                 SHIFT ARGUMENT TO MULTIPLY BY 1024
         B     AFTRID2
         DC    CL8'BUFNO'         BUFNO ROUTINE ID
AFTRID2  BCR   0,RET              NOP RETURN
         L     RDCB,OIEXPDCB      PROTECTED COPY OF DCB
         USING IHADCB,RDCB
***********************************************************************
*        DO NOT PROCESS EXCP, BSAM, DSORG NOT PS OR PO,
*                       DCBBUFNO SPECIFIED
***********************************************************************
         TM    DCBMACF1,DCBMRECP  EXCP DCB?
         BO    RETBUFNO           RETURN IF EXCP
         TM    DCBMACF1,DCBMRRD   READ MACRO
         BO    RETBUFNO           RETURN IF READ-NOT QSAM
         TM    DCBMACF2,DCBMRWRT  WRITE MACRO
         BO    RETBUFNO           RETURN IF WRITE-NOT QSAM
         TM    DCBDSRG1,DCBDSGPS+DCBDSGPO PS OR PO
         BZ    RETBUFNO           EXIT IF NOT PS OR PO
         CLI   DCBBUFNO,0         IS DCBBUFNO SPECIFIED
         BNE   RETBUFNO           RETURN IF DCBBUFNO SPECIFIED
***********************************************************************
*        DEFAULT DCBBUFNO TO 1 FOR SYSIN, SYSOUT, TERMINAL, DUMMY
***********************************************************************
         L     RTIOT,OIEXTIOT     TIOT ENTRY ADDRESS
         USING TIOENTRY,RTIOT
         L    RODD,INOUTBN        BUFNO DEFAULT FOR SYSIN/SYSOUT/      *
                                  DD DUMMY
         TM    TIOELINK,TIOESSDS+TIOTTERM SYSIN/SYSOUT OR TERMINAL
         BNZ   STORE              BRANCH IF SYSIN OR SYSOUT OR TERMINAL
         L     RJFCB,OIEXJFCB     JFCB ADDRESS
         USING INFMJFCB,RJFCB
         CLC   JFCBDSNM(L'NULLFILE),NULLFILE DUMMY DATA SET
         BE    STORE              BRANCH IF DUMMY
***********************************************************************
*        EXIT IF NO UCB ADDRESS OR BLOCK SIZE NOT POSITIVE
***********************************************************************
         L     RUCB,OIEXUCB       UCB ADDRESS
         LTR   RUCB,RUCB          ANY UCB?
         BZ    RETBUFNO           EXIT IF NO UCB
         LH    RBKSIZ,DCBBLKSI    DCB BLOCK SIZE
         LTR   RBKSIZ,RBKSIZ      ANY BLOCK SIZE?
         BNP   RETBUFNO           RETURN IF NO BLOCK SIZE
```

```
 ***********************************************************************
 *        GET TAPE OR DASD MAX TABLE
 ***********************************************************************
          USING UCBOB,RUCB
          TM    UCBTBYT3,UCB3DACC  DASD UCB?
          LA    RMAX,DAMAX         MAX TABLE FOR DASD
          BO    CALC               BRANCH IF DASD
          TM    UCBTBYT3,UCB3TAPE  TAPE UCB?
          LA    RMAX,TPMAX         MAX TABLE FOR TAPE
          BZ    RETBUFNO           RETURN IF NOT DASD OR TAPE
 CALC     EQU   *                  DEFAULT DCBBUFNO
 ***********************************************************************
 *        CALCULATE DEFAULT BUFFER NUMBER
 ***********************************************************************
          USING MAX,RMAX
          XR    REVEN,REVEN        ZERO EVEN REG
          L     RODD,MAXBUF        MAXIMUM STORAGE FOR BUFFERS
          SLL   RODD,ONEK          SHIFT TO MULTIPLY BY 1024
          DR    REVEN,RBKSIZ       DIVIDE MAS BUFFER SPACE BY BKSI
          C     RODD,MAXBNO        ARE THERE TOO MANY BUFFERS?
          BNH   STORE              USE CALCULATION IF NOT TOO LARGE
          L     RODD,MAXBNO        USE MAXIMUM NUMBER OF BUFFERS
 STORE    EQU   *                  DEFAULT DCBBUFNO FOR USER/COPY DCB
          STC   RODD,DCBBUFNO      PUT IN PROTECTED COPY OF DCB
          L     RDCB,OIEXUDCB      USER DCB
          XR    REVEN,REVEN        MODESET USES REG 6 = REVEN
          MODESET KEYADDR=OIEXUKEY,WORKREG=6 GET IN USER KEY
+* /* MACDATE Y-3 77277                                       @ZA26071*/
+* /*
+         IC    6,OIEXUKEY          GET KEY FROM SAVE LOCATION
+         SPKA  0(6)                SET PSW KEY
          STC   RODD,DCBBUFNO      PUT IN USER DCB
          MODESET EXTKEY=ZERO      BACK TO KEY ZERO
+* /* MACDATE Y-3 77277                                       @ZA26071*/
+* /*
+         SPKA  0(0)                SET PSW KEY
 RETBUFNO EQU   *                  RETURN FROM BUFNO
          BR    RET                RETURN
 INOUTBN  DC    A(INOUTBNO)        SYSIN/SYSOUT/DUMMY BUFNO DEFAULT
 ***********************************************************************
 *
 *        MAX TABLE FOR TAPE
 *
 ***********************************************************************
          DS    0F
          DC    CL8'TPMAX'         TPMAX ID
 TPMAX    DS    0F
 TPMAXK   DC    A(TPMXK)           MAXIMUM SIZE FOR BUFFERS IN UNITS    *
                                   OF 1024
 TPMAXBNO DC    A(TPMXBNO)         MAXIMUM NUMBER OF BUFFERS
 ***********************************************************************
 *
 *        MAX TABLE FOR DASD
 *
 ***********************************************************************
          DS    0F
          DC    CL8'DAMAX'         DAMAX ID
 DAMAX    DS    0F
 DAMAXK   DC    A(DAMXK)           MAXIMUM SIZE FOR BUFFERS IN UNITS    *
                                   OF 1024
 DAMAXBNO DC    A(DAMXBNO)         MAXIMUM NUMBER OF BUFFERS
```

```
SCREEN   EQU   *                  SCREEN OUT CASES WHERE RLSE, *
                                  AND SQTY SHOULD NOT EXECUTE
***********************************************************************
*        DO NOT PROCESS IF
*           SYSIN/SYSOUT/TERMINAL
*           DD DUMMY
*           USER ASKS JFCB NOT BE RE-WRITTEN
*           SYSTEM DATA SET ('SYS1.XXX')
*           NON-DASD UCB
*           NOT A FORMAT 1 DSCB
*           EXCP DCB
*           DSORG IN DCB IS NEITHER PS NOR PO
*           DSORG IN DSCB IS NEITHER PS NOR PO
*           NEITHER PUT NOR WRITE MACRO CODED IN DCB
*           OPEN FOR OTHER THAN OUTPUT OR OUTIN
***********************************************************************
         B     AFTRID3
         DC    CL8'SCREEN'        SCREEN ROUTINE ID
AFTRID3  L     RTIOT,OIEXTIOT     TIOT ENTRY ADDRESS
         TM    TIOELINK,TIOESSDS+TIOTTERM SYSIN/SYSOUT OR TERMINAL
         BNZ   EXIT               EXIT IF SYSIN OR SYSOUT OR   TERMINAL
         L     RJFCB,OIEXJFCB     JFCB ADDRESS
         CLC   JFCBDSNM(L'NULLFILE),NULLFILE DUMMY DATA SET
         BE    EXIT               EXIT IF DUMMY
         CLC   SYS1,JFCBDSNM      SYS1.XXX DATA SET
         BE    EXIT               EXIT IF SYSTEM DATA SET
         TM    JFCBTSDM,JFCNWRIT   DON'T MODIFY JFCB
         BO    EXIT               EXIT IF YES
         L     RUCB,OIEXUCB       UCB ADDRESS
         LTR   RUCB,RUCB          ANY UCB?
         BZ    EXIT               EXIT IF NO UCB
         TM    UCBTBYT3,UCB3DACC  DASD UCB?
         BNO   EXIT               EXIT IF NOT DASD
         L     RDSCB,OIEXDSCB     FORMAT 1 DSCB ADDRESS
         USING DS1FMTID,RDSCB
         CLI   DS1FMTID,C'1'      IS THIS A FORMAT 1 DSCB
         BNE   EXIT               EXIT IF NOT
         L     RDCB,OIEXPDCB      PROTECTED DCB ADDRESS
         TM    DCBMACF1,DCBMRECP  EXCP DCB?
         BO    EXIT               EXIT IF EXCP
         TM    DCBDSRG1,DCBDSGPS+DCBDSGPO PS OR PO DCB
         BZ    EXIT               EXIT IF NOT PS OR PO
         NC    DS1DSORG,DS1DSORG  IS DSORG SPECIFIED
         BZ    TSTMACRF           TRUST DCB IF NOT SPECIFIED
         TM    DS1DSORG,DS1DSGPS+DS1DSGPO IS DATA SET PS OR PO
         BZ    EXIT               EXIT IF NOT PS OR PO
TSTMACRF EQU   *                  TEST MACRF IN DCB
         TM    DCBMACF2,DCBMRPUT  PUT MACRO
         BO    TSTOOPT            TEST OPEN OPTION
         TM    DCBMACF2,DCBMRWRT  WRITE MACRO
         BZ    EXIT               EXIT IF NOT WRITE
TSTOOPT  EQU   *                  TEST OPEN OPTION
         TM    OIEXOOPT,OIEXOOUT  OPEN FOR OUTPUT
         BO    SCREENOK           BRANCH IF YES
         TM    OIEXOOPT,OIEXOOIN  OPEN FOR OUTIN
         BNO   EXIT               EXIT IF NO
SCREENOK EQU   *
         BR    RET                RETURN TO CALL RLSE, SQTY
```

```
RLSE     EQU   *                  SET PARTIAL RELEASE
***********************************************************************
*
*        DEFINE DEFAULT VALUES
*          RLSE0   = NUMBER OF EXTENTS. IF THE DATA SET HAS THIS
*                    NUMBER OF EXTENTS OR MORE, THEN PARTIAL RELEASE
*                    WILL NOT BE ALLOWED.
*          RLSE1   = NUMBER OF EXTENTS. IF THE DATA SET HAS LESS THAN
*                    THIS NUMBER OF EXTENTS, PARTIAL RELEASE IS
*                    REQUIRED.
*
*        NOTE THAT RLSE0 MUST NOT BE GREATER THAN RLSE1
*
*        SETTING RLSE0 TO 17 OR GREATER WILL CAUSE THIS ROUTINE TO
*          NEVER PREVENT A REQUEST FOR PARTIAL RELEASE
*
*        SETTING RLSE1 TO 0 WILL CAUSE THIS ROUTINE TO
*          NEVER FORCE A REQUEST FOR PARTIAL RELEASE
*
***********************************************************************
RLSE0    EQU   8                  SET RELEASE BIT TO ZERO IF NUMBER OF *
                                  EXTENTS EQUAL OR GREATER THAN THIS
RLSE1    EQU   8                  SET RELEASE BIT TO ONE IF NUMBER OF  *
                                  EXTENTS LESS THAN THIS
         B     AFTRID4
         DC    CL8'RLSE'          RLSE ROUTINE ID
AFTRID4  BCR   0,RET              NOP RETURN
         L     RDSCB,OIEXDSCB     FORMAT 1 DSCB ADDRESS
         TM    DS1DSORG,DS1DSGPO  IS DATA SET PARTITIONED
         BO    TSTRLSE            DO NOT SET RELEASE FOR PARTITIONED
         CLC   DS1NOEPV,EXTRLSE1  FEW ENOUGH TO SET RELEASE
         BNL   TSTRLSE            BRANCH IF NOT
         L     RJFCB,OIEXJFCB
         OI    JFCBIND1,JFCRLSE   SET RELEASE
         LA    RINCODE,MODJFCB    JFCB MODIFIED
         B     RETRLSE            RETURN
TSTRLSE  CLC   DS1NOEPV,EXTRLSE0  ENOUGH TO ZERO RELEASE
         BL    RETRLSE            BRANCH IF NO
         NI    JFCBIND1,255-JFCRLSE ZERO RELEASE
         LA    RINCODE,MODJFCB    JFCB MODIFIED
RETRLSE  EQU   *                  RETURN FROM RLSE
         BR    RET                RETURN
         DC    CL8'RLSECONS'      RLSE CONSTANTS ID
         DS    0H
EXTRLSE1 DC    AL1(RLSE1)         IF FEWER THAN THIS NUMBER OF EXTENTS,*
                                  PARTIAL RELEASE WILL BE SET
EXTRLSE0 DC    AL1(RLSE0)         IF THIS NUMBER OR MORE EXTENTS,      *
                                  PARTIAL RELEASE WILL BE ZEROED
```

```
SQTY     EQU   *                   SET SECONDARY QUANTITY
***********************************************************************
*
*        DEFINE DEFAULT VALUES
*          SQTYDFL = DEFAULT SECONDARY QUANTITY. THIS QUANTITY IS
*                    SET IF THE SECONDARY QUANTITY IS ZERO AND THE
*                    PRIMARY QUANTITY IS ZERO OR ONE. IT IS USED
*                    IF SECONDARY QUANTITY IS ONE, AND THE NUMBER OF
*                    EXTENTS IS EQUAL OR GREATER TO EXTSQT.
*          EXTSQT  = NUMBER OF EXTENTS. IF THE DATA SET HAS THIS MANY
*                    EXTENTS OR MORE, THEN INCREASE SECONDARY QUANTITY.
*
***********************************************************************
SQTYDFL  EQU   5                   DEFAULT SECONDARY QUANTITY
EXTSQT   EQU   10                  IF DATA SET HAS THIS MANY EXTENTS,   *
                                   THEN INCREASE SECONDARY QUANTITY
         B     AFTRID6
         DC    CL8'SQTY'           SQTY ROUTINE ID
AFTRID6  BCR   0,RET               NOP RETURN
         L     RJFCB,OIEXJFCB      JFCB ADDRESS
         NC    JFCBSQTY,JFCBSQTY   ANY SECONDARY QUANTITY
         BZ    TSTPRIM             TEST PRIMARY IF NOT
         L     RDSCB,OIEXDSCB      FORMAT 1 DSCB ADDRESS
         CLC   DS1NOEPV,EXTSQTY    ENOUGH TO ADD TO SECONDARY QTY
         BL    RETSQTY             BRANCH IF NOT
         XR    RODD,RODD
         ICM   RODD,7,JFCBSQTY     GET SECONDARY QUANTITY
         LR    REVEN,RODD          SAVE IN REVEN
         SRL   REVEN,1             HALVE SECONDARY QUANTITY
         LTR   REVEN,REVEN         IS SECONDARY ONE
         BZ    SETDFLT             DEFAULT SECONDARY IF ONE
         AR    RODD,REVEN          150% OF SECONDARY
         B     STSQTY
TSTPRIM  EQU   *                   SECONDARY QUANTITY IS ZERO
         NC    JFCBPQTY,JFCBPQTY   IS PRIMARY QUANTITY ZERO
         BZ    DFLTSQTY            DEFAULT SECONDARY
         XR    RODD,RODD
         ICM   RODD,7,JFCBPQTY
         SRL   RODD,1              HALVE PRIMARY
         LTR   RODD,RODD           IS PRIMARY ONE
         BNZ   STSQTY              BRANCH IF NOT
SETDFLT  EQU   *                   USE QUANTITY IN SQTYDFLT
         L     RODD,SQTYDFLT       DEFAULT SECONDARY
         B     STSQTY              STORE SECONDARY
DFLTSQTY EQU   *                   PRIMARY AND SECONDARY ZERO
         L     RODD,SQTYDFLT       GET DEFAULT SECONDARY
         TM    JFCBCTRI,JFCBSPAC
         BNZ   STSQTY
         CLI   DS1EXT1,X'01'       TRACK EXTENT
         BE    DFLTTRK             YES -- SET TRACKS
         CLI   DS1EXT1,X'81'       CYL EXTENT
         BNE   RETSQTY             NO -- RETURN
         OI    JFCBCTRI,JFCBCYL    SET CYLINDER UNITS
         B     STSQTY
DFLTTRK  EQU   *                   SET TRACK UNITS
         OI    JFCBCTRI,JFCBTRK    MAKE TRACK REQUEST
STSQTY   EQU   *                   STORE SECONDARY QTY
         STCM  RODD,7,JFCBSQTY
         LA    RINCODE,MODJFCB     JFCB MODIFIED
RETSQTY  EQU   *                   RETURN FROM SQTY
         BR    RET                 RETURN
         DS    0F
         DC    CL8'SQTYCONS'       SQTY ROUTINE CONSTANTS ID
SQTYDFLT DC    A(SQTYDFL)          DEFAULT SECONDARY QUANTITY
         DC    AL1(0)              NOTE ONE BYTE OF ZERO BEFORE EXTSQTY
EXTSQTY  DC    AL1(EXTSQT)         IF DATA SET HAS THIS MANY EXTENTS,   *
                                   THEN ADD TO SECONDARY QUANTITY
```

```
************************************************************************
*
*        CONSTANTS / PATCH AREA
*
************************************************************************
NULLFILE DC    C'NULLFILE '       DD DUMMY DATA SET NAME
SYS1     DC    C'SYS1.'           START OF SYSTEM DATA SET NAMES
         DS    0F
PATCH    DC    C'IFG0EX0B PATCH AREA '
         DC    XL50'00'
************************************************************************
*
*        MAX TABLE MAPPING DSECT (MAPS TPMAX OR DAMAX)
*
************************************************************************
MAX      DSECT
MAXBUF   DS    A                  MAXIMUM SIZE FOR BUFFERS
MAXBNO   DS    A                  MAXIMUM NUMBER OF BUFFERS
************************************************************************
*
*        DCB OPEN INSTALLATION EXIT PARAMETER LIST
*        - THE IECOIEXL MACRO IS IN SYS1.MACLIB
*
************************************************************************
         IECOIEXL
******** THE MACRO EXPANSION IS NOT SHOWN
************************************************************************
*
*        DCB - THE IHADCB MACRO IS IN SYS1.MACLIB
*
************************************************************************
         IHADCB DSORG=PS,DEVD=DA
******** THE MACRO EXPANSION IS NOT SHOWN
************************************************************************
*
*        UCB - THE IEFUCBOB MACRO IS IN SYS1.AMODGEN
*
************************************************************************
UCB      DSECT
         IEFUCBOB LIST=YES
******** THE MACRO EXPANSION IS NOT SHOWN
************************************************************************
*
*        TIOT - THE IEFTIOT1 MACRO IS IN SYS1.AMODGEN
*
************************************************************************
TIOT     DSECT
         IEFTIOT1
******** THE MACRO EXPANSION IS NOT SHOWN
************************************************************************
*
*        JFCB - THE IEFJFCBN MACRO IS IN SYS1.AMODGEN
*
************************************************************************
JFCB     DSECT
         IEFJFCBN LIST=YES
******** THE MACRO EXPANSION IS NOT SHOWN
************************************************************************
*
*        FORMAT 1 DSCB - THE IECSDSL1 MACRO IS IN SYS1.AMODGEN
*
************************************************************************
F1DSCB   DSECT
         IECSDSL1 (1)
******** THE MACRO EXPANSION IS NOT SHOWN
         END
```

INDEX

GC26-4056-1

MVS/370 System Programming Library: Data Management (File No. S370-30) Printed in U.S.A. GC26-4056-1

MVS/370 System Programming Library:
Data Management
GC26-4056-1

Reader's Comment Form

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

List TNLs here:

If you have applied any technical newsletters (TNLs) to this book, please list them here:

Last TNL ____________________

Previous TNL ____________________

Previous TNL ____________________

Fold on two lines, tape, and mail. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.) Thank you for your cooperation.

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

GC26-4056-1

Reader's Comment Form

Fold and tape | Please do not staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
P.O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and tape | Please do not staple | Fold and tape

IBM®

MVS/370 System Programming Library: Data Management (File No. S370-30) Printed in U.S.A. GC26-4056-1