GC26-3830-4
File No. S370-30

Systems

OS/VS2 System Programming Library: Data Management

Release 3.8

IBM

This publication was produced using the
IBM Document Composition Facility
(program number 5748-XX9) and
the master was printed on the IBM 3800 Printing Subsystem.

Fifth Edition (October 1981)

This is a major revision of, and makes obsolete, GC26-3830-3, its technical newsletters GN26-0942, GN26-0945, GN26-0950, GN26-0983, GN26-0986, and GN26-0997, and the System Library Supplement, GD26-6017-0.

This edition applies to Release 3.8 of OS/VS2 MVS 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 republication 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 Bibilography, GC20-0001, for the editions that are applicable and current.

It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

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 reader's 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 any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

PREFACE

This publication provides information on how to modify and extend the data management capabilities of the MVS system control program; the intended audience is system programmers.

Some topics included are:

- Using catalog management macro instructions
- Maintaining the volume table of contents
- Executing your own channel programs
- Using XDAP to read and write data sets on direct-access devices
- Password-protecting your data sets

The MVS system control program provides simpler ways (for example, access-method services, job control language, utility programs, access-method routines) to do each of these things. The information presented in this book (consisting of macro specifications and how-to information) is intended to provide greater flexibility in using the data management capabilities of MVS.

Other topics presented are:

- Using system macro instructions to refer to, validate, and modify system data areas. Other system macro instructions described allow you to obtain device characteristics, modify a job file control block, protect data by verifying the data extent block, stop the processing of I/O requests, and perform track capacity calculations.
- Adding to the image library and retrieving FCB images
- JES2 printer support
- Perform special processing before or after certain macro instructions
- How storage management routines control space on direct-access volumes

PREREQUISITE READING

Readers are expected to understand how to:

- Code programs in assembler language as described OS/VS-DOS/VS-VM/370 Assembler Language, GC33-4010.
- Use the standard linkage conventions as described in OS/VS2 Supervisor Services and Macro Instructions, GC28-0683.
- Maintain the catalog and VTOC as described in OS/VS2 Access Method Services, GC26-3841, OS/VS2 MVS Utilities, GC26-3902, and OS/VS2 MVS Data Management Services Guide, GC26-3875.
- Use the access methods to do input/output using the data management macros as described in OS/VS2 MVS Data Management Services Guide, GC26-3875, and OS/VS2 MVS Data Management Macro Instructions, GC26-3873.
- Protect data sets as described under "IEHPROGM" in OS/VS2 MVS Utilities, GC26-3902.

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

RELATED READING

All of the chapters of this publication refer to OS/VS2 System Programming Library: Debugging Handbook, Volume 1, GC28-0708, OS/VS2 System Programming Library: Debugging Handbook, Volume 2, GC28-0709, and OS/VS System Programming Library: Debugging Handbook, Volume 3, GC28-0710, which contain detailed descriptions of system control blocks and common work areas. More specific related reading is listed at the beginning of each chapter, as it relates to the topic under discussion.

Other publications referenced in this manual are:

- IBM System/370 Principles of Operation, GA22-7000
- IBM 2821 Control Unit Component Description, GA24-3312
- IBM 3203 Printer Component Description and Operator's Guide, GA33-1515
- 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, GC26-3846
- MVS/System Product Release 2 Installation, Initialization, and Tuning: JES2 Component, SC23-0046
- OS/VS2 MVS Checkpoint/Restart, GC26-3877
- OS/VS2 DADSM Logic, SY26-3828
- Data Facility Device Support: User's Guide and Reference, SC26-3952
- Device Support Facilities, GC35-0033
- OS/VS2 I/O Supervisor Logic, SY26-3823
- OS/VS2 JCL, GC28-0692
- OS/VS Message Library: VS2 System Messages, GC38-1002
- OS/VS2 MVS CVOL Processor, GC26-3864
- OS/VS2 MVS Resource Access Control Facility (RACF): General Information Manual, GC28-0722
- OS/VS2 Open/Close/EOV Logic, SY26-3827
- OS/VS2 System Programming Library: Initialization and Tuning Guide, GC28-0681
- OS/VS2 MVS System Programming Library: JES2, GC23-0002
- OS/VS2 System Programming Library: JES3, GC28-0608
- System Programming Library: Network Job Entry Facility for JES2, SC23-0003
- OS/VS2 System Programming Library: Supervisor, GC28-0628
- OS/VS2 System Programming Library: System Generation Reference, GC26-3792
- OS/VS Tape Labels, GC26-3795
- OS/VS2 TSO Command Language Reference, GC28-0646

- OS/VS Virtual Storage Access Method (VSAM) Programmer's Guide, GC26-3838

- OS/VS Linkage Editor and Loader, GC26-3813

HOW TO USE THIS BOOK

You can use the chapter on catalog management macro instructions to retrieve catalog information or add, delete, and update catalog entries for non-VSAM data sets.

If you want to read a data set control block, rename a data set, or delete a data set using the system macros, the chapter on maintaining the volume table of contents (VTOC) provides macro specifications, coding examples, and how-to information.

If you want to code your own channel programs to modify the control program or to provide support for unsupported I/O devices, the chapter on using EXCP provides detailed descriptions of the control blocks you must provide and the functions you must perform.

Macro specifications and how-to information are provided for using the XDAP macro instruction to read from and write to direct-access devices without using access-method routines (SAM, ISAM, or BDAM).

If you want to use data set protection for your facility, the chapter on data set protection:

- Tells how to build a PASSWORD data set.

- Describes how the system control program responds to job control language and IEHPROGM utility statements in maintaining the PASSWORD data set.

- Tells you how to use the PROTECT macro instruction to maintain (add records to, delete records from, changes records in) and read the PASSWORD data set.

The chapter on system macro instructions provides how-to information and macro specifications for:

- Using system mapping macros to allow you to access system control blocks and work areas using symbolic names.

- Examining device-type information in unit control blocks (UCBs).

- Modifying a job file control block (JFCB) before opening a data set.

- Stopping the processing of specified I/O requests, permanently or temporarily.

- Protecting your data sets by verifying data extent blocks.

- Performing track capacity calculations.

You can use the coding examples and how-to information in the chapter on adding and retrieving from the image library to help you add a universal character set (UCS) image or a forms control buffer (FCB) image to the system image library (SYS1.IMAGELIB).

Other topics presented are:

- 1403, 3203-5, and 3211 printers JES2 support

- OPEN and EOV macro user exits form when a format-1 DSCB cannot be found

- How you can perform special processing before and after the CATALOG, SCRATCH, and RENAME macros by replacing the supplied dummy modules

- How the direct access device storage management (DADSM) routines control space on DASD volumes.

In this manual, any references made to an IBM program product are not intended to state or imply that IBM's program product only may be used; any functionally equivalent program may be used instead. This manual has references to the following IBM program product:

RACF - Resource Access Control Facility,
       Program Number 5740-XXH

SUMMARY OF AMENDMENTS

SAM-E ENHANCEMENTS

The section "Specifying Buffer Numbers for SAM-E DASD Data Sets" has been added.

DATA FACILITY DEVICE SUPPORT—3375 SUPPORT

The information received from issuing the DEVTYPE macro for the IBM 3375 has been included.

OTHER CHANGES

The section "Controlling Space on DASD Volumes" has been added. This information was previously contained in OS/VS2 DADSM Logic.

The explanation of return code 8 from the TRKCALC macro has been updated.

OS/VS2 MVS DATA FACILITY DEVICE SUPPORT ENHANCEMENTS

Information to support the above has been added to the section "Initiation of the Channel Program."

OS/VS2 MVS DATA FACILITY DEVICE SUPPORT (DFDS) PROGRAM PRODUCT

The information to support CVAF (Common VTOC Access Facility), and the IBM 3375 and 3380 Disk Storage is included. For more information see, Data Facility Device Support: User's Guide and Reference, SC26-3952, Introduction to 3375 Direct Access Storage, GA26-1666, and Introduction to 3380 Direct Access Storage, GA26-1662.

OS/VS2 MVS 3800 ENHANCEMENTS

IEBIMAGE can now be used to build library character set modules to be stored in SYS1.IMAGELIB.

NEW PROGRAMMING SUPPORT

The information to support the IBM 3203 model 5 is now included. For additional information about the IBM 3203 Printer, see IBM 3203 Printer Component Description and Operator's Guide, GA33-1515.

The figure "Output Obtained from Issuing DEVTYPE Macro" now includes the 3203 Printer.

The sections "Adding to the Image Library and Retrieving FCB Images," "Adding a UCS Image to the Image Library," and "Adding an FCB Image to the Image Library" have been updated to include the 3203 Printer.

The figure "Sample Code to Add a 3203 UCS Image to SYS1.IMAGELIB" has been added.

SERVICE CHANGES

The following sections have been updated:

- Catalog Order of Search
- Retrieving Information from a VS2 Catalog
- Retrieving Information by Data Set Name (LOCATE and CAMLST NAME)
- Retrieving Information by Generation Data Set Name (LOCATE and CAMLST NAME)
- Deleting a Data Set (SCRATCH and CAMLST SCRATCH)
- Renaming a Data Set (RENAME and CAMLST RENAME)
- Interruption Handling and Error Recovery Procedures
- Start I/O (SIO) Appendage
- Program Controlled Interruption (PCI) Appendage
- ATLAS-Assigning an Alternate Track and Copying Data from the Defective Track
- Executing Fixed Channel Programs in Real Storage (EXCPVR)
- Page Fix List Processing
- Mapping System Data Areas
- IEFUCBOB-Mapping the UCB
- IEFJFCBN-Mapping the JFCB
- CVT-Mapping the CVT
- DEVTYPE Macro Specification
- Reading and Modifying a Job File Control Block
- Adding a UCS Image to the Image Library
- The figures:
  - Sample Code to Add a 1403 UCS Image to SYS1.IMAGELIB
  - Sample Code to Add a 3211 UCS Image to SYS1.IMAGELIB
- Adding an FCB Image to the Image Library
- The figures:
  - Sample of the Standard FCB Image STD1
  - Sample of the Standard FCB Image STD2
  - Sample Code to Assemble and Add an FCB Load Module to SYS1.IMAGELIB

Two new figures have been added:

- Generation Index Pointer Entry
- Alias Entry

OS/VS MVS DATA MANAGEMENT SUPPORT FOR MASS STORAGE SYSTEM (MSS) EXTENSIONS PROGRAM PRODUCT

The MSS Extensions Program Product is supported by the addition of the section "Scratch Dummy Module."

CONTENTS

FIGURES

USING CATALOG MANAGEMENT MACRO INSTRUCTIONS

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

- Retrieve information from an MVS catalog. Three kinds of catalogs qualify as MVS catalogs: the MVS master catalog, VSAM user catalogs, and OS CVOLs (control volumes).
- Catalog non-VSAM data sets.
- Uncatalog non-VSAM data sets.
- Recatalog non-VSAM data sets.
- Read a block from a CVOL.
- Build an index in a CVOL.
- Build a generation index in a CVOL.
- Delete an index.
- Assign an alias to a high-level index in a CVOL.
- Delete an index alias from a CVOL.
- Connect two CVOLs.
- Disconnect two CVOLs.

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

- OS/VS-DOS/VS-VM/370 Assembler Language, which contains information you will need to code programs in the assembler language.
- OS/VS2 Access Method Services, which tells how to use programs that offer some of the same services as catalog management macros plus additional services that catalog management macros cannot provide.
- OS/VS2 JCL, which tells how to catalog and uncatalog data sets using job control language statements.
- OS/VS2 MVS CVOL Processor, which tells how to use CVOLs in the MVS environment.

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

A CVOL is identified in the master catalog as a non-VSAM data set with a name of the form SYSCTLG.yyy where yyy is any unique string, unless the CAMLST CVOL operand is to be specified in CVOL requests. If CVOL is specified, the CVOL must be defined in the MVS master catalog as a non-VSAM data set with the name SYSCTLG.Vyyyyyy where yyyyyy is the volume serial number of the CVOL. The high level data set name is defined as an alias of the CVOL entry.

The volume serial of a CVOL may be specified as the third operand of the CAMLST macro (described later in this section). If this operand is specified, searching begins directly with the specified CVOL. Searching may continue on the other CVOLs if these CVOLs have been connected with the CAMLST LNKX macro with the high-level qualifier of the data set name. Searching will never go to any VSAM format catalogs when the CVOL volume operand is specified.

If the CVOL volume operand is not specified, searching begins in the JOBCAT or STEPCAT catalog if specified. If not found or no JOBCAT/STEPCAT Catalog was specified, searching continues in the master catalog.

In the master catalog, a search is made to determine if the first qualifier of the data set name is the name of a (VSAM) user catalog or the alias of a private catalog (either a VSAM user catalog or a CVOL).

If the first qualifier is the name or alias of a private catalog, the search continues in the private catalog. Otherwise, the process terminates in the master catalog.

For information about how CVOLs are defined, identified, and searched, see OS/VS2 MVS CVOL Processor.

RETURN CODE CONSIDERATIONS

The interpretation of catalog management return codes depends on whether the request is initiated using a CAMLST or a VSAM parameter list, and whether the request is satisfied in a VSAM catalog or a CVOL.

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

If a VSAM parameter list is used and the request is satisfied in a CVOL, register 15 contains the VSAM return code, register 0 is not meaningful, and register 1 is nonzero. If a VSAM parameter list is used and the request is satisfied in a VSAM catalog, registers 15 and 0 contain a VSAM return code and a VSAM reason code respectively. These codes are explained in OS/VS Message Library: VS2 System Messages under message IDC009I.

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

RETRIEVING INFORMATION FROM AN MVS CATALOG

To read an entry from an MVS 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 a 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 a CVOL, the address of a volume control block will follow the volume list entries.

If you specify a partially qualified data set name, the first block in the CVOL catalog 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 parameter in the CAMLST macro. Register 15 will contain the return code 12. If you specify a single qualifier and do not include the CVOL parameter, the CVOL identifier 'SYSCTLG.Vyyyyyy' is read into your work area. You may then insert 'yyyyyy' as the CVOL 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.

For MVS, you must add a step when specifying either an unqualified name or the highest level of a partially qualified name to retrieve information from a CVOL. Because the search of the VSAM master catalog is different from the search for the OS/VS1 and OS/VS2 Release 1 system catalog, you do not receive the volume information or the index block from the CVOL as you might expect. You receive, instead, the volume information for the CVOL that is found in the VSAM 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 CVOL volume serial number in your CAMLST so that the search starts in the 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 OS/VS2 System Programming Library: 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 a 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 format of the parameter list of this macro is described in OS/VS2 System Programming Library: Debugging Handbook in the section "SVC Summary."

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 an MVS catalog by name.

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 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 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" earlier in this section.

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.

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
           Check Returns
*                                           READ CATALOG ENTRY
INDAB       CAMLST    NAME,AB,,LOCAREA      FOR DATA SET A.B.
AB          DC        CL44'A.B'             INTO VIRTUAL STORAGE
LOCAREA     DS        0D                    AREA NAMED LOCAREA.
            DS        265C                  LOCAREA MAY ALSO
*                                           CONTAIN A 3-BYTE
*                                           TTR AND THE 6-BYTE
                                            CVOL SERIAL
```

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

8 One of the following happened:

- The entry was not found. Register 0 contains the number of valid index levels if in a CVOL. Register 0 contains the VSAM catalog return code if in a VSAM catalog.
- The user is not authorized to perform this operation. Register 0 contains 56 (decimal).
- A GDG alias was found (VSAM catalog only). Register 0 contains the number of valid index levels. The alias name was replaced by the true name.

12 One of the following happened:

- An index or generation 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 by the true name.
- An invalid low level GDG name was found.

16 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 A syntax error exists in the name.

24 One of the following happened:

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

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

32 Reserved.

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 of the parameter list of this macro is described in OS/VS2 System Programming Library: Debugging Handbook in the section "SVC Summary."

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 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" earlier in this section.

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
        Check Returns

INDGX    CAMLST    NAME,APAY,,LOCAREA     DATA SET A.PAY (-3)'
APAY     DC        CL44'A.PAY(-3)'        INTO YOUR STORAGE
LOCAREA  DS        0D                     AREA NAMED LOCAREA.
         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)."

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 (GDGs) cataloged in CVOLs.

The format of the parameter list of this macro is described in OS/VS2 System Programming Library: Debugging Handbook in the section "SVC Summary."

The format is:

[symbol]	**LOCATE**	list-addrx
listname	**CAMLST**	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 an MVS 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 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" earlier in this section.

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 VS2 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
        Check Returns

INDAB    CAMLST    NAME,ABC,,LOCAREA        DATA SET X.B.C INTO
ABC      DC        CL44'X.B.C.'             VIRTUAL STORAGE AREA
LOCAREA  DS        0C                       NAMED LOCAREA.
         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 the section "Retrieving Information by Data Set Name (LOCATE and CAMLST NAME)."

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

You can read any block in a CVOL catalog 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 of the parameter list of this macro is described in OS/VS2 System Programming Library: Debugging Handbook in the section "SVC Summary."

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 Exceptional Returns

BLK       CAMLST    BLOCK,TTR,VOLSER,LOCAREA
*                                       READ A BLOCK INTO
*                                       VIRTUAL STORAGE. AREA
TTR       DC        H'5'                NAMED LOCAREA
          DC        X'03'               RELATIVE TRACK 5
VOLSER    DC        C"111111'           BLOCK 3 ON TRACK
LOCAREA   DS        0D                  VOLUME SERIAL OF CVOL
          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 in "Retrieving Information by Data Set Name (LOCATE and CAMLST NAME)."

BUILDING AND DELETING INDEXES

You handle 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 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 of the parameter list of this macro is described in OS/VS2 System Programming Library: Debugging Handbook in the section "SVC Summary."

The format is:

[symbol] listname	**INDEX** **CAMLST**	list-addrx **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 blanks. 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 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" earlier in this section.

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

```
          INDEX    INDEXA          BUILD INDEX A

       Check Exceptional Returns

          INDEX    INDEXB          BUILD INDEX STRUCTURE
*                                  A.B

       Check Exceptional Returns

          INDEX    INDEXC          BUILD INDEX STRUCTURE
*                                  A.B.C

       Check Exceptional Returns

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
```

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.

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 Interpretation

4 The CVOL does not exist or cannot be opened.

8 One of the following happened:

- The existing catalog structure is inconsistent with the operation requested. If the error was detected while processing in a 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 master catalog processing, register 0 contains the VSAM 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 An attempt was made to delete an index or generation index that has an alias or has indexes or data sets cataloged under it. The index is unchanged.

16 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 Space is not available on the specified control volume.

24 Not used with the INDEX macro instruction.

28 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 a 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 of the parameter list of this macro is described in OS/VS2 System Programming Library: Debugging Handbook in the section "SVC Summary."

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, it must be followed by blanks. 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 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" earlier in this section.

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 control volume, 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 data set has been successfully uncataloged and the DELETE operand has been specified, the catalog management routines issue a SCRATCH macro (see "Maintaining the Volume Table of Contents") 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 Exceptional Returns

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, operand of CAMLST, specifies that a generation index be built. DLEVEL specifies the virtual storage location of an area into which you have placed the fully qualified name of a generation index. VOLNUM 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 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)."

DELETING AN INDEX (INDEX AND CAMLST DLTX)

You can delete any number of index levels from an existing 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 of the parameter list of this macro is described in OS/VS2 System Programming Library: Debugging Handbook in the section "SVC Summary."

The format is:

[symbol]	**INDEX**	list-addrx
listname	**CAMLST**	**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.

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 blanks. 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 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" earlier in this section.

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 Exceptional Returns

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 in "Building an Index (INDEX and CAMLST BLDX)."

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

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 of the parameter list of this macro is described in OS/VS2 System Programming Library: Debugging Handbook in the section "SVC Summary."

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 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" earlier in this section.

alias name-relexp
: 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, index level A is assigned an alias of X.

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

       Check Exceptional Returns

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 in "Building an Index (INDEX and CAMLST BLDX)."

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

You can delete an alias previously assigned to a high level index by using the INDEX and CAMLST macro instructions.

The format of the parameter list of this macro is described in OS/VS2 System Programming Library: Debugging Handbook in the section "SVC Summary."

The format is:

[symbol]	**INDEX**	list-addrx
listname	**CAMLST**	**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 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.

cvol-relexp
: specifies the virtual storage location of a 6-byte volume serial number of the 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" earlier in this section.

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 Exceptional Returns

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 CONTROL VOLUMES

You connect and disconnect control volumes by using combinations of the INDEX and CAMLST macro instructions.

CONNECTING CONTROL VOLUMES (INDEX AND CAMLST LNKX)

You connect two control volumes (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 a 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 control volumes is that the volume serial number of the control volume 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 of the parameter list of this macro is described in OS/VS2 System Programming Library: Debugging Handbook in the section "SVC Summary."

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 name relexp
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 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" earlier in this section.

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 control volume.

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

```
              INDEX     CONNECT              CONNECT TWO CONTROL
*                                            VOLUMES

         Check Exceptional Returns

CONNECT       CAMLST    LNKS,INDXNAME,OLDCVOL
*                                            WHOSE SERIAL NUMBERS
INDXNAME      DC        CL8'HIGHINDX'        ARE 000155 AND 001555.
OLDCVOL       DC        CL6'000155'
NEWCVOL       DC        X'30C02008'          2314 DISK DEVICE CODE
              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 binary 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 control volumes have been successfully connected, register 15 will contain zeros. Otherwise, register 15 will contain one of the exception return codes described in the section "Building an Index (INDEX and CAMLST BLDX)."

DISCONNECTING CONTROL VOLUMES (INDEX AND CAMLST DRPX)

You disconnect two control volumes by using the INDEX and CAMLST macro instructions.

The result of disconnecting control volumes is that the control volume pointer is removed from the volume index of the volume from which you are disconnecting.

The format of the parameter list of this macro is described in OS/VS2 System Programming Library: Debugging Handbook in the section "SVC Summary."

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 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" earlier in this section.

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

```
         INDEX    DISCNECT              DISCONNECT TWO
*                                       CONTROL VOLUMES
      Check Exceptional Returns

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 control volumes be disconnected. INDEXNAME, 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 control volume to be disconnected.

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

WORKING WITH NONVSAM DATA SET CATALOG ENTRIES

You can catalog, uncatalog, and recatalog non-VSAM data sets 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.

CATALOGING A NONVSAM DATA SET (CATALOG AND CAMLST CAT)

The format of the parameter list of this macro is described in OS/VS2 System Programming Library: Debugging Handbook in the section "SVC Summary."

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 a 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 blanks. 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 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" earlier in this section.

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 OS/VS2 System Programming Library: 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 re-catalogs 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 Returns

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'30C02008'     2314 DISK DEVICE CODE
          DC         CL6'000014'     VOLUME SERIAL NUMBER
          DC         H'0'            DATA SET SEQUENCE NUMBER
          DC         X'30C02008'     2314 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 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 Either the required catalog does not exist, it is not open, or the "do not allocate" bit is on.

8 One of the following happened:

- The existing catalog structure is inconsistent with the operation requested. If the error was detected while processing in a 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 a VSAM catalog, register 0 contains the VSAM catalog return code and register 1 contains zero.

- The user is not authorized to perform the operation. Register 0 contains decimal 56 and register 1 contains zero.

12 Not used with the CATALOG macro instruction.

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

20 There is insufficient space on the catalog data set.

24 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 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 a CVOL.
- There was a nonzero return code from ESTAE.

UNCATALOGING A NONVSAM 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 parameter list of this macro is described in OS/VS2 System Programming Library: Debugging Handbook in the section "SVC Summary."

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 blanks. 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 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" earlier in this section.

In the following example, the catalog entry for data set A.B.C is removed from an MVS catalog. In a 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 Returns

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 specifies that references to a data set be removed from the catalog. DSNAME specifies the virtual storage location of the 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 in the section "Cataloging a NonVSAM Data Set (CATALOG and CAMLST CAT)."

RECATALOGING A NONVSAM 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 OS/VS2 System Programming Library: Debugging Handbook.)

The format of the parameter list of this macro is described in OS/VS2 System Programming Library: Debugging Handbook in the section "SVC Summary."

The format of the CATALOG and CAMLST macros is:

[symbol]	CATALOG	list-addrx
listname	CAMLST	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 blanks. 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 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" earlier in this section.

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 Returns

RECATLG   CAMLST    RECAT,DSNAME,,VOLUMES
DSNAME    DC        CL6'A.B.C'             POINTER TO THE VOLUME
*                                          LIST.
VOLUMES   DC        H'3'                   FOR DELIMITER ONE BLANK
*                                          THREE VOLUMES.
          DC        X'30C02008'            2314 DISK DEVICE CODE
          DC        CL6'000014'            VOLUME SERIAL NUMBER
          DC        H'0'                   SEQUENCE NUMBER
          DC        X'30C02008'            2314 DISK DEVICE CODE
          DC        CL6'000015'            VOLUME SERIAL NUMBER
          DC        H'0'                   SEQUENCE NUMBER
          DC        X'30C02008'            2314 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 in the section "Cataloging a NonVSAM Data Set (CATALOG and CAMLST CAT)."

CVOL CATALOG ENTRY FORMATS

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

VOLUME INDEX CONTROL ENTRY

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

0 8 11 12

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

12 15 16 19 20

<——————Total Length: 22 Bytes——————>

Field 1: Name (8 bytes)—contains only a binary one 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 binary five to indicate that five 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 Volume Index Control Entry

INDEX CONTROL ENTRY

Field 1	Field 2	Field 3	Field 4	Field 5	Field 6
X'00000000000000001' Name	TTR of last block in this index	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 quite 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 binary one 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 binary three 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 index. If the index is not a high level index, this field is zero.

Field 6: Unused (2 bytes)

Figure 2. The Index Control Entry

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)	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	00 Count

0 8 11

<——Total Length: 12 Bytes——>

The index link and index pointer entries are quite 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 Index Link and Index Pointer Entries

DATA SET POINTER ENTRY

Field 1	Field 2	Field 3	Field 4
Lowest level name of data set or complemented generation number (if part 'of GDG)	DSCB TTR or zeros	Count	Volume count

0 8 11 12 14

Field 5	Field 6	Field 7
Device Code	Serial number of volume on which data set resides	Data set sequence number (zero for direct access

14 18 24

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 five 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 G<u>xxxx</u>V00, where <u>xxxx</u> 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: Volume 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 Data Set Pointer Entry

VOLUME CONTROL BLOCK POINTER ENTRY

Field 1	Field 2	Field 3	Field 4
Lowest level of data set name	TTR of volume control block	01 Count	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 binary one to indicate that one halfword follows.

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

Figure 5. The 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	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 one to twenty 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 Volume Control Block

CONTROL VOLUME (CVOL) POINTER ENTRY

Field 1	Field 2	Field 3
Name of index on other control volume	Dummy pointer field: zeros	05 Count

0 8 11 12

Field 4	Field 5
Device code of control volume	Serial number of control volume

12 16

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

Note: Prior to Release 17, the control volume pointer entry contained a count of 03 and did not have a device code field (field 4).

The CVOL pointer entry is used to indicate that a particular index resides on a volume other than the system residence volume. Control volume 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 control volume 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 number 5 to indicate that five halfwords follow.

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

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

Figure 7. The Control Volume (CVOL) Pointer Entry

CONTROL VOLUME POINTER ENTRY (OLD)

Until Release 17 of OS MFT/MVT, the control volume pointer entry was the same as the present control volume pointer, except that there was no field 4 (device code). The old CVOL pointer entry was 18 bytes long; after Release 17, it is 22 bytes long. Since some control volumes may still contain entries in the old format, and since the catalog management routines still check for it, it is mentioned here.

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 x 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. Generation Index Pointer Entry

ALIAS NAME

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

0 8 11 12

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

An alias entry defines an alternate 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: Halfword 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. Alias Name

MAINTAINING THE VOLUME TABLE OF CONTENTS

This chapter contains information on how to read and change the volume table of contents (VTOC) used on direct-access storage device volumes. The information consists of how-to information, macro specifications, and coding examples for the OBTAIN, SCRATCH, and RENAME macro instructions.

More detailed information about how the routines called by these macros work is available in OS/VS2 DADSM Logic.

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

- OS/VS-DOS/VS-VM/370 Assembler Language, which contains information you will need in order to code programs in the assembler language.
- OS/VS2 MVS Data Management Services Guide, contains a general description of direct-access device characteristics and the volume table of contents.
- OS/VS2 MVS Utilities, tells how to use utility programs to maintain the volume table of contents.
- OS/VS2 System Programming Library: Debugging Handbook, which contains descriptions of (1) the data set control block (DSCB) formats and (2) the contents of the fields of each DSCB.
- Data Facility Device Support: User's Guide and Reference contains information about VTOC indexes.

INTRODUCTION

In the same way that the catalog management routines keep track of cataloged data sets, the direct-access device space management (DADSM) routines maintain the volume table of contents (VTOC) on direct-access storage devices. This chapter tells how to use the OBTAIN, SCRATCH, and RENAME macro instructions. These macros are most commonly used by the system control program 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 second example). (DSCB formats and field descriptions are contained in OS/VS2 System Programming Library: Debugging Handbook.)

You can delete a 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 identifier (format-1) DSCB for the data set to be replaced with 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 of the parameter list of this macro is described in OS/VS2 System Programming Library: Debugging Handbook in the section "SVC Summary."

The format is:

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

list-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' 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 Returns

```
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	The required volume was not mounted.
12	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	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 of the parameter list of this macro is described in OS/VS2 System Programming Library: Debugging Handbook in the section "SVC Summary."

The format is:

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

list-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 Returns

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	The required volume was not mounted.
8	The format-1 DSCB was not found in the VTOC of the specified volume.
12	A permanent I/O error was encountered, or an invalid format-4 DSCB was found when processing the specified volume, or an unexpected error return code was received from CVAF (Common VTOC Access Facility).
16	Invalid work area pointer.
20	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 data set stored on direct-access volumes 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.

Refer to OS/VS2 MVS Resource Access Control Facility (RACF): General Information Manual for information on RACF-defined data sets. 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).

Refer to Data Facility Device Support: User's Guide and Reference for information on scratching a VTOC index data set.

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 OS/VS2 System Programming Library: Debugging Handbook.

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 him to mount the volume. (A volume mount message will not be issued for an MSS virtual 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 SCRATCH macro instruction is issued.

If the operator cannot mount the requested volume, he issues a reply indicating that he cannot fulfill the request. A condition 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	list-addrx SCRATCH ,dsname-relexp ,,vol list-relexp [,,OVRD]

list-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 Returns

DELABC    CAMLST    SCRATCH,DSABC,,VOLIST,,OVRD
DSABC     DC        CL44'A.B.C'        DATA SET NAMES
VOLIST    DC        H'2'               NUMBER OF VOLUMES
          DC        X'30C02008'        2314 DISK DEVICE CODE
          DC        CL6'000017'        VOLUME SERIAL NO.
          DC        H'0'               SCRATCH STATUS CODE
          DC        X'30C02008'        2314 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 in the chapter titled "Data Set Protection."

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.

Return Code in Reg. 15	**Meaning**
4	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	An unusual condition was encountered on one or more volumes.
12	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.
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 (Common VTOC Access Facility).

5 It could not be verified that this volume was mounted, and no device was available on which this 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 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 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 stored on one or more direct-access volumes by using the RENAME and CAMLST macro instructions. This causes the data set name in all identifier (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.

Refer to OS/VS2 MVS Resource Access Control Facility (RACF): General Information Manual for information on RACF-defined data sets. Only an accessor with alter access authority may rename a RACF-defined data set.

Refer to Data Facility Device Support: User's Guide and Reference for information on renaming a VTOC index 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 OS/VS2 System Programming Library: Debugging Handbook. Volumes are processed in the order 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 him to mount the volume. (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, he issues a reply indicating that he cannot fulfill the request. A condition 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	list-addrx RENAME ,dsname-relexp ,new name-relexp ,vol list-relexp

list-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. 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 Returns

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'30C02008'         2314 DISK DEVICE CODE
          DC       CL6'000017'         VOLUME SERIAL NO.
          DC       H'0'                RENAME STATUS CODE
          DC       X'30C02008'         2314 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 that follow. 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.

Return Code in Reg. 15	Meaning
4	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	An unusual condition was encountered on one or more volumes.
12	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.
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 (Common VTOC Access Facility).
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 chapter titled "Password Protecting Your Data Sets" provides a description of the protection mode indicator.

EXECUTING YOUR OWN CHANNEL PROGRAMS (EXCP)

The execute-channel-program (EXCP) macro instruction provides you with device dependence in organizing data and controlling I/O devices. 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.

To understand this chapter, you need to understand the information in these publications:

- OS/VS2 MVS Data Management Services Guide, which explains the standard procedures for I/O processing under the operating system.

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

- OS/VS2 MVS Data Management Macro Instructions, which describes the system macro instructions that can be used in programs coded in the assembler language.

- OS/VS2 System Programming Library: Debugging Handbook, which 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 I/O supervisor. (I/O supervisor is the name this chapter uses for two VS2 components, the EXCP processor and the I/O supervisor. For your 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, SYSOUT, or VSAM 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 also 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 supervisor 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, 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 supervisor will then:

5. Construct a request queue element (RQE) for scheduling the request.
6. If the requestor is in a pageable address space, fix the buffers so that they cannot be paged out and translate the requestor's virtual channel program into a real channel program.
7. Issue a start input/output (SIO) instruction to cause the channel to execute the real channel program.
8. Process I/O interruptions and schedule error recovery procedures when necessary.
9. Post a completion code in the event control block after the channel program has been executed.

Note: If the requestor is in a nonpageable address space, he provides a real channel program, 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 perform the procedures that the access methods 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 supervisor always 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 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 NONPAGEABLE ADDRESS SPACE

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

Modification of an active channel program by data read in or by CPU instructions is legitimate in a nonpageable address space, but not in a pageable 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 channel command words (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 formats used with specific I/O devices can be found in IBM publications for those devices. All channel command words described in these publications can be used, with the exception of REWIND and UNLOAD (RUN). In addition, both data chaining and command chaining may be used.

Chaining is the successive loading of channel command words into a channel from contiguous doubleword locations in real storage. Data chaining occurs when a new channel command word loaded into the channel defines a new storage area for the original I/O operation. Command chaining occurs when the new channel command word specifies a new I/O operation. For detailed information about chaining, refer to IBM System/370 Principles of Operation.

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.

If a channel program includes a list of channel command words that chain data for reading operations, no channel command word may alter the contents of another channel command word in the same list. (If such alteration were allowed, specifications could be placed into a channel command word without being checked for validity. If the specifications were incorrect, the error could not be detected until the chain was completed. Data could be read into incorrect locations and the system could not correct the error.)

CONTROL BLOCKS

When using EXCP, you must be familiar with the function and structure of the input/output block (IOB), the event control block (ECB), the data control block (DCB), and the data extent block (DEB). IOB and ECB fields are illustrated in the section "Control Block Fields." DCB fields are illustrated in the section "Macro Specifications for Use with EXCP." 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 supervisor validates the request by checking certain fields of the control blocks associated with this request. If the I/O supervisor detects invalid information in a control block, it initiates abnormal termination procedures.

The I/O supervisor 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 in the section "Appendages"—you should not be concerned with the contents of RQEs.)

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

- Execute the I/O operation normally, or

- Skip the I/O operation.

See "Appendages" in this chapter for a description of the SIO appendage and its linkage to the I/O supervisor.

If you are issuing EXCP from in a pageable address space, the channel program you construct contains virtual addresses. Because channels cannot use virtual addresses, the I/O supervisor 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 I/O supervisor builds the translated (real) channel program in a portion of real storage called the system queue area. 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 translated channel program into the channel address word (CAW) and issues a start input/output (SIO) instruction.

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 the 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 supervisor builds a define-extent command and passes control to your real channel program. (You cannot issue the initial seek, set the file mask, or define extent yourself. 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 or RDBACK, write commands are also prohibited.)

Before issuing SIO 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.)

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 nonpageable address space. It cannot run in a pageable address space because of the channel program translation performed by the I/O supervisor. (In a pageable 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.)

A program of this type can be changed to run in a pageable 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 (CSW). 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 in the unit control block (UCB) 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 supervisor 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 CPU 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 interruptions 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 is then restarted with another request that is not related to the channel program in error. (The paragraphs following this one under this topic 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 IFLGS field of the data control block. You are informed of the error by an error code that the system puts 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 I/O supervisor. To find out how, see "Input/Output Control Block Fields" in the section "Control Block Fields."

If a permanent error occurs in the channel program of a related request, the I/O supervisor does the following:

- Removes the request queue elements for all dependent channel programs from their queue and makes them available.

- Chains together the IOBs (input/output blocks) for the dependent channel programs.

The IOB chain 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 3800 Enhancements, 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 also OS/VS1 System Data Areas and Reference Manual for the IBM 3800 Printing Subsystem.)

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:

- Start I/O

- Program controlled interruption

- End of extent

- Channel end

- Abnormal end

Appendages get control in supervisor state, receiving the following pointers from the I/O supervisor:

- 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 I/O supervisor to which control is to be returned after execution of an appendage. When returning control to the I/O supervisor, you may use displacements from the return address in register 14. Allowable displacements are summarized in Figure 10 on page 52 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 10 on page 52 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 supervisor or I/O supervisor 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.

START-I/O (SIO) APPENDAGE

Unless an error procedure is in control, the I/O supervisor passes control to the SIO appendage just before the I/O supervisor translates your channel program. If I/O activity is not initiated because of a busy condition and the I/O request has not been translated, the appendage is not reentered before the SIO instruction is issued.

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

Reg. 14 + 0
Normal return. Normal channel program translation and SIO instruction execution occur.

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 CVT0PT01 in the communications vector table—in register 15.
3. Place 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 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'). (Refer to the topic "Block Multiplexor

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

[1] Certain register conventions for passing parameters from appendages to the I/O supervisor must be followed. These conventions are described in the appendage descriptions.

Figure 10. Entry Points, Returns, and Available Work Registers for Appendages

Channel Programming Notes" later in this chapter for special PCI conditions encountered with command retry.)

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. If the step is running ADDRSPC=REAL 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.

2. Put X'80' in the high-order byte of register 11 and the address of the ECB in the low-order bytes. This BR 14 must be in storage which is addressable from any address space (for example, CVTBRET).

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.

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

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

   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 four.

   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 obtained from the SRBPASID.

```
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.

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.

This procedure can be used even if the PCI appendage uses virtual storage addresses, but performance may be slightly slower. For more information on the POST routine, see OS/VS2 System Programming Library: Supervisor.

To return control to the I/O supervisor 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 I/O supervisor updates the seek address to the next higher cylinder or track address, and reexecutes 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 I/O supervisor, 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 routine, and then the abnormal-end appendage will be immediately entered with the completion code in IOBECBCC set to X'41'.

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 reexecution 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.)

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."

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 ERP encountering an unexpected load point, or

   b. The tape 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 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 _OS/VS2 System Programming Library: 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 of characters from WA to Z9. Note, however, if your program runs in a non-pageable address space and uses a PCI appendage, the PCI appendage and any appendage that the PCI appendage refers to cannot be placed in SYS1.LPALIB. Instead, these appendages 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 OS/VS2 System Programming Library: Initialization and Tuning Guide.

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 "authorized" program is one that runs in a protection key less than 8 or one that has 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:

```
//              EXEC     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 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:

```
//              EXEC     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,
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 ABE appendage.

BLOCK MULTIPLEXOR CHANNEL PROGRAMMING NOTES

Command retry is a function of the block multiplexor channel supporting the 2305, 3330/3333, 3340/3344, 3350, 3375, and 3380 direct-access devices. When the channel receives a retry request, it repeats the execution of the channel command word (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, since 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 interruptions 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 OS/VS2 MVS 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 11 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 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.

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 HIARC	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 11. Data Control Block Format for EXCP (After OPEN)

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 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 the LPA library.

PCIA=symbol

2-byte identification of a PCI appendage that you have entered into the LPA library.

SIOA=symbol

2-byte identification of a SIO appendage that you have entered into the LPA library.

CENDA=symbol

2-byte identification of a CHE appendage that you have entered into the LPA library.

XENDA=symbol

2-byte identification of an ABE appendage that you have entered into the LPA library.

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 exceptional conditions. The format of this exit list is given in OS/VS2 MVS Data Management Services Guide.

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.
- 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 OS/VS2 MVS Data Management Macro Instructions. When writing a data set to be read later, the RECFM, LRECL, and BLKSIZE should be specified to identify the data set attributes. LRECL and BLKSIZE can only be specified in a DD statement, since 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:

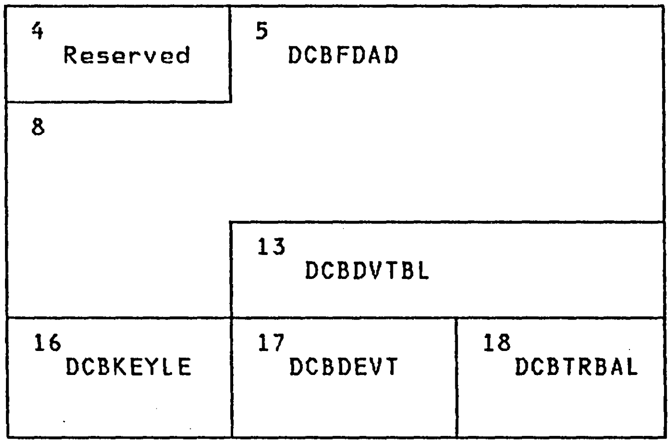
Code	**Device**
DA	Direct access
TA	Magnetic tape
PT	Paper tape
PR	Printer
PC	Card punch
RD	Card reader

Note: For MSS virtual volumes, DA should be used.

If you do not wish to select a specific device until job set-up time, you should specify the device type requiring the largest area.

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. 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 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[1]
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**
0 200 (2400 only)	—
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.

[1] Trademark of Teletype Corporation

DSORG Parameter of the DCBD Macro

In addition to the operands described in OS/VS2 MVS 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 each of these, label processing 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 piece of 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 in "Reading and Modifying a Job File Control Block."

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 an alternate good 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:

[symbol]	ATLAS	PARMADR={address} [,CHANPRG={R|NR}] [,CNTPTR={P|F}] [,WRITS={YES|NO}]

PARMADR
Address of a parameter address list of the following format:

0	Parameter list
4	IOB for the channel program that encountered the error
8	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 I/O supervisor 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 (except in the count area)
IOBSENS1	X'80'	Data check in the count area
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
        BO       ATLASGO              YES, VALID ERROR
        TM       IOBSENS1,X'80'       DATA CHECK IN COUNT
        BO       ATLASGO              YES, VALID ERROR
        TM       IOBSENS1,X'0A'       MISSING ADDRESS
*                                     MARKER AND NO RECORD
*                                     FOUND
*                                     YES, ATLAS CANNOT
        BO       OTHER                HANDLE ERROR; DO
ATLASGO EQU      *                    OTHER ERROR
*                                     PROCESSING.
*                                     NO, MISSING ADDRESS
*                                     MARKER ONLY.
        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 0, 36, 40, or 44, the contents of register 0 may be significant.

Decimal Return Code	Meaning
0	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	This device type does not have alternate tracks that can be assigned by programming.
8	All alternate tracks for the device have been assigned.
12	A request for storage (GETMAIN macro instruction) could not be satisfied.
16	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	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	The format-4 DSCB of the VTOC cannot be read; therefore alternate track information is not available to ATLAS.
28	The record specified by the user was the format-4 DSCB and it could not be read.
32	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	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	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 number of the record found defective but copied anyhow. If the channel program is reexecutable, it has been successfully reexecuted.
44	Error/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 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 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 The DCB reflects a track overflow data set, but the UCB device type shows that the device does not support track overflow.

60 The CCHH of the user-specified count area is not within the extents of the data set.

64 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 tape mark 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 tape mark 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 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 in "OPEN—Initialize Data Control Block."

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 OS/VS2 MVS 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 have been described with the parameters of the DCB macro instruction in the section "EXCP Programming Specifications."

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 12 the diagonally-ruled areas indicate fields in which you must specify information; the hyphen areas indicate fields in which you may 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.

```
+---------------------------+---+--------+-------------+-----------+-----------+
| 0(0)/|          |/|-|     |-- |        |             |           |           |  --+
|//////|IOBFLAG1  |/|-|     |-- |IOBFLAG2|  IOBSENS0   | IOBSENS1  |           |    |
|//////|          |/|-|     |-- |        |             |           |           |    |
+---------------------------+--------------------------------------------------+    |
| 4(4)                      |/////////////////////////////////////////////     |    |
|        IOBECBCC           |//////////////// IOBECBPT //////////////          |    |
|                           |/////////////////////////////////////////////     |    |
+---------------------------+--------------------------------------------------+    |
| 8(8)                      |                                                  |    |
|        IOBFLAG3           |                                                  |    |
+---------------------------+                 IOBCSW                           |    |
| 12(C)                                                                        |    |
|                                                                              |    |
+---------------------------+--------------------------------------------------+    > All
| 16(10)                    |/////////////////////////////////////////////     |    | Devices
|        IOBSIOCC           |//////////////// IOBSTART //////////////          |    |
|                           |/////////////////////////////////////////////     |    |
+---------------------------+--------------------------------------------------+    |
| 20(14)                    |/////////////////////////////////////////////     |    |
|        Reserved           |//////////////// IOBDCBPT //////////////          |    |
|                           |/////////////////////////////////////////////     |    |
+---------------------------+--------------------------------------------------+    |
| 24(18)                    |                                                  |    |
|        IOBRESTR           |                 IOBRESTR+1                       |    |
+---------------------------+-------------------+------------------------------+    |
| 28(1C) ///////////////////////////            |                              |    |
| ///////////// IOBINCAM //////////             |          IOBERRCT            |    |
| /////////////////////////////////             |                              |  --+
+---------------------------+-------------------+------------------------------+
| 32(20) //////////         |  --+
| ///    IOBSEEK    //      |    > Direct Access, Teleprocessing, and
| / (first byte, M)         |  --+            Graphic Devices
+---------------------------+--------------------------------------------------+
                            | 33(21) ////////////////////////////////////      |  --+ Direct
                            | ////////////////////////////////////////////     |    | Access
                            | ////////////////////////////////////////////     |    | 
+---------------------------+ ///             IOBSEEK              ////        |    > Storage
| ///////////////////////// (second through eighth bytes,          ////        |    | Devices
| /////////////////////////               BBCCHHR)                 ////        |    | (DASD)
| //////////////////////////////////////////////////////// 39(27)              |  --+
+------------------------------------------------------------------------------+
```

Figure 12. Input/Output Block Format

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 I/O supervisor 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 an earlier system are executed as type-1 requests if not modified.)

IOBSENS0 and IOBSENS1 (2 bytes)

are placed into the input/output block by the system 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 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."

IOBECBPT (3 bytes)

the address of the 4-byte event control block that 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 SIO 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."

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, etc.). 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 13 on page 77). 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 OS/VS2 System Programming Library: 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 13. 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 pageable area and provide the I/O supervisor with real channel programs. This eliminates the translation of channel programs by the I/O supervisor. 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 the OS/VS2 System Programming Library: Supervisor.

[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. You fix these data areas by building a list that contains these addresses of these 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 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 in the section titled "Appendages."

There is no advantage in using EXCPVR with a virtual input/output (VIO) data set. If EXCPVR is used, then virtual addresses must be used in the CCWs and indirect address lists (IDALs) are not allowed.

BUILDING THE LIST OF DATA AREAS TO BE FIXED

The I/O supervisor 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.

4. Any control blocks or other areas referred to in your PCI appendage (as well as, the DEB). Control blocks can be divided into two groups—system control blocks and user control blocks. The control blocks can be fixed or not fixed. If the control blocks are not fixed, they must be fixed in real storage.

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 I/O supervisor (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 I/O supervisor 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 addressing is required, the SIO appendage should also build the IDALs and turn on the IDAL indicators in the associated CCWs.

Translating Virtual Addresses and Building the IDAL: You can use the load real address (LRA) instruction to 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 page boundaries. If they do, you must provide an entry in the indirect address list (IDAL) for each page boundary crossed. The channel uses the IDAL to identify the address at which it will continue reading or writing when a page boundary is crossed during a read or write operation. If each buffer page is accessed, causing the page to be paged in, the LRA instruction can be used to translate the virtual addresses in the IDAL to real addresses. The IDAL must contain real addresses when it is processed by the channel.

```
CCW
 ______________________________________________________________
| Command  |   Address of the  |   04   | ////////// |  Byte   |
| Code     |   IDAL   |        |        | ////////// |  Count  |
|__________|__________|________|________|____________|_________|
0         7 8         |      31 32    39 40        47 48
                      |               IDAL
                      |__________> ________________________
                                  | 0                      |
                                  |   First Indirect Address|
                                  |________________________|
                                  | 4                      |
                                  |   Second Indirect Address|
                                  |________________________|
                                  | 8                      |
                                 _    Subsequent Indirect   _
                                 T    Address               T
                                  |________________________|
```

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 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 whether the data crosses a 2K-byte page boundary. 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 2K-byte page boundary, then two IDAL entries are required. If your data is 3000 bytes long, at least two IDAL entries are required. If your data crosses two 2K-byte page boundaries, three 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 pages (on a page boundary of 2048 or X'800') of the data area.

For example, if the data area real address is X'707FF' and the byte count is X'802' 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
```

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 instruction 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 OS/VS2 MVS Data Management Services Guide, 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, SYSOUT, or VSAM 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 device 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 contained in the "Executing Your Own Channel Programs (EXCP)" section of this publication.

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, 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" 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" 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. Yc 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 key-length-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.

blkref-addr—RX-type address or (2-12)
the address of a field in virtual storage containing the actual device 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. (See "Conversion of Relative Block Address to Actual Device Address" later in this chapter for more detailed information.)

sector-addr—RX-type address or (2-12)
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. (See "Obtaining Sector Number of a Block on a Device with the RPS Feature" later in this chapter for more detailed information.)

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 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 an absolute expression or decimal integer 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 OS/VS2 MVS 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 14 on page 87). 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. The "Control Block Fields" section in the EXCP section of this publication contains a diagram of the input/output block (Figure 14 on page 87). You may wish 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	**Interpretation**
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 14. 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 15.

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 15. The XDAP Channel Programs

When a sector address is specified with an RI, VI, or WI operation, the channel program is 32 bytes in length. Each of the channel programs in Figure 15 on page 87 would be, in this case, preceded by a set sector command.

CONVERSION OF RELATIVE TRACK ADDRESS TO ACTUAL DEVICE ADDRESS

To issue XDAP, you must provide the actual device 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 OS/VS2 System Programming Library: Debugging Handbook.)

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, etc.); 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	Successful conversion.
4	The relative block address converts to an actual device address outside the extents defined in the DEB.

CONVERSION OF ACTUAL DEVICE ADDRESS TO RELATIVE TRACK ADDRESS

To get the relative track address when you know the actual device 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 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.

PASSWORD PROTECTING YOUR DATA SETS

OS/VS password protection does not apply to VSAM data sets. Information about VSAM data set protection is in OS/VS Virtual Storage Access Method (VSAM) Programmer's Guide and OS/VS2 Access Method Services. Refer to OS/VS2 MVS Resource Access Control Facility (RACF): General Information Manual for information on RACF and its relationship to password protection. 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.
- For OS/VS2 systems with 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:

- OS/VS2 MVS Data Management Services Guide, which contains a general description of the data set protection feature.
- OS/VS Message Library: VS2 System Messages, which contains a description of the operator messages and replies associated with the data set protection feature for VS2.
- OS/VS2 JCL, which contains a description of the data definition (DD) statement parameter used to indicate that a data set is to be password protected.
- OS/VS2 DADSM Logic, which contains a description of the PASSWORD data set record format.
- OS/VS2 MVS Utilities, which contains a description of how to maintain the PASSWORD data set using the utility control statements of the IEHPROGM utility program.
- OS/VS2 TSO Command Language Reference, which 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, 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 the section in this chapter called "Using the PROTECT Macro Instruction to Maintain the PASSWORD Data Set." If you use the IEHPROGM utility program to maintain the PASSWORD data set, see OS/VS2 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. OS/VS2 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 system control program 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, or, if the program requesting that the data set be opened is running under TSO in the foreground, 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). These records are 80 bytes long (LRECL=80,BLKSIZE=80) and form the data area of the PASSWORD data set records on direct-access storage. In these direct-access storage records, the data area is preceded by a key area of 52 bytes (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. OS/VS2 DADSM Logic describes the PASSWORD data set record format.

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=) is used to indicate that a data set is to be placed under protection. Operating procedures at your installation must ensure that password records for all data sets currently under protection are entered in the PASSWORD data set. You may, for example, create a data set and set the protection indicator in its label, without entering a password record for it in the PASSWORD data set. However, once the data set is closed, any subsequent attempt to open results in termination of the program attempting to open the data set, unless the password record is available and the operator can honor the request for the password. (Note that if the protection mode is NOPWREAD and the request is to open the data set for input, 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 OS/VS Tape Labels 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/PWRITE).

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.

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/PWRITE 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 To PWREAD/PWRITE 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.

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 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 16 on page 97 through Figure 19 on page 101 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.

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.

Figure 16 (Part 1 of 2). Parameter List for ADD Function

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 16 (Part 2 of 2). 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.

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.

Figure 17 (Part 1 of 2). Parameter List for REPLACE Function

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 17 (Part 2 of 2). 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.

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 18. 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.

9 Pointer to current password.
    The address of the password of the record that you want listed.

Figure 19. Parameter List for LIST Function

RETURN CODES FROM THE PROTECT MACRO

When the PROTECT macro finishes processing, register 15 contains a return code that indicates what happened during the processing. Figure 20 contains the return codes and their meanings.

Register 15	Meaning
0	The updating of the PASSWORD data set was successfully completed.
4	The PASSWORD of the data set name was already in the password data set.
8	The password of the data set name was not in the PASSWORD data set.
12	A control password is required or the one supplied is incorrect.
16	The supplied parameter list was incomplete or incorrect.
20	There was an I/O error in the PASSWORD data set.
24[1]	The PASSWORD data set was full.
28	The validity check of the buffer address failed.
32[2]	The LOCATE macro failed. LOCATE's return code is in register 1, and the number of indexes searched is in register 0.
36[2]	The OBTAIN macro failed. OBTAIN's return code is in register 1.
40[2]	The DSCB could not be updated.
44	The PASSWORD data set does not exist.
48[2]	Tape data set cannot be protected.
52[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 these return codes, the PASSWORD data set has been updated, but the DSCB has not been flagged to indicate the protected status of the data set.

Figure 20. Return Codes from the PROTECT Macro Instruction

SYSTEM MACRO INSTRUCTIONS

This chapter describes miscellaneous macro instructions that allow you either to modify control blocks, obtain information from control blocks and system tables, or to perform track capacity calculations.

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

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

- OS/VS2 System Programming Library: Debugging Handbook, which 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)

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 can be made to it.

The fields in these blocks are shown and described in OS/VS2 System Programming Library: 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:

<table>
<tr><td>[symbol]</td><td>IEFUCBOB</td><td>[LIST={NO|YES}]
[,PREFIX={NO|YES}]</td></tr>
</table>

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. Code a DSECT statement before the macro statement if you require this.

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 topics that follow discuss the macro itself, 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 non-direct-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 devices; for any other type of device, two words of informat:on 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 zeroes 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 OS/VS2 System Programming Library: Debugging Handbook.

Word 1

Maximum blocksize. For direct-access devices, this value is the smaller of either the maximum size of an unkeyed block or the maximum blocksize allowed by the operating system; for magnetic or paper tape devices, this value is the maximum blocksize allowed by the operating system. For all other devices, this value is the maximum blocksize 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.

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 one (maximum blocksize) 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 and 3, if a block is not keyed.

If bit 3, byte 1 of word 4 is one, 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 formulas in OS/VS2 MVS Data Management Services Guide.

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 one, 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 21 on page 108 shows the actual output for each device type that results from issuing of 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]	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
2520 B2-B3	80	Not Applicable	Not Applicable
2520 B2-B3 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[2]	Not Applicable	Not Applicable
1403 w/UCS	120[2]	Not Applicable	Not Applicable
1404 Printer	120[2]	Not Applicable	Not Applicable
1443 Printer	120[2]	Not Applicable	Not Applicable
3203-5 Printer	132	Not Applicable	Not Applicable
3211 Printer	132[2]	Not Applicable	Not Applicable
3800 Printing Subsystem	136[3]	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

Figure 21 (Part 1 of 3). Output Obtained from Issuing DEVTYPE Macro

Device[1]	Maximum Record Size (Word 1, In Decimal)	DEVTAB (Words 2, 3, and 4, In Hexadecimal)	RPS (Word 5, In Hexadecimal)
3210 Printer-Keyboard	130	Not Applicable	Not Applicable
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
2305-1 Fixed-Head Storage	14660	0030000838E8027ACA080200	02985A57
2305-2 Fixed-Head Storage	14660	006000083A0A01215B080200	0140B4B1
3330/3333 Disk Storage	13030	019B0013336DBFBF38000200	00ED807C
3330V MSS Virtual Volume	13030	019B0013336DC1C13A010200	00ED807C
3330-1 (or 3333-11) 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
2250-1 Display Unit		Not Applicable	Not Applicable
2250-2 Display Unit		Not Applicable	Not Applicable

Figure 21 (Part 2 of 3). Output Obtained from Issuing DEVTYPE Macro

Device[1]	Maximum Record Size (Word 1, In Decimal)	DEVTAB (Words 2, 3, and 4, In Hexadecimal)	RPS (Word 5, In Hexadecimal)
2253-3 Display Unit		Not Applicable	Not Applicable
Communication Equipment		Record Size	
1030,1050,83B3, TWX,2250,S360		Not Applicable	
1060,115A,1130		Not Applicable	
2780		Not Applicable	
2740		Not Applicable	

Figure 21 (Part 3 of 3). Output Obtained from Issuing DEVTYPE Macro

Notes to Figure 21 on page 108 :

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 OS/VS2 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.

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." The symbolic names and field descriptions of the JFCB are contained in OS/VS2 System Programming Library: 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).

- 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 for a discussion of the facility).

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 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 System Code abnormal end 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 or 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 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 X'80' 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 in MVS 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 DEQed 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 DEQed 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 DEQed, the volume becomes available for allocation to another job. However, since 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 DEQed 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 DEQed 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 DEQed 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 which 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 DEQed 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 DEQed 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 DEQed 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 non-specific 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 OS/VS2 MVS 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 OS/VS2 MVS 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 OS/VS2 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 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 OS/VS2 MVS 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 22 on page 115, 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 22. 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 OS/VS2 MVS Data Management Services Guide. 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 (the 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, then a return code of 4 is placed in register 15.

Cautions: 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 OS/VS2 System Programming Library: Supervisor.

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 OS/VS2 MVS Data Management Services Guide, is available when an 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, a user with malicious intent 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.AMODGEN. 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 I/O supervisor 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. Use the **TYPE=ADD** and the **TYPE=VERIFY** operands of the macro, respectively.

To prevent an asynchronous routine from changing, deleting, or assigning a new DEB to a DCB, you must hold the local lock. To do this 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 _OS/VS2 Open/Close/EOV Logic_.

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. 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

```
[symbol]   DEBCHK   cbaddr
                    [,TYPE={VERIFY|ADD|DELETE|PURGE}]
                    [,AM={amtype|(amaddr)|((amreg))}]
                    [,BRANCH={NO|YES}]
                    [,TCBADDR=address]
                    [,KEYADDR=address]
                    [,SAVREG=reg]
                    [,MF=L]
```

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 data control block (DCB) 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.

for BRANCH=YES
The A-type address of a 4 byte field, or a register (1) or (3-12), that points to the DCB 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 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
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 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. **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 system control program, 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 1-14 that contains the access method value in its low-order byte (bit positions 24-31). 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. 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

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	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	An invalid TYPE was specified. (The DEBCHK routine was entered by a branch, not by the macro.)
12	Your program was not authorized and TYPE was not VERIFY.
16	DEBDCBAD did not contain the address of the DCB that was passed to the DEBCHK routine.
20	The AM value does not equal the value in the DEBAMTYP field.
24	The DEB is not on the DEB chain and TYPE=ADD was specified.
28	TYPE=ADD was specified for a DEB that was already entered in the DEB table.
32	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 EXCP macro instruction and are currently controlled by the I/O supervisor.
- Stop the processing of those requests that are not yet controlled by 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. See OS/VS2 I/O Supervisor Logic if you want to know the full scope of purge and restore processing.)

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 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 23 on page 121 and Figure 24 on page 121. 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 23. Macro Definition, JCL, and Utility Statements for Adding PURGE Macro to Your 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 24. Macro Definition, JCL, and Utility Statements for Adding RESTORE Macro to Your 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.)

.... 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 12-15. (If this bit is off, the purge routines don't 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 second word of each DEB but the last 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 isn't.)

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, or 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 25 on page 124.

12 A byte in which you can specify additional purge options. These are the bit settings and their meanings:

..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 selectjon 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 resotre request.)

.... .0.. Must be zero.

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 hexadecimal return codes:

Code	Meaning
4	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	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.
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

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 25 on page 124). (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.

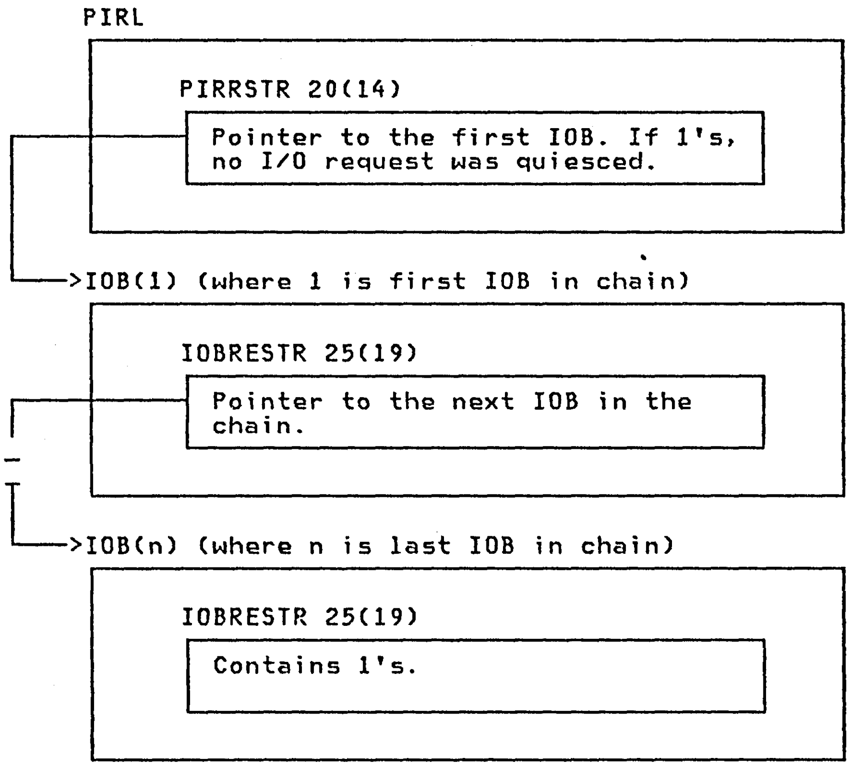

PIRL
PIRRSTR 20(14)
Pointer to the first IOB. If 1's, no I/O request was quiesced.
>IOB(1) (where 1 is first IOB in chain)
IOBRESTR 25(19)
Pointer to the next IOB in the chain.
>IOB(n) (where n is last IOB in chain)
IOBRESTR 25(19)
Contains 1's.

Figure 25. The PIRL and IOB Chain

TRKCALC—PERFORM 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
calculates the new track balance. Depending upon whether the record fits on the track, one of the following occurs:

- The record fits on the track. Register 0 contains the new track balance.
- If the record does not fit on the track and MAXSIZE=YES is not specified, a "record does not fit" return code is given.
- 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.

TRKCAP
calculates the number of fixed length records that can be written on a whole track or the number of records that could be added to a partial track of R ≠ 1 (record number ≠ 1 in the R or RKDD keywords).

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.

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 specified is the value returned when you last issued TRKCALC if R ≠ 1. If R ≠ 1 the balance is reset to track capacity by TRKCALC.

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|NO}

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=addr—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=addr—RX-type address, (2-12), (0), (14), or n
you may specify either the address of the cords's key length, or you may specify the key length using the low-order byte of a register or immediate data (n). Specify a decimal digit for n (immediate data).

K=addr—RX-type address, (2-12), (0), (14), or n
you may specify either the address of the record's key 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).

DD=addr—RX-type address, (2-12), (0), (14), or n
you may specify either the address 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 immediately date (n). Specify a decimal digit for n (immediate data).

REGSAVE={YES|NO}

YES
specifies registers 1-14 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

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. Not available as an input register.

OUTPUT FROM TRKCALC

FUNCTN=TRKBAL

Register 15=0
The record fits on the track. Register 0 contains the new track balance.

Register 15=4
Record does fit on the track. If MAXSIZE=YES is specified, a partial record does not fit either. Register 0 is 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 is 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 calculated input track balance or the specified record number is 1. STARBAL is updated to the new (output) track balance if the record does not fit. 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 set to the calculated input track balance if you do not provide the balance, or the specified record number is one.

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,{parameter list address|(1)})

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,{parameter list address|(1)})
    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

    parameter list address—RX-type address, (2-12), (0), (14), or (1)

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
          .
          .
          ST    0,NUMREC    SAVE NUMBER OF RECORDS
          .
          .
CL        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
          .
          .
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.

ADDING TO THE IMAGE LIBRARY AND RETRIEVING FCB IMAGES

This chapter provides a detailed description of how to add either an IBM UCS (universal character set) image or an IBM FCB (forms control buffer) image to SYS1.IMAGELIB. It also describes a procedure that can be used to read an FCB image into virtual storage for the purpose of modifying it before loading it into the forms control buffer.

For the IBM 3800 Printing Subsystem, a utility, IEBIMAGE, is provided to build the 3800 control modules (character arrangement table modules, forms control buffer modules, graphic character modification modules, and copy modification modules) and store them in SYS1.IMAGELIB. With 3800 Enhancements, IEBIMAGE can also be used to build library character set modules to be stored in SYS1.IMAGELIB. For additional information, see IBM 3800 Printing Subsystem Programmer's Guide.

Before reading this section, you should be familiar with the information in these publications:

- IBM 2821 Control Unit Component Description contains the information necessary to create a user-designed chain/train for the 1403 Printer.

- OS/VS2 MVS Data Management Macro Instructions describes the SETPRT macro instruction that loads a UCS image and an FCB image into their respective buffers.

- OS/VS2 JCL describes the UCB and FCB parameters that can be specified in a DD statement to load the UCS and FCB buffers when they are opened.

- IBM 3203 Printer Component Description and Operator's Guide contains the information necessary to create 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 the information necessary to create a user-designed train for the 3211 Printer.

- OS/VS2 MVS System Programming Library: JES2 or System Programming Library: Network Job Entry Facility for JES2 for reference information for JES2.

- OS/VS2 System Programming Library: JES3 for reference information for JES3.

ADDING A UCS IMAGE TO THE IMAGE LIBRARY

All IBM standard character set images are included in SYS1.IMAGELIB at system generation, when you code the DATAMGT macro. You may subsequently add a character set image to SYS1.IMAGELIB by following these rules:

1. The member name must be either the four characters UCS1 for the 1403,1UCS2 for the 3211, or UCS3 for the 3203 printer. The member name must be followed by a unique character set code that is one to four characters long. This character set code can be any valid combination of letters and numbers according to the rules for assembler language symbols. The single letters U or C should not be used as a character set code, since 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 instruction to load the image into the UCS buffer.

2. The first byte in the load module of a character set image specifies whether or not the image is a default. (Default images may be used by the system for jobs that do not request a specific image.) You may specify the following in the first byte if you have JES2:

   JES2

X'80'	indicates a default image
X'40'	indicates 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

   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.

4. Each byte of the next n bytes indicates the number of characters to be printed on each verification line. (Note: For the 3211 printer, the maximum number of characters printed per line is 48; the associative bytes are not printed during verification.)

5. A 240-byte 1403 UCS image, a 240-byte 3203 UCS image, or a 512-byte 3211 UCS image must follow the previously described fields. (A 3211 UCS image has 432 characters, followed by 15 bytes of X'00', 64 bytes of associative bits, and a reserved byte (the 512th byte) of X'00'. A 3203 UCS image has 240 characters followed by 64 bytes of associative bits.) Two apostrophes or two ampersands must be coded to represent a single apostrophe or a single ampersand, respectively, which is a part of a character set image.

Figure 26 on page 132 is an example of adding a 1403 UCS image, IM, to the image library.

Figure 27 on page 133 shows the code used to add a 3211 UCS image (IMG) to the image library. Two ampersands must be coded to represent a single ampersand that is part of the character set image.

The 64 bytes of associative bits must be coded to avoid data checks. To determine how to code these bits for a particular train, see IBM 3211 Printer, 3216 Interchangeable Train Cartridge, and 3811 Printer Control Unit Component Description and Operator's Guide.

Figure 28 on page 134 shows the code used to add a 3203 UCS image (YN) to the image library. A 3203 UCS image has 240 characters, followed by 64 bytes of associative bits. Two ampersands or two apostrophes must be coded to represent a single ampersand or a single apostrophe, respectively, that is part of the character set image.

The 64 bytes of associative bits must be coded to avoid data checks. To determine how to code these bits for a particular train, see IBM 3203 Printer Component Description and Operator's Guide.

Notes:

1. Executing the ASMFCL procedure does not actually generate executable code. The assembler/linkage editor is used as a vehicle to load the UCS image into the image library.

2. The SPACE parameter is overridden here because the IBM-distributed ASMFCL cataloged procedure has secondary allocation specified. All members must reside completely in the first extent.

```
//ADDIM        JOB  MSGLEVEL=1
//STEP         EXEC PROC=ASMFCL,PARM.ASM='NODECK,LOAD',
//             PARM.LKED='LIST,OL,REFR,RENT,XREF' (See note)
//ASM.SYSIN    DD     *
UCS1IM         CSECT
               DC  X'80'    (THIS IS A DEFAULT IMAGE)
               DC  AL1(6)  (NUMBER OF LINES TO BE PRINTED)
               DC  AL1(39) (39 CHARACTERS PRINTED ON 1ST LINE)
               DC  AL1(42) (42 CHARACTERS PRINTED ON 2ND LINE)
               DC  AL1(39) (39 CHARACTERS PRINTED ON 3RD LINE)
               DC  AL1(39) (39 CHARACTERS PRINTED ON 4TH LINE)
               DC  AL1(42) (42 CHARACTERS PRINTED ON 5TH LINE)
               DC  AL1(39) (39 CHARACTERS PRINTED ON 6TH LINE)
               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(UCS1IM),DISP=OLD,SPACE=
```

Note: The RENT and REFR linkage editor attributes are used for performance considerations in a paging environment and may be omitted.

Figure 26. Sample Code to Add a 1403 UCS Image to SYS1.IMAGELIB

```
//ADDIMG       JOB  MSGLEVEL=1
//STEP         EXEC PROC=ASMFCL,PARM.ASM='NODECK,LOAD',
//             PARM.LKED='LIST,OL,REFR,RENT,XREF' (See note)
//ASM.SYSIN    DD    *
UCS2IMG        CSECT
               DC  X'80'        (THIS IS A DEFAULT IMAGE)
               DC  AL1(9)       (NUMBER OF LINES TO BE PRINTED)
               DC  AL1(48)      (48 CHARACTERS PRINTED ON 1ST LINE)
               DC  AL1(48)      (48 CHARACTERS PRINTED ON 2ND LINE)
               DC  AL1(48)      (48 CHARACTERS PRINTED ON 3RD LINE)
               DC  AL1(48)      (48 CHARACTERS PRINTED ON 4TH LINE)
               DC  AL1(48)      (48 CHARACTERS PRINTED ON 5TH LINE)
               DC  AL1(48)      (48 CHARACTERS PRINTED ON 6TH LINE)
               DC  AL1(48)      (48 CHARACTERS PRINTED ON 7TH LINE)
               DC  AL1(48)      (48 CHARACTERS PRINTED ON 8TH LINE)
               DC  AL1(48)      (48 CHARACTERS PRINTED ON 9TH LINE)
*                               THE FOLLOWING NINE LINES REPRESENT
*                               THE TRAIN IMAGE
               DC  C'1<.+IHGFEDCBA*$-RQPONMLKJ%,&&ZYXWVUTS/@#098765432'
               DC  C'1<.+IHGFEDCBA*$-RQPONMLKJ%,&&ZYXWVUTS/@#098765432'
               DC  C'1<.+IHGFEDBCA*$-RQPONMLKJ%,&&ZYXWVUTS/@#098765432'
               DC  C'1<.+IHGFEDCBA*$-RQPONMLKJ%,&&ZYXWVUTS/@#098765432'
               DC  C'1<.+IHGFEDBCA*$-RQPONMLKJ%,&&ZYXWVUTS/@#098765432'
               DC  C'1<.+IHGFEDCBA*$-RQPONMLKJ%,&&ZYXWVUTS/@#098765432'
               DC  C'1<.+IHGFEDBCA*$-RQPONMLKJ%,&&ZYXWVUTS/@#098765432'
               DC  C'1<.+IHGFEDCBA*$-RQPONMLKJ%,&&ZYXWVUTS/@#098765432'
               DC  C'1<.+IHGFEDBCA*$-RQPONMLKJ%,&&ZYXWVUTS/@#098765432'
               DC  15X'00'      RESERVED FIELD, BITS 433-447
*  THE FOLLOWING FOUR DC INSTRUCTIONS DEFINE THE ASSOCIATIVE BITS,
*  UCSB BYTE POSITIONS 448-511
               DC  X'C0101010101010101010004040424000401 0'
               DC  X'10101010101010101000404100004040101 0'
               DC  X'10101010101000404000000010101010101 0'
               DC  X'1010101000404044480 0'
               DC  X'00'     RESERVED FIELD, BYTE 512
               END
/*
//LKED.SYSLMOD     DD     DSNAME=SYS1.IMAGELIB(UCS2IMG),DISP=OLD,SPACE=
```

Note: The RENT and REFR linkage editor attributes are used for performance considerations in a paging environment and may be omitted.

Figure 27. Sample Code to Add a 3211 UCS Image to SYS1.IMAGELIB

```
//ADYN3203   JOB  MSGLEVEL=1
//STEP       EXEC PROC=ASMFCL,PARM.ASM='NODECK,LOAD',
//           PARM.LKED='LIST,OL,REFR,RENT,XREF' (See note)
//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 PRINTED ON 1ST LINE)
             DC  AL1(42)     (42 CHARACTERS PRINTED ON 2ND LINE)
             DC  AL1(39)     (39 CHARACTERS PRINTED ON 3RD LINE)
             DC  AL1(39)     (39 CHARACTERS PRINTED ON 4TH LINE)
             DC  AL1(42)     (42 CHARACTERS PRINTED ON 5TH LINE)
             DC  AL1(39)     (39 CHARACTERS PRINTED ON 6TH LINE)
*                            THE FOLLOWING SIX LINES REPRESENT
*                            THE TRAIN IMAGE
             DC  C'1234567890STABCDEFGHIJKLMNOPQRUVWXYZ*,.'
             DC  C'1234567890STABCDEFGHIJKLMNOPQRUVWXYZ*,.#-$'
             DC  C'1234567890STABCDEFGHIJKLMNOPQRUVWXYZ*,.'
             DC  C'1234567890STABCDEFGHIJKLMNOPQRUVWXYZ*,.'
             DC  C'1234567890STABCDEFGHIJKLMNOPQRUVWXYZ*,.#-$'
             DC  C'1234567890STABCDEFGHIJKLMNOPQRUVWXYZ*,.'
* THE FOLLOWING FOUR DC INSTRUCTIONS DEFINE THE ASSOCIATIVE BITS,
* UCSB BYTE POSITIONS 241-304
             DC  X'C01010101010101010100040000000000010'
             DC  X'101010101010101000404000000040001010'
             DC  X'101010101010004000000000101010101010'
             DC  X'10101010004000000000'
             END
/*
//LKED.SYSLMOD   DD       DSNAME=SYS1.IMAGELIB(UCS3YN),DISP=OLD,SPACE=
```

Note: The RENT and REFR linkage editor attributes are used for performance considerations in a paging environment and may be omitted.

Figure 28. Sample Code to Add a 3203 UCS Image to SYS1.IMAGELIB

ADDING AN FCB IMAGE TO THE IMAGE LIBRARY

For the 3800 Printing Subsystem, refer to the IBM 3800 Printing Subsystem Programmer's Guide.

Two standard FCB images, STD1 and STD2, can be included in SYS1.IMAGELIB during system generation for a 3211 or 3203 printer (see Figure 29 on page 136 and Figure 30 on page 137 for a sample of STD1 and STD2 images). STD1 prints six lines per inch on an 8-1/2-inch form. STD2 prints six lines per inch on an 11-inch form. Channels for both images are evenly spaced, with channel one on the fourth line and channel nine on the last line.

In addition to the IBM-supplied images, user images can be defined. Each user image is added to the image library as part of a load module. To add an FCB image to the image library, follow these rules:

1. The member name cannot exceed eight bytes. The first four characters of this member name must be FCB2. The characters that follow FCB2 identify the FCB image and are referred to as the image identifier. Any combination of characters that are valid in assembler language can be used with the exception of a single "C" or a single "U" as an image identifier. The image identifier must be specified in a DD statement or in the SETPRT macro instruction to load the image in the FCB buffer.

2. The first byte of the load module of a forms control image specifies whether or not the image is a default. A default image is indicated by X'80' and is used for all jobs that do not have the FCB parameter coded on the DD statement; 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 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. A description of the print position indexing feature and its use may be found in IBM 3211 Printer, 3216 Interchangeable Train Cartridge, and 3811 Printer Control Unit Component Description and Operator's Guide.

   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 contain a channel identifier.

   Typically, the length of an FCB image is constant 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 which is 51 bytes in length (8-1/2-inches times 6 LPI).

   - X'1n' means eight lines are printed per inch.

   - X'0n' means six lines are printed per inch.

5. All remaining bytes (lines) must contain X'0n' except the last byte. The last byte must be X'1n'. The letter n can be a hexadecimal value from 1 to C, representing a channel (one to twelve); or it can be zero (0), which means no channel is indicated.

In Figure 29 on page 136, an FCB load module is assembled and added to SYS1.IMAGELIB. The image defines a print density of eight lines per inch on an 11-inch form with a right shift of 15 line character positions (1-1/2 inches).

```
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 29. Sample of the Standard FCB Image STD1

RETRIEVING AN FCB IMAGE

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, it's necessary to issue another IMGLIB macro, but this time 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.

The format of the BLDL and the SETPRT macros is given in *OS/VS2 MVS Data Management Macro Instructions*; the format of the LOAD macro is given in *OS/VS2 Supervisor Services and Macro Instructions*. Shown here is the format of the IMGLIB macro:

[symbol]	**IMGLIB**	{OPEN|CLOSE,addr}

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.

```
FCB2STD2    CSECT
            DC     X'80'                 DEFAULT IMAGE
            DC     AL1(66)               FCB IMAGE LENGTH = 66
            DC     X'000000'             LINES 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 FORM
            END
```

Figure 30. Sample of the Standard FCB Image STD2

CLOSE
specifies that SYS1.IMAGELIB is to be closed.

addr
RX-type address of word that points to the DCB. If coded in the form (1-12), then the register contains the address of the DCB, not the address of the fullword.

Return codes for IMGLIB OPEN:

Decimal Return Code	Meaning
0	Successful.
4	Either the volume containing SYS1.IMAGELIB is not mounted or a required catalog volume was not mounted.
8	Either SYS1.IMAGELIB does not exist on the volume to which the catalog points, or it is not cataloged.
12	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.

```
//ADDFCB       JOB    MSGLEVEL=1
//STEP         EXEC PROC=ASMFCL,PARM.ASM='NODECK,LOAD',
//  PARM.LKED='LIST,OL,REFR,RENT,XREF' (See note)
//ASM.SYSIN    DD     *
FCB2ID1        CSECT
*THIS EXAMPLE IS FOR A FORM LENGTH OF 11 INCHES
*WITH 8 LINES OF PRINT PER INCH (88 LINES)
               DC     X'80'     THIS IS A DEFAULT IMAGE
               DC     AL1(89)   LENGTH OF FCB IMAGE
               DC     X'8F'     OFFSET PRINT LINE 15
*CHARACTER POSITIONS 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'
               DC     X'03'
               DC     XL6'0'
               DC     X'04'
               DC     XL6'0'
               DC     X'05'
               DC     XL6'0'
               DC     X'06'
               DC     XL6'0'
               DC     X'07'
               DC     XL6'0'
               DC     X'08'
               DC     XL6'0'
               DC     X'09'
               DC     XL6'0'
               DC     X'0A'
               DC     XL6'0'
               DC     X'0B'
               DC     XL6'0'
               DC     X'0C'     CHANNEL 12 IN LINE 83
               DC     XL4'0'    4 LINES NO CHANNEL
               DC     X'10'     LINE 88--LAST LINE IN IMAGE
               END
/*
//LKED.SYSLMOD DD     DSNAME=SYS1.IMAGELIB(FCB2ID1),DISP=OLD,SPACE=
```

Note: The RENT and REFR linkage editor attributes are used for performance considerations in a paging environment and may be omitted.

Figure 31. Sample Code to Assemble and Add an FCB Load Module to SYS1.IMAGELIB

JES2 SUPPORT FOR THE IBM 1403, 3203-5, AND 3211 PRINTERS

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. (See the DATAMGT Macro in OS/VS2 System Programming Library: System Generation Reference.)

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 OS/VS 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/inch and FCB28 for 8 lines/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/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 1-32 (default=1). The index value sets the left-hand margin (1 indicates flush-left position; other values cause indentation of the print line by N-1 position).

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/inch) other than X'80', JES2 uses that specification and supplies an index value of 1.

3203-5 PRINTER

The 3203-5 Printer is treated the same as a 3211 printer by JES2 and JES2 NJE, except that the 3203-5 does not support the 3211 indexing feature, and any indexing commands from JES2 or JES2 NJE are ignored by the 3203-5. The 3203-5 uses 3211 FCB images and its own unique UCS images. UCS images are listed in OS/VS2 System Programming Library: System Generation Reference.

FORMAT-1 DSCB-NOT-FOUND USER EXIT IN OPEN AND EOV

The function of the Format-1 DSCB-not-found user 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.PALIB. 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	Unpredictable
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:

0 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 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 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').

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.

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 into MVS, see _OS/VS2 System Programming Library: Supervisor_. Your modules may replace IGG026DU, IGG029DU, 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, and IGG030DU.

The prolog of each of the dummy modules contains register conventions and other information about these modules.

SCRATCH DUMMY MODULE

The load module for SCRATCH(SVC29) 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 special error processing is required after SVC29, the dummy module can be replaced with your own module. Your replacement module 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 into MVS see OS/VS2 System Programming Library: Supervisor. Your module may replace IGG029DM in SYS1.AOSD0 prior to system generation, or you may replace the dummy module in SYS1.LPALIB after system generation. Information on how to replace the dummy module with your module 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 SCRATCH with SMP without modifying your own version of IGG029DM.

The prolog of the dummy module contains register conventions and other information about this module.

CONTROLLING SPACE ON DASD VOLUMES

INTRODUCTION

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. The VTOC is built when the volume is initialized by the direct-access storage device initialization (Device Support Facilities, IEHDASDR or IBCDASDI) utility program. See "The Volume Table of Contents" in this section for more information about the VTOC.

The VTOC is a collection of data set control blocks (DSCBs). The different types of DSCBs are:

1. Free VTOC record DSCB—format-0

2. Identifier DSCB—format-1

3. Index DSCB—format-2

4. Extension DSCB—format-3

5. VTOC DSCB—format-4

6. Free space DSCB—format-5

7. Shared extent DSCB—format-6

Each DSCB corresponds either to a data set or data space currently residing on the volume, or to contiguous, unassigned tracks on the volume. DSCBs are the data set labels, which contain characteristics of the data sets or data spaces and a description of the tracks on which the data sets resides. DSCBs for unassigned tracks indicate the location of unassigned, contiguous tracks.

The Allocate and Extend routines assign tracks and cylinders on direct-access volumes. The Allocate routines are used by the scheduler component to get space for new data sets. The Extend routine is called by the system catalog management and End-of-Volume components to get more space for a data set (or VSAM data space) that has already been allocated, but needs more space. Other DADSM routines (Scratch and Partial Release) are used to release space that is no longer needed on a direct-access volume.

When space is needed on a volume, the DADSM routines check the VTOC 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 DADSM routines delete the DSCBs of the deleted data set or data space. A free space (format-5) DSCB is built, or modified if existent, to indicate that the tracks containing the affected data set or data space can be reallocated.

DADSM ROUTINES

DADSM's space management routines are concerned with:

1. Allocating primary space, which involves finding space for new data sets or for VSAM data spaces. These are the Allocate routines.

2. Allocating secondary space, which involves finding additional space for data sets or VSAM data spaces that have exceeded their original, primary allocations. This is the Extend routine.

3. Releasing space, which involves both deleting entire data sets or data spaces that are no longer needed, and freeing unused space in data sets that are being retained. These are the Scratch and Partial Release routines.

DADSM's VTOC-related service routines are concerned with:

1. Changing the names of data sets. This is the Rename routine.

2. Making control information available for examination. This is the Obtain routine.

3. Determining the space available on a direct-access volume. This is the LSPACE routine.

4. Maintaining the system PASSWORD data set, which controls access to data sets and their associated control information. This is the Protect routine.

ALLOCATING AND RELEASING SPACE ON DIRECT-ACCESS VOLUMES

The DADSM routines which allocate space (Allocate and Extend), and release space (Scratch and Partial Release), add, delete, and modify records of the VTOC. These records are called data set control blocks (DSCBs). To make space available to a new data set or to increase the space allocated to a data set, the appropriate DSCBs are searched for available space; the space is allocated to the data set by writing the description of the space, called an extent, to the data set's DSCB and deleting the extent from the space available for allocation. To release space allocated to a data set, the allocate operation is reversed: the released extent is deleted from the data set's DSCB and added to the DSCB that describes available space.

Components of the operating system use the DADSM routines to allocate and release space in response to data definition (DD) statements. For example, job management (scheduler) routines call the Allocate routines to obtain space for a new data set. The End-of-Volume component of Open/Close/End-of-Volume (O/C/EOV) calls the Extend routine when an existing data set needs more space; the MVS catalog management routines call the Extend routine to get more space for a VSAM data space; and the OS catalog management routines call the Extend routine to allocate additional space for an OS catalog. Similarly, job management routines use the Scratch routine to delete data sets, and the catalog management routines use the Scratch routine to delete a data set when uncataloging involves deleting a data set of a generation data group. Utility programs (IEHPROGM, IEHMOVE, and IEBCOPY) use the Scratch and Allocate routines. Scratch processing is also available to the system programmer through the SCRATCH macro instruction.

The virtual storage access method (VSAM) allocates and releases space using the DADSM Allocate, Extend, and Scratch routines. These DADSM routines are called by the MVS catalog management routines to allocate, extend, and delete VSAM data spaces.

The Partial Release routine is called by the Close routine of O/C/EOV to release unused space before a data set is closed. Partial Release is also called by the reposition-I/O routine of Checkpoint/Restart to release unused space.

VTOC-RELATED SERVICE ROUTINES

While Rename, Obtain, LSPACE, and Protect routines are used to read and change control information on the VTOC, none allocates or releases space. System macro instructions can be used to invoke the Rename, Obtain, and Protect routines (information for these macro instructions is provided in the section "Using Catalog Management Macro Instructions").

The Rename routine finds the DSCB for a specified data set and changes its name, after verifying that the requested name does not duplicate one already on the volume.

The Obtain routine finds the DSCB for a specified data set, then reads the DSCB into virtual storage. The Obtain routine is also used to get information about VSAM data sets from the VTOC, the MVS master catalog, or a VSAM user catalog.

The LSPACE routine is called either (1) by routines issuing demount messages for direct-access volumes (for example, scheduler and O/C/EOV) when the operator has issued a "MONITOR SPACE" command or (2) by the System Management Facilities (SMF). The available space on the volume is calculated by searching and totaling the extents contained in the free space (format-5) DSCBs. At the same time, the largest available extent on the volume is located. If SMF information is required, an SMF type-19 record is gathered and written to the SMF data set.

The Protect routine adds, replaces, deletes, or lists entries in the PASSWORD data set. When the security protection status of a data set changes, the Protect routine also modifies the protection mode indicator field in the protected data set's DSCB.

THE VOLUME TABLE OF CONTENTS

The volume table of contents (VTOC) is a data set consisting of 140-byte data set control blocks (DSCBs) that describe the contents of a direct-access storage device volume. The VTOC data set resides in a single extent (that is, it is a continuous data set); its address is located in the VOLVTOC field of the standard volume label (see Figure 32 on page 148). There are seven different kinds of DSCBs. Each has a different purpose and is, consequently, given a different name and format number. Figure 33 on page 149 lists each DSCB and its use.

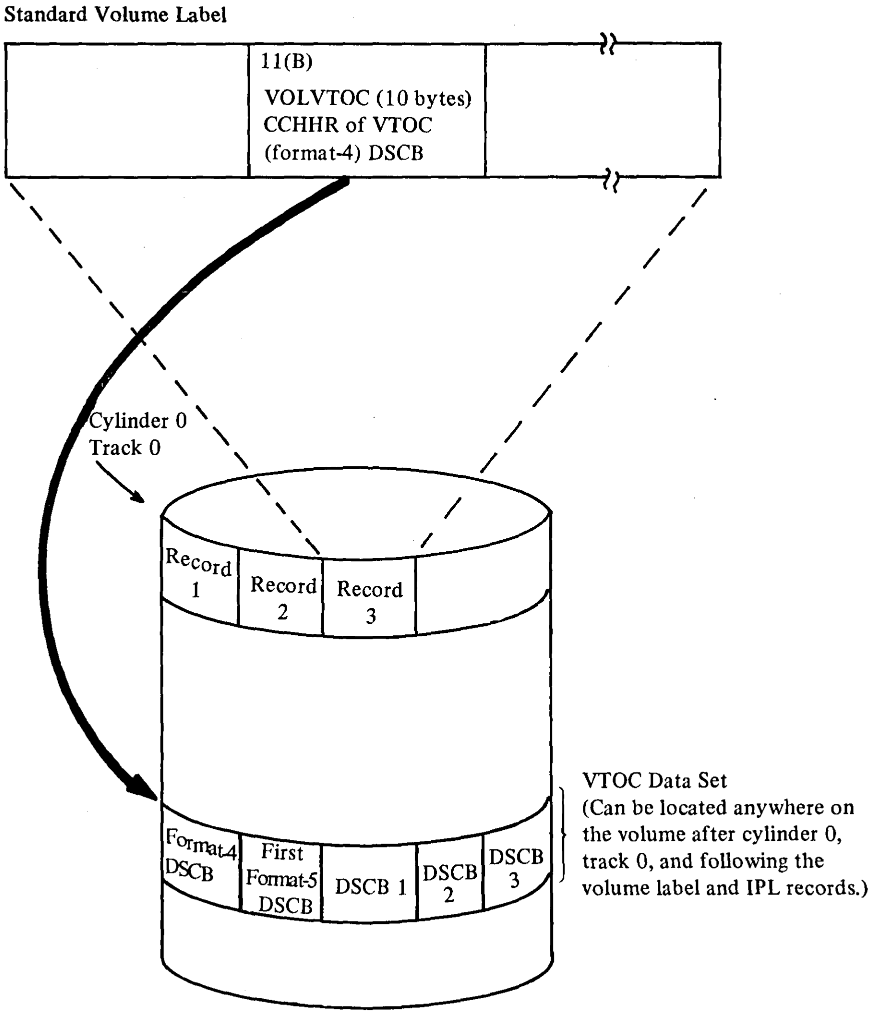


Figure 32. Locating the Volume Table of Contents (VTOC)

DSCB Name	DSCB Format Number	Function	How Many	How Found
Identifier	1	Describes a data set set or VSAM data space and the first three extents.	One for every data set or data space on the volume, except the VTOC.	Search on key equal to the data set name.
Index	2	Describes the indexes of an ISAM	See "ISAM Data Set Allocation" in OS/VS2 DADSM Logic	Chained from a format-1 DSCB that represents the data set.
Extension	3	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.)	One for each data set or VSAM data space on the volume that has more than three extents.	Chained from a format-2 or a format-1 DSCB that represents the data set or VSAM data space.
VTOC	4	Describes the extent and contents of the VTOC and volume and device characteristics.	One on each volume.	VOLVTOC field of the standard volume label contains its address. It is always the first record in the VTOC.
Free Space	5	Describes the space on a volume that has not been allocated to a data set or to a VSAM data space (available space).	One for every 26 non-contiguous extents of available space on the volume.	The first format-5 DSCB on the volume is always the second record of the VTOC. If there is more than one format-5 DSCB, it will be chained from the first format-5 DSCB via the DS5PTRDS field of each format-5 DSCB.
Shared Extent	6	Describes the extents shared by two or more data sets (split-cylinder extents).	One for every 26 split-cylinder extents on the VTOC.	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 to the first via the DS6FTRDS field of the format-6 DSCB.

Figure 33 (Part 1 of 2). Data Set Control Block (DSCB) Format Types and Use

DSCB Name	DSCB Format Number	Function	How Many	How Found
Free VTOC Record	0	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.	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.	Search on key equal to X'00' (sometimes X'00000000').

Figure 33 (Part 2 of 2). Data Set Control Block (DSCB) Format Types and Use

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

A data set or VSAM data space is defined by one, two, or three 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 fourth through the sixteenth extents). Figure 34 on page 151 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 34 on page 151) 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 can no longer be allocated on MVS systems, but existing split-cylinder data sets can still be processed.

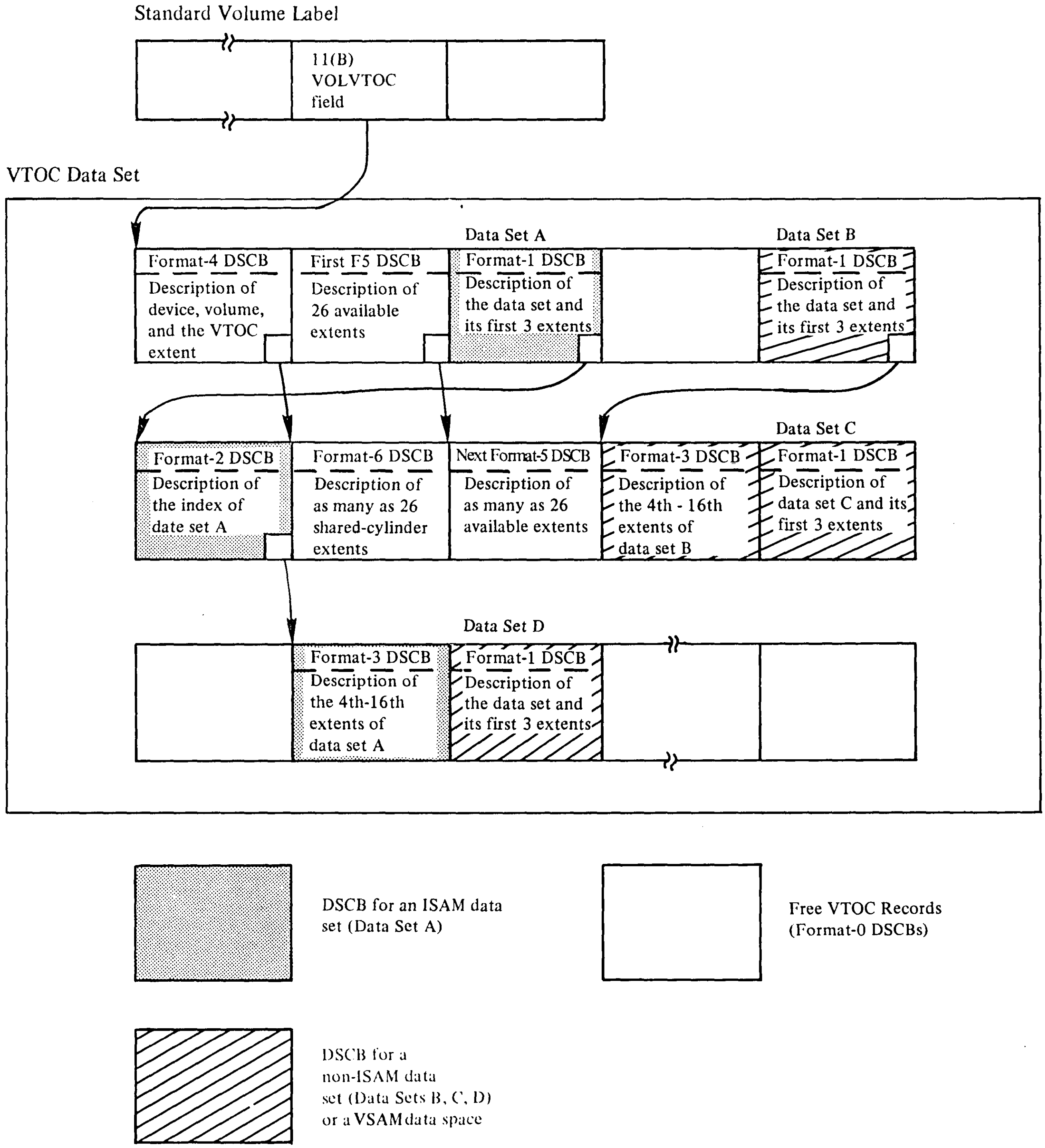

Standard Volume Label
11(B)
VOLVTOC
field
VTOC Data Set
Format-4 DSCB
Description of device, volume, and the VTOC extent
First F5 DSCB
Description of 26 available extents
Data Set A
Format-1 DSCB
Description of the data set and its first 3 extents
Data Set B
Format-1 DSCB
Description of the data set and its first 3 extents
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
Data Set C
Format-1 DSCB
Description of data set C and its first 3 extents
Format-3 DSCB
Description of the 4th-16th extents of data set A
Data Set D
Format-1 DSCB
Description of the data set and its first 3 extents
DSCB for an ISAM data set (Data Set A)
Free VTOC Records (Format-0 DSCBs)
DSCB for a non-ISAM data set (Data Sets B, C, D) or a VSAM data space

Figure 34. Contents of VTOC—DSCBs Describing Data Sets

To prepare a volume for use (to initialize it), the Device Support Facilities, IBCDASDI or IEHDASDR utility is used. One of the things these utilities do is build the VTOC. After initialization, this VTOC will contain a format-4 DSCB and a format-5 DSCB. 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.

SIZE OF THE VOLUME TABLE OF CONTENTS

The number of DSCBs in the VTOC determines the number of data sets or VSAM data spaces that can reside on a volume and is therefore essential information for the DADSM routines that allocate and release space.

The types of direct-access storage devices supported by this operating system and the number of DSCBs that will fit on a single track of each type, are:

Direct-Access Device Type	Number of DSCBs per Track
IBM 2305-1 Fixed Head Storage	18/track
IBM 2305-2 Fixed Head Storage	34/track
IBM 2314 Direct-Access Storage Facility	25/track
IBM 2319 Disk Storage	25/track
IBM 3330 Disk Storage, models 1 and 11	39/track
IBM 3333 Disk Storage and Control, models 1 and 11	39/track
IBM 3340 Direct Access Storage Facility	27/track
IBM 3344 Direct Access Storage	27/track
IBM 3350 Direct Access Storage	47/track
IBM 3375 Direct Access Storage	51/track
IBM 3380 Direct Access Storage	53/track

The DS4DSREC field of the format-4 DSCB contains a count of the number of free VTOC records (format-0 DSCBs) in the VTOC. This count is checked before each allocation. There must be enough free VTOC records for all the DSCBs required to define the data set or VSAM data space, as well as an extent or a combination of extents large enough to contain the data set or VSAM data space. The number of DSCBs needed to define a single data set or VSAM data space can be one, two, or three, depending on (1) whether it is an ISAM data set (a format-2 may be required) and (2) whether the data set has more than three extents (a format-3 DSCB is needed to list the fourth through the sixteenth extent). In addition, the DADSM Allocate routines make sure there is room for an additional format-5 in case it is necessary to create one during the allocation.

VOLUME TABLE OF CONTENTS INTEGRITY

In an operating system with only one processor, two or more tasks may require access to the same VTOC simultaneously for the purpose of reading or updating (that is, adding, deleting, or modifying DSCBs) that VTOC. If more than one processor has access to the same device or devices, it becomes necessary to protect VTOCs from being accessed while the DADSM routines are in process.

To be sure that a VTOC is not changed while the DADSM routines are in process, the DADSM routines issue RESERVE, ENQ, and DEQ macro instructions. These macro instructions provide exclusive control of the VTOC for the task issuing the macro instruction. The

RESERVE macro instruction is needed for systems in which two or more processors are processing concurrently, using the same data sets. These macro instructions provide exclusive control of the VTOC for the task issuing the macro instruction. Depending on the macro instruction, the "set-must-complete" option or the "release-must-complete" option may be specified in an operand of the macro instruction. The Allocate, Extend, Scratch, Rename, Partial Release, LSPACE, and Protect routines of DADSM issue these macro instructions. Of these routines, only Allocate, Scratch, and Partial Release use SMC=STEP in the ENQ and RESERVE macros, and RMC=STEP in the DEQ macro. The Extend routine links to the status routine (rather than issuing the ENQ macro) to obtain "step-must-complete" status, if the task that called Extend has not already done so.

The MVS catalog management routines modify the DS4AMCAT and DS4AMTIM fields of the VTOC (format-4) DSCB. These routines also issue the RESERVE, ENQ, and DEQ macro instructions to maintain exclusive control while making modifications.

Note: When operating in an environment in which direct-access storage devices are not shared among systems, the RESERVE macro instruction defaults to (acts as) an ENQ macro instruction.

DADSM Interrupt Recording Facility (DIRF)

If a system fails or a permanent I/O error occurs during allocation of space or during a routine that updates the VTOC, the VTOC will probably be in error. To make sure the error is recorded, the DADSM routines use the DADSM interrupt-recording facility (DIRF). DIRF processing involves turning on a bit in the VTOC at entry to the DADSM function, and if no I/O errors occur during DADSM processing, turning it off again at exit from that function.

This bit is called the DIRF bit and is bit 5 of the DS4VTOCI field of the format-4 DSCB. The Scratch and Partial Release routines also turn on the DIRF bit if they encounter overlapping extents in a format-5 DSCB.

The next time an attempt is made to allocate space on a volume that has the DIRF bit set, the VTOC Conversion routine is invoked by Allocate or Extend, whichever is attempting to allocate more space on the volume. The VTOC Conversion routine builds new format-5 DSCBs to represent the available space on the volume, updates the format-4 DSCB, and returns to Allocate or Extend to continue the allocation. The "Diagnostic Aids" section of OS/VS2 DADSM Logic tells how to deactivate the VTOC conversion by altering the DADSM routines.

SPECIFYING BUFFER NUMBERS FOR SAM-E 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 with SAM-E 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 which 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 (available in SAM-E Enhancements) can use installation criteria to a default buffer number for QSAM DCBs. The NCP field of the DCB must be set by the program for BSAM DCBs.

INDEX

GC26-3830-4

OS/VS2 System Programming Library: Data Management (File No. S370-30) Printed in U.S.A. GC26-3830-4

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

OS/VS2 System
Programming Library:
Data Management
GC26-3830-4

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. Comments may be written in your own language; English is not required.

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.

GC26-3830-4

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

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

OS/VS2 System Programming Library: Data Management (File No. S370-30) Printed in U.S.A. GC26-3830-4