GC26-3830-1
File No. S370-30

Systems

OS/VS2 System Programming Library: Data Management

Release 3

IBM

Second Edition (February 1975)

This edition replaces the previous edition (numbered GC26-3839-0) and makes that edition obsolete.

This edition applies to Release 3 of OS/VS2 and to all subsequent releases of that system unless otherwise indicated in new editions or technical newsletters. (Information on the Mass Storage System is only for planning purposes until the availability of that product.)

Significant system changes are summarized under "Summary of Amendments" following the list of figures. In addition, miscellaneous editorial and technical changes have been made throughout the publication. Each technical change is marked by a vertical line to the left of the change.

Information in this publication is subject to significant change. Any such changes will be published in new editions or technical newsletters. Before using the publication, consult the latest *Virtual Storage Supplement (to IBM System/360 and System/370 Bibliography),* GC20-0001, and the technical newsletters that amend the bibliography, to learn which editions and technical newsletters are applicable and current.

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

Forms for readers' comments are provided at the back of the publication. If the forms have been removed, comments may be addressed to IBM Corporation, System Development Division, LDF Publishing—Department J04, 1501 California Avenue, Palo Alto, California 94304. All comments and suggestions become the property of IBM.

PREFACE

This publication provides information on how to modify and extend the data management capabilities of the OS/VS2 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 OS/VS2 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 OS/VS2.

Other topics presented are:

- Using system macro instructions to refer to, validate, and modify system data areas
- Adding to the image library and retrieving FCB images

Prerequisite Reading

Readers are expected to understand how to:

- Code programs in assembler language as described in *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/VS Utilities*, GC35-0005, and *OS/VS Data Management Services Guide*, GC26-3783.
- Use the access methods to do input/output using the data management macros as described in *OS/VS Data Management Services Guide*, GC26-3783, and *OS/VS Data Management Macro Instructions*, GC26-3793.
- Protect data sets as described under "IEHPROGM" in *OS/VS Utilities*, GC35-0005.

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: Handbook for Debugging*, GC28-0632, which contains 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.

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:

1. Tells how to build a PASSWORD data set.
2. Describes how the system control program responds to job control language and IEHPROGM utility statements in maintaining the PASSWORD data set.
3. 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:

1. Using system mapping macros to allow you to access system control blocks and work areas using symbolic names.
2. Examining device-type information in unit control blocks (UCBs).
3. Modifying a job file control block (JFCB) before opening a data set.
4. Stopping the processing of specified I/O requests, permanently or temporarily.
5. Protecting your data sets by verifying data extent blocks.

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

CONTENTS

FIGURES

SUMMARY OF AMENDMENTS

Release 3

Mass Storage System (MSS)

Some restrictions are given for using the RDJFCB macro instruction if the data set resides on MSS virtual volumes.

MSS virtual devices are not supported by the ATLAS macro instruction.

The MSS virtual volume DEVD code for PCI appendage is provided in the EXCP section of the manual.

Release 2

Reduction of Services Done by Catalog Management Macros

These services are no longer done by catalog management macros:

- Reading a block by relative block address
- Building and deleting indexes
- Connecting and disconnecting control volumes
- Cataloging a data set by creating required index levels
- Uncataloging a data set and removing index levels

Protected Use of Appendages

The chapter "Executing Your Own Channel Programs (EXCP)" explains, under "The Authorized Appendage List (IEAAPP00)," how to prevent unauthorized access to an appendage by putting its name in SYS1.PARMLIB.

JFCB Modifications

Some JFCB modifications can compromise the security of a password-protected data set. The section "Reading and Modifying a Job File Control Block" in "System Macro Instructions" lists them.

New Purge Options

You can now specify additional options with the PURGE macro instruction by using a longer parameter list. See "PURGE—Halt or Finish I/O-Request Processing" in the chapter "System Macro Instructions."

Procedure for Retrieving an FCB Image

If you want to modify an FCB image in virtual storage before loading it into a forms control buffer, the chapter "Adding to the Image Library and Retrieving FCB Images" gives a sequence of macro instructions that you can use to read the FCB image into virtual storage. See "Retrieving an FCB Image" in that chapter.

USING CATALOG MANAGEMENT MACRO INSTRUCTIONS

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

- Retrieve information from a VS2 catalog. (Three kinds of catalogs qualify as VS2 catalogs: the VS2 master catalog, user (or private) catalogs, and CVOLs (control volumes) that are brought to an OS/VS2 system from other OS systems.)
- Catalog non-VSAM data sets.
- Uncatalog non-VSAM data sets.
- Recatalog non-VSAM data sets.

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*, GC33-4010, which contains information you will need to code programs in the assembler language.
- *OS/VS2 Access Method Services*, GC26-3841, which tells how to use programs that offer the same services as catalog management macros and additional services that catalog management macros cannot provide.
- *OS/VS2 JCL*, GC28-0692, which tells how to catalog and uncatalog data sets using job control language statements.

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

Retrieving Information from a VS2 Catalog

To retrieve information from a VS2 catalog, use the LOCATE and CAMLST macro instructions. You specify the entry you want to read into your work area by using the fully qualified name of a data set. When 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, only the first 20 volumes are returned. To get a listing of more than 20 volumes, it's necessary to use access method services.

If you specify a partially qualified data set name, no information will be returned to your work area.

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 *OS/VS2 System Programming Library: Handbook for Debugging.*)

If the named data set is stored on only one volume, bytes 252-254 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.

The format is:

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

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

NAME
this operand must be coded as shown to retrieve information from a VS2 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.

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 a non-VSAM data set resides on one volume, bytes 252-254 may contain the relative track address of the DSCB. This address is relative to the beginning of the volume. (If the return code in register 0 is not 0, the work area will contain zeros.)

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. The search for the catalog entry starts in the STEPCAT or JOBCAT (JCL-specified user catalogs), if either is specified, and continues in the master catalog. If A.B is not found in the STEPCAT, JOBCAT, or master catalog and a user catalog has an alias name *A*, it is also searched.

```
            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
```

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. If data set A.B resides on only one volume, bytes 252-254 of your area may contain the relative track address of the DSCB for 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 volume was not mounted or is not open; register 0 will be set to 0.
8	Either the catalog entry could not be found (register 0 will be set to 0), or the operator did not supply the correct password for a password-protected VSAM data set (register 0 will be set to 56).
20	A syntax error exists in the name (for example, nine characters, a double delimiter, blank name field, or a qualified name when a simple name is needed).
24	An uncorrectable error was found when processing the catalog. Register 0 will contain the catalog return code.
28	The relative track address (TTR) cannot be supplied to LOCATE.

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 is built in virtual storage 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. Nothing is read into your work area, because there are no entries in the catalog.

The format is:

[*symbol*] *listname*	**LOCATE** **CAMLST**	*list-addrx* **NAME** ,*dsname-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.

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 252-254 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. The search for the catalog entry starts in the STEPCAT or JOBCAT (JCL-specified user catalogs), if either is specified, and continues in the master catalog. If A.B is not found in the STEPCAT, JOBCAT, or master catalog and a user catalog has an alias name *A*, it is also searched.

```
            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                  LOCAREA MAY ALSO
*                                           CONTAIN A 3-BYTE
*                                           TTR
```

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, your 265-byte area contains: the catalog entry for generation data set A.PAY(-3). If data set A.PAY(-3) resides on one volume, bytes 252-254 of your area may contain the relative track address of the DSCB for that data set (relative to the beginning of the volume). In addition, 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 one of the return codes described in the previous example.

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.

The format is:

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

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

NAME
this operand must be coded as shown to retrieve information from a VS2 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.

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 252-254 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.) The search for the catalog entry starts in the STEPCAT or JOBCAT (JCL-specified user catalogs), if either is specified, and continues in the master catalog. If X.B.C is not found in the STEPCAT, JOBCAT, or master catalog and a user catalog has an alias name *X*, it is also searched.

```
          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                 LOCAREA MAY ALSO
*                                        CONTAIN 3-BYTE TTR
```

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.

After execution of these macro instructions, the 265-byte area contains: a volume list for the data set A.B.C. If data set A.B.C resides on only one volume, bytes 252-254 of your area may contain the relative track address of the DSCB for data set A.B.C (relative to the beginning of the volume).

Working with Non-VSAM 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 Non-VSAM Data Set (CATALOG and CAMLST CAT)

The format of the CATALOG and CAMLST macros is:

[*symbol*] *listname*	**CATALOG** **CAMLST**	*list-addrx* **CAT** ,*name-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
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 than 44 characters, it must be followed by blanks. The name may 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 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: Handbook for Debugging.*)

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 VTOC.

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.

If there is a STEPCAT or JOBCAT (JCL-specified user catalogs), data set A.B.C will be cataloged in whichever is specified. If there is neither, A.B.C will be cataloged in the master catalog, unless a user catalog has an alias name *A* and is connected to the master catalog; then A.B.C will be cataloged in the user catalog. A.B.C cannot be cataloged in any catalog already having an entry named *A*.

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 volume was not mounted or is not open.
8	The existing catalog structure is inconsistent with the operation requested. (If register 0 is set to 0, the catalog entry already exists.)
20	Space is not available on the catalog.
28	An uncorrectable error was encountered when processing the catalog. (If register 0 is set to 0, an error was found in the catalog parameter list.)

Uncataloging a Non-VSAM Data Set (CATALOG and CAMLST UNCAT)

When the UNCAT operand of the CAMLST macro instruction is used, a data set reference is removed.

The format of the CATALOG and CAMLST macros is:

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

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

UNCAT
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.

In the following example, the catalog entry for data set A.B.C is removed a from a VS2 catalog. The search for the catlaog entry starts in the STEPCAT or JOBCAT (JCL-specified user catalogs), if either is specified, and continues in the master catalog. If A.B.C is not found in the STEPCAT, JOBCAT, or master catalog and a user catalog has an alias name *A*, it is also searched.

```
          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 following return codes.

Code	Meaning
4	Either the required catalog volume was not mounted or is not open.
8	The existing catalog structure is inconsistent with the operation requested. (If register 0 is set to 0, there was no catalog entry for the specified data set. If register 0 is set to 60, the uncatalog request was for a VSAM data set.)
28	An uncorrectable error was encountered when processing the volume. (If register 0 is set to 0, an error was found in the catalog parameter list.)

Recataloging a Non-VSAM Data Set (CATALOG and CAMLST RECAT)

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

As in the original cataloging procedure, you must build a complete volume list in virtual storage. This volume list consists of an entry for each volume on which the data set resides. The first 2 bytes of the list indicate the number of entries in the list; the number may not be zero. Each 12-byte volume pointer consists of a 4-byte device code, a 6-byte volume serial number, and a 2-byte data set sequence number. The sequence number is always zero for direct access volumes. (Device codes are presented in *OS/VS2 System Programming Library: Handbook for Debugging.*)

The format of the CATALOG and CAMLST macros is:

[*symbol*]	**CATALOG**	*list-addrx*
listname	**CAMLST**	**RECAT**
		,*name-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.

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 following return codes.

Code	Meaning
4	Either the required catalog volume was not mounted or is not open.
8	The existing catalog structure is inconsistent with the operation requested. (If register 0 is set to 0, there was no catalog entry for the specified data set. If register 0 is set to 60, the recatalog request was for a VSAM data set.)
20	Space is not available on the catalog.
28	An uncorrectable error was encountered when processing the catalog. (If register 0 is set to 0, an error was found in the catalog parameter list.)

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*, SY26-3828.

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*, GC33-4010, which contains information you will need in order to code programs in the assembler language.
- *OS/VS Data Management Services Guide*, GC26-3783, contains a general description of direct-access device characteristics and the volume table of contents.
- *OS/VS Utilities*, GC35-0005, tells how to use utility programs to maintain the volume table of contents.
- *OS/VS2 System Programming Library: Handbook for Debugging*, GC28-0632, which contains descriptions of (1) the data set control block (DSCB) formats and (2) the contents of the fields of each DSCB.

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: Handbook for Debugging.*)

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 exceptional 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.

[*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. The name must be defined by a C-type Define Constant (DC) instruction. Note: A 44 character DSNAME or 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.
8	The format-1 DSCB was not found in VTOC of specified volume.
12	A permanent I/O error was encountered or an invalid format-1 DSCB was found when processing the specified volume.
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 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 02 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.
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.

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. Device codes are presented in *OS/VS2 System Programming Library: Handbook for Debugging.*

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. 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 half-word 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, 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.
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.
7	The specified data set could not be scratched because it was being used.

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.

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. Device codes are presented in *OS/VS2 System Programming Library: Handbook for Debugging*. 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. 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 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.
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.
7	The specified data set could not be renamed on this volume because it was being used.

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/VS Data Management Services Guide*, GC26-3783, which explains the standard procedures for I/O processing under the operating system.
- *OS/VS – DOS/VS – VM/370 Assembler Language*, GC33-4010, which contains the information necessary to code programs in the assembler language.
- *OS/VS Data Management Macro Instructions*, GC26-3793, which describes the system macro instructions that can be used in programs coded in the assembler language.
- *OS/VS2 System Programming Library: Handbook for Debugging*, GC28-0632, 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 (e.g., 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*, GA22-7000.

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 2314 or 2319 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.

Before issuing the SIO instruction for a 2314 or 2319 direct-access device, the I/O supervisor issues an initial seek, which is overlapped with other operations. You specify the seek address in the input/output block. When the

seek has completed, the I/O supervisor constructs a command chain to reissue the seek, sets the file mask specified in the data extent block, and passes control to your real channel program. (You cannot issue the initial seek or set the file mask yourself. The file mask is set to prohibit seek-cylinder commands, or, if space is allocated by tracks, seek-track 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 (i.e., 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 IFLGS 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*, GA22-7000. 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.
- Creates a *PIRL* (purged I/O restore list) and, at an offset of X'14', stores the address of the first IOB in the chain.
- Puts the address of the PIRL in the DEBUSPRG field of the data extent block.

The IOB chain reflects the order in which request queue elements are removed from their queue. If you want to inspect the chain, refer to Figure 14; it shows the format of the chain. If you want to modify the chain, additionally refer to "Modifying the IOB chain" in the chapter "System Macro Instructions."

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 issue a RESTORE macro instruction, specifying, as the only parameter, the address of the DEBUSPRG field of the data extent block. This causes execution of all the dependent channel programs. (The RESTORE macro definition and how to add it to the macro library are in "System macro Instructions.") Alternatively, to restart only particular channel programs rather than all of them, you may reissue EXCP for each channel program desired.

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 1 and described later for each appendage.
- *Register 15:* Points to the entry point of the appendage.

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

*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 1. Entry Points, Returns, and Available Work Registers for Appendages

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 1 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 TCB address from the DEB in register 12.
4. Place ECB address from the IOB in register 11.
5. Set the completion code in register 10.
6. 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 Channel Programming Notes" later in this chapter for special PCI conditions encountered with command retry.)

If you want to post the channel program from a PCI appendage, you can use the procedure described for the start-I/O appendage, unless the PCI appendage uses real storage addresses. In the latter case, follow this procedure:

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.
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 in register 13 the address of the ASCB (address space control block) for the address space in which the EXCP macro was issued. The address of the ASCB can be found in PSAAOLD in fixed low storage by the program issuing the EXCP macro or by any other appendage.
5. Put the address of the post routine—found at CVT0PT01 in the communications vector table—in register 15.
6. Go to the post routine using BALR 14,15.

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

To return control to the 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 re-executes the request. If the new seek address is within the data set's extent, the request is executed; if the new seek address is not within the data set's extent, the end-of-extent appendage is entered. If you wish to try the request in the next extent, you must move the new seek address to the location pointed to by register 6.

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

Channel-End (CHE) Appendage

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

If you use the return address in register 14 to return control to the 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 condition code may be directly tested by using a BC instruction. A CC=0 means no UEX or WLR accompanied this interruption. 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 re-execution of the channel program, you must re-initialize 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 (i.e., 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 due to 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 due to 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 due to:
   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 due to 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 due to 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 due to 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 re-execution of the channel program, you must re-initialize 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 DATAS 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*, GC28-0681.

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 3340 Disk Storage, the 3330 Disk Storage, and the 2305 Fixed Head Storage 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 re-execution. 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 re-executed.
- *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 is the correct length of the data transfer. Since the channel will not allow more data to be transferred than is specified in the count field of the CCW, this situation will occur only when reading variable records or undefined record types.

- *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/VS 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 2 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.

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.

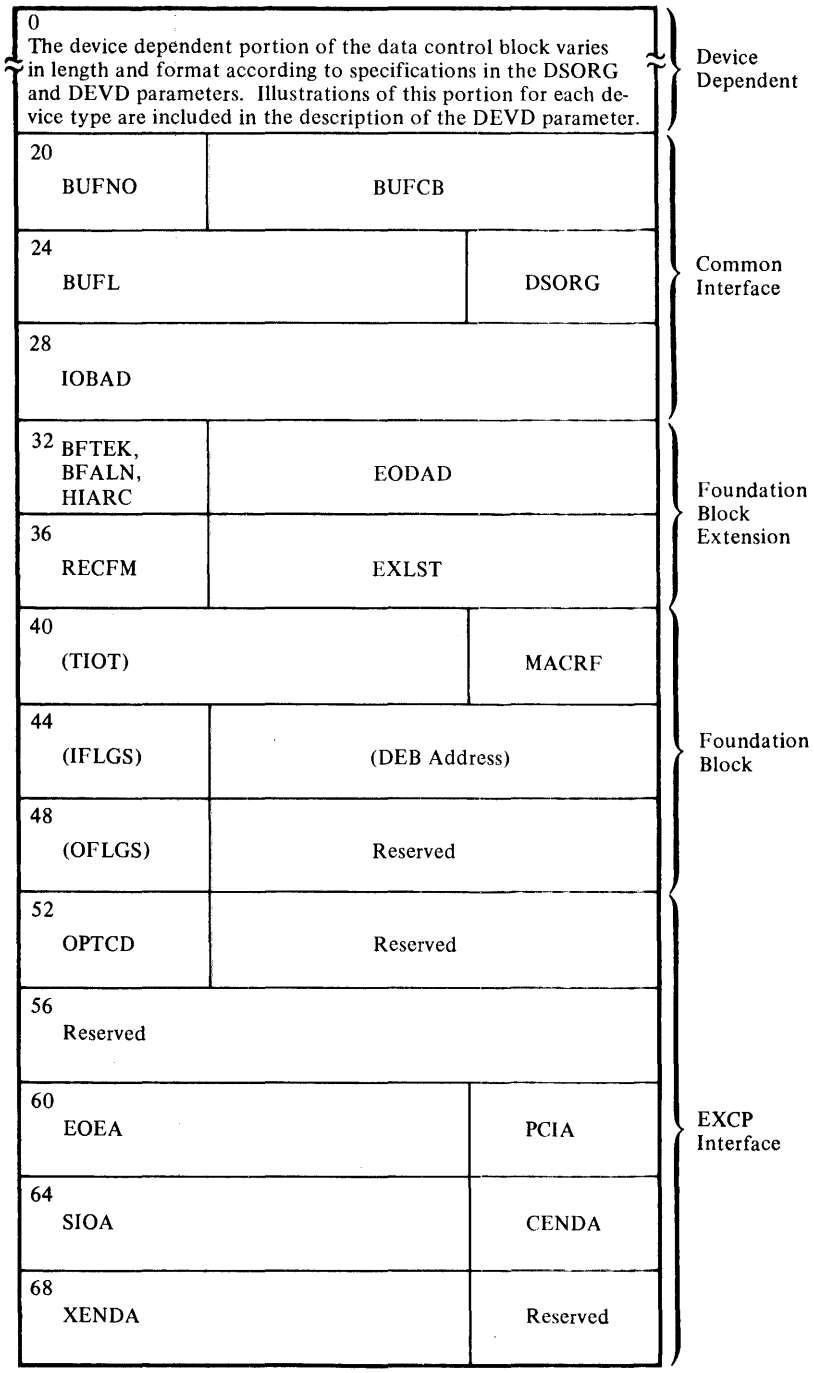


Figure 2. Data Control Block Format for EXCP (After OPEN)

REPOS= {Y | N}
Magnetic tape volumes: This parameter controls whether the DDR routine will attempt to reposition the volume after swapping devices. (To have the DDR routine attempt to reposition your tape volume, you must maintain the block count in the DCBBLKCT field.)

Y—Yes, attempt to reposition.

N—No, do not attempt to reposition.

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. (See Note A.)

PCIA=symbol
2-byte identification of a PCI appendage that you have entered into the LPA library. (See Note A.)

SIOA=symbol
2-byte identification of a SIO appendage that you have entered into the LPA library. (See Note A.)

CENDA=symbol
2-byte identification of a CHE appendage that you have entered into the LPA library. (See Note A.)

XENDA=symbol
2-byte identification of an ABE appendage that you have entered into the LPA library. (See Note A.)

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/VS 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 | DA}
the data set organization. PS means that records will be read or written sequentially; DA, nonsequentially.

For direct-access devices, if you specify PS, 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 to the last cylinder specified by these fields if the data set is re-opened for OUTPUT, INOUT, or OUTIN.

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/VS 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, i.e., the 8-byte field preceding the buffers in a buffer pool.

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:

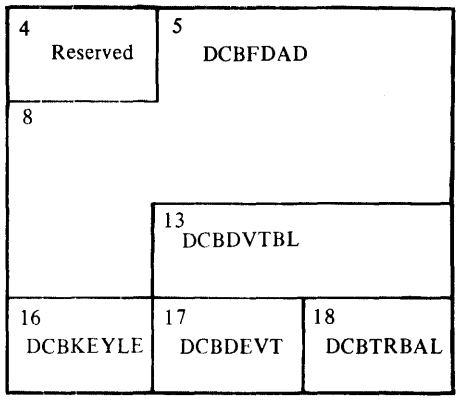
Code	Device
DA	Direct access
TA	Magnetic tape
PT	Paper tape
PR	Printer
PC	Card punch
RD	Card reader

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.

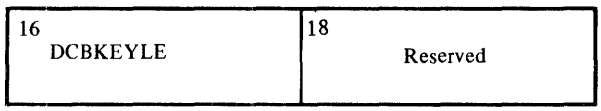
The following diagrams illustrate the device-dependent portion of the data control block for each combination of device type specified in the DEVD parameter and data set organization specified in the DSORG parameter. Fields that correspond to device-dependent parameters in addition to DEVD are indicated 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:

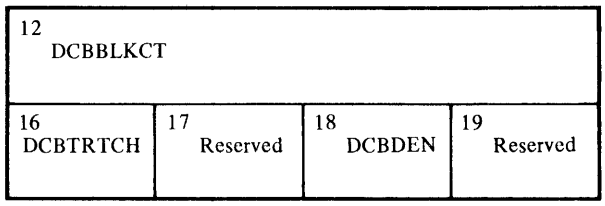


For output data sets, the system uses the contents of the full disk address (DCBFDAD) field plus one to write a file mark when the data control block is closed, provided the track balance (DCBTRBAL) field indicates that space is available. You must maintain the contents of these two fields yourself if 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:



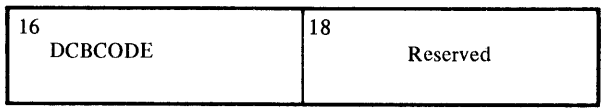
Device-dependent portion of data control block when DEVD=TA and DSORG=PS:



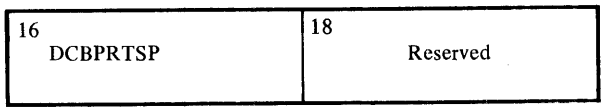
The system uses the contents of the block count (DCBBLKCT) field to write the block count in trailer labels when the data control block is closed or when the EOV macro instruction is issued. You must maintain the contents of this field yourself if the system is to have the correct block count. (Note: 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.

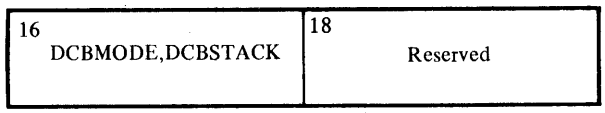
Device-dependent portion of data control block when DEVD=PT and DSORG=PS:



Device-dependent portion of data control block when DEVD=PR and DSORG=PS:



Device-dependent portion of data control block when DEVD=PC or RD and DSORG=PS:



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

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.)

option$_1$
the intended method of I/O processing of the data set. You may specify this parameter as either INPUT, RDBACK, or OUTPUT. 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.

If this parameter is omitted, INPUT is assumed.

option$_2$
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 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).

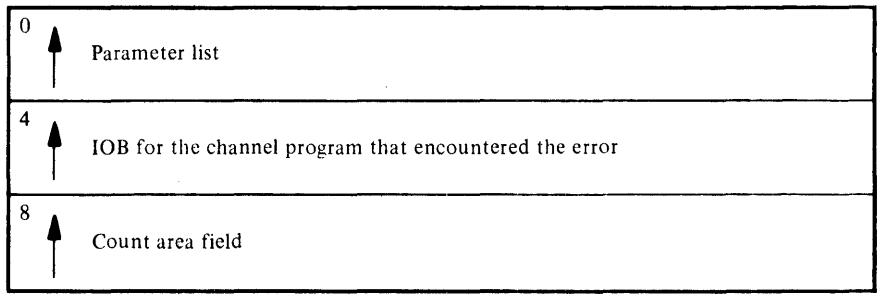
Your DCB must include the DCBRECFM field, and the field must show whether the data set is in the track overflow format. If it is, recovery from errors in last records on tracks depends on your identifying the track overflow record segments.

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:



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 re-execution 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.

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      IOBSENSO,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.

Decimal Return Code	Meaning
	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 re-execution of his 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 his 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 discription 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/VS 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 3, the diagonally-ruled areas indicate fields in which you must specify information; the shaded 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.

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.)

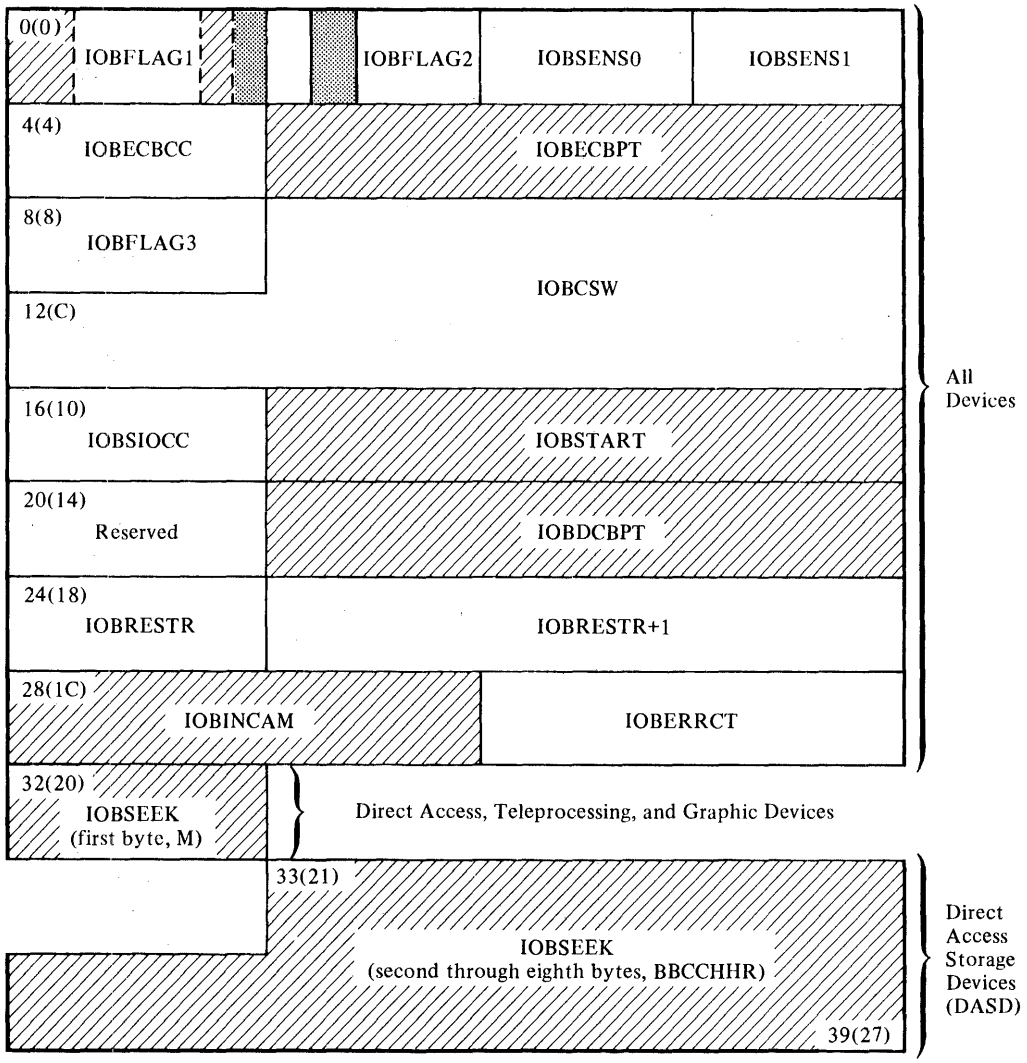


Figure 3. Input/Output Block Format

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.

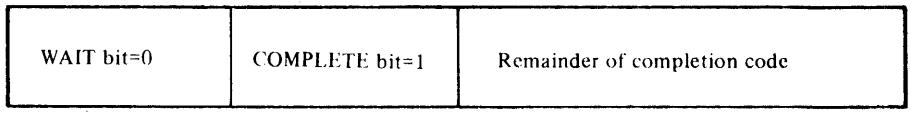
IOBSEEK (last 7 bytes, BBCCHHR)
for direct-access devices, the seek address for your channel program.

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 4). 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: Handbook for Debugging.*



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 wasn't started because of a permanent error associated with the previous request. You may reissue the EXCP macro instruction to start 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 4. 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.

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: Job Management, Supervisor, and TSO*, GC28-0682.

[*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.
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."

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 will want to consider fixing in real storage are:

1. The channel program.
2. The data areas from which your channel program will be writing and to which your channel program will be reading.
3. The PCI appendage.
4. Any control blocks or other areas referred to in your PCI appendage (for example, the DEB).

You need not fix areas that have already been fixed, such as the modules that reside in the fixed link pack area (LPA).

Page Fix (PGFX) and Start-I/O (SIO) Appendage

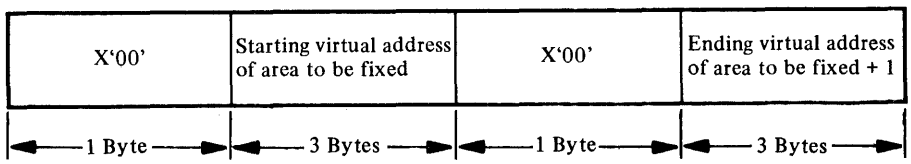
This appendage comprises two essentially independent appendages. The complete appendage can be viewed as a re-enterable subroutine having two entry points, one for the SIO appendage and one for the PGFX appendage.

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 subroutine.

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, including the boundary of any items used to create the list. 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:

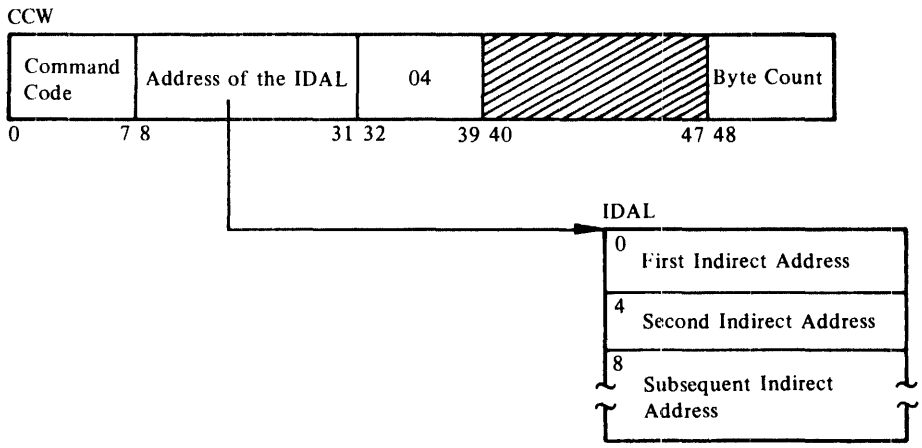


On return from your PGFX appendage to the I/O supervisor (via the return address provided in register 14), register 10 must point to the first page-fix entry and register 11 must contain the number of page-fix entries in the work area. The I/O supervisor then fixes the pages corresponding to the areas listed by the PGFX appendage. The pages remain fixed until the associated I/O request terminates.

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. You can also use the LRA to translate the virtual addresses in the IDAL to real addresses. The IDAL must contain real addresses when it is processed by the channel.



Note 1: You must put one entry in the IDAL for each page boundary your data area crosses.

Note 2: If the CCW has an IDAL address rather than a data address, bit 37 must be set to signal this to the channel.

Note 3: The 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

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/VS Data Management Services Guide*, GC26-3783, 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. You must issue OPEN for all data control blocks that are to be used by the direct-access program. Some of the procedures performed when OPEN is executed are:

- Construction of data extent block (DEB).

- Transfer of information from DD statements and data set labels to the data control block.
- Verification or creation of standard labels.
- Loading of programmer-written appendage routines.

The two parameters of the OPEN macro instruction are the address(es) of the data control block(s) to be initialized, and the intended method of I/O processing of the data set. The method of processing may be specified as either INPUT or OUTPUT; however, if neither is specified, INPUT is assumed.

XDAP—Execute Direct-Access Program

The XDAP macro instruction produces the XDAP control block (i.e., 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 register 2-12. 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 register 2-12. The *length-value* and *keylength-value* operands can be specified as an absolute expression or decimal integer or supplied in register 2-12.

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/VS Data Management Macro Instructions* to learn what these options are and how they are specified.

Control Blocks Used with XDAP

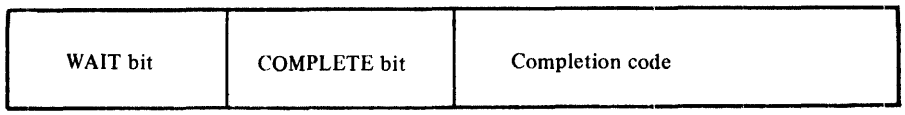
The three control blocks generated during execution of the XDAP macro instruction are described here.

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 5). 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 3). You may wish to examine the IOBSENS0, IOBSENS1, and IOBCSW fields if the ECB is posted with X'41'.



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 5. 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 6.

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 6. 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 6 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 Data Areas.*)

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.

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, IEC0SCR1. The address of this routine is in field CVT0SCR1 of the CVT, and the address of the CVT is in location 16. The routine should be invoked via a BALR 14,15 instruction.

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. *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) Programmers Guide,* GC26-3838, and *OS/VS2 Access Method Services,* GC26-3841. 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/VS Data Management Services Guide,* GC26-3783, which contains a general description of the data set protection feature.
- *OS/VS Message Library: VS2 System Messages,* GC28-1002, which contains a description of the operator messages and replies associated with the data set protection feature for VS2.
- *OS/VS2 JCL,* GC28-0692, 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,* SY26-3828, which contains a description of the PASSWORD data set record format.
- *OS/VS Utilities,* GC35-0005, 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,* GC28-0646, 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 writer.
- 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."

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/VS 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, i.e., 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 aset 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, e.g., 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 topic.

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 Figures 7 through 10 for the parameter lists and codes used to identify the functions.

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 11 contains the return codes and their meanings.

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.

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 7. 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 zero if the current password is the control password.

16 Number of volumes.
If the data set is not cataloged and you want to have it flagged as protected, you have to specify the number of volumes in this field. A zero indicates that the catalog information should be used.

17 Pointer to volume list.
If the data set is not cataloged and you want to have it flagged as protected, you have to provide the address of a list of volume serial numbers in this field. If this field is zero, the catalog information will be used.

20 Protection code.
A one-byte number indicating the type of protection: X'00' indicates that the protection is default protection (for the REPLACE function the default protection is the protection specified in the current password record of the data set); X'01' indicates that the data set is to be read and written; X'02' indicates that the data set is to be read only; and X'03' indicates that the data set can be read without a password, but a password is needed to write into the data set.

21 Pointer to new password.
The address of the password that you want to replace the current password.

24 String length.
The length of the character string (maximum 77 bytes) that you want to place in the optional information field of the password record. Set this field to zero if you don't want to add additional information.

25 Pointer to string.
The address of the character string that is going to be put in the optional information field of the password record. Set the address to zero if you don't want to add additional information.

Figure 8. 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 zero 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 9. 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 10. Parameter List for LIST Function

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	The PASSWORD data set was full.
28	The validity check of the buffer address failed.
*32	The LOCATE macro failed. LOCATE's return code is in register 1, and the number of indexes searched is in register 0.
*36	The OBTAIN macro failed. OBTAIN's return code is in register 1.
*40	The DSCB could not be updated.
44	The PASSWORD data set does not exist.
*48	Tape data set cannot be protected.
*52	Data set in use.

*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.

**For this return code, a message is written to the console indicating that the PASSWORD data set is full.

Figure 11. Return Codes from the PROTECT Macro Instruction

SYSTEM MACRO INSTRUCTIONS

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

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

- *OS/VS – DOS/VS – VM/370 Assembler Language*, GC33-4010, which contains the information necessary to code programs in the assembler language.
- *OS/VS2 System Programming Library: Handbook for Debugging*, GC28-0632, 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)

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. When coding these instructions, you must precede each with a DSECT statement.

The CVT, IEFUCBOB, and IEFJFCBN macro definitions are in a distribution library named SYS1.AMACLIB. Before you can issue the macros, you must copy them from SYS1.AMACLIB into SYS1.MACLIB. The IEBCOPY utility can be used to copy the macros.

The fields in these blocks are shown and described in *OS/VS2 System Programming Library: Handbook for Debugging.*

IEFUCBOB—Mapping the UCB

This macro instruction defines the symbolic names of all fields in the unit control block (UCB). Code this macro instruction with blank name and operand fields, and precede it with a DSECT statement.

The format is:

[*symbol*]	**DSECT** **IEFUCBOB**	

IEFJFCBN—Mapping the JFCB

This macro instruction defines the symbolic names of all fields in the job file control block (JFCB). Code this macro instruction with blank name and operand fields, and precede it with a DSECT statement.

The format is:

[*symbol*]	**DSECT** **IEFJFCBN**	

CVT—Mapping the CVT

This macro instruction defines the symbolic names of all fields in the communication vector table (CVT). Code this macro instruction with blank name and operand fields, and precede it with a DSECT statement.

The format is:

[*symbol*]	**DSECT** **CVT**	

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 address 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 address of an area into which the device information is to be placed. The area can be one, 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 information are placed in your area.

RPS

If RPS is specified, DEVTAB must also be specified. The RPS parameter causes one additional full word of RPS information to be included with the DEVTAB information.

Note: Any reference for a DUMMY data set in the DEVTYPE macro instruction will cause eight bytes of 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.

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: Handbook for Debugging.*

Word 1

Maximum blocksize. For direct-access devices, this value is the maximum size of an unkeyed block; 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, 3330-1, and 3340 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.
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.

Word 4

Byte 0 — Block overhead, block without key—the number of bytes to be subtracted if a block is not keyed.

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 1-3 — Reserved.
- 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 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

Output for Each Device Type

Device†	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

Device†	Maximum Record Size (Word 1, In Decimal)	DEVTAB (Words 2, 3, and 4, In Hexadecimal)	RPS (Word 5, In Hexadecimal)
1287 Optical Reader	0	Not Applicable	Not Applicable
1288 Optical Reader	0	Not Applicable	Not Applicable
3886 Optical Reader	0	Not Applicable	Not Applicable
3890 Document Processor	0	Not Applicable	Not Applicable
1419/1275 Reader/Sorter	0	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*	Not Applicable	Not Applicable
1403 w/UCS	120*	Not Applicable	Not Applicable
1404 Printer	120*	Not Applicable	Not Applicable
1443 Printer	120*	Not Applicable	Not Applicable
3211 Printer	132*	Not Applicable	Not Applicable
2671 Paper Tape Reader	32760	Not Applicable	Not Applicable
1052 Printer-Keyboard	130	Not Applicable	Not Applicable
1053 Printer		Not Applicable	Not Applicable
3210 Printer-Keyboard	130	Not Applicable	Not Applicable
3215 Printer-Keyboard	130	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	14138	0030000838E80278CA080200	02985A57
2305-2 Fixed-Head Storage	14660	006A00083A0A01215B080200	0140B4B1
3330 Disk Storage	13030	019B0013336DBFBF38000200	00ED807C
3330V MSS Virtual device	13030	019B0013336DBFBF38000200	00ED807C
3330-1 Disk Storage	13030	032F0013336DBFBF38000200	00ED807C
3340 Disk Storage(35 megabytes)	8368	015D000C2157F2F24B000200	0125403D
(70 megabytes)	8368	02BA000C2157F2F24B000200	0125403D
2250-1 Display Unit		Not Applicable	Not Applicable
2250-2 Display Unit		Not Applicable	Not Applicable
2253-3 Display Unit		Not Applicable	Not Applicable

Legend

CI-card image feature, d.c.-data conversion, d.d.-dual density, p.e.-phase encoding, UCS-universal character set, w/-with

*Although certain models can have a larger line size, the minimum line size is assumed.

†Device codes are presented in *OS/VS2 System Programming Library: Handbook for Debugging.*

Communication Equipment	Record Size
1030,1050,83B3, TWX,2250, S360	Not Applicable
1060,115A,1130	Not Applicable
2780	Not Applicable
2740	Not Applicable

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.

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: Handbook for Debugging.*) 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 JFCBTSDM field of the JFCB to X'08' before you issued the OPEN macro instruction.

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 Note 1).
- Moving the secondary allocation quantity from the DSCB into the JFCB (see Note 1).
- Moving the DCB fields from the DSCB into the JFCB.
- Adding volume serial numbers to the JFCB (see Note 1).
- Modifying the data set sequence number field in the JFCB.
- Modifying the number-of-volumes field in the JFCB (see Note 1).

Note 1: 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.

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.

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/VS 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 his 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, $option_1$, and $option_2$ operands of the OPEN macro instruction, as shown in *OS/VS Data Management Macros*)

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: 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; $option_2$ 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        OF,176C
            .
            .
            .
LSTB        DS        0F
            .
            .
            .
```

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/VS Data Management Services Guide.* The format of the job file control block exit list entry is as follows:

Type 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, or UPDAT), 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: Job Management, Services, and TSO,* GC28-0682.

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/VS Data Management Services Guide,* is available when a RDJFCB macro instruction is issued.

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 his own routine in supervisor state to read from and write to data sets to which he would otherwise be denied access.

To guarantee protection of the DEB, the DEBCHK macro instruction is provided. 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.

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 O/C/EOV Logic*, SY26-3827.

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***))}]** **[,MF=L]**

cbaddr 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.

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.

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 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*, SY26-3823, 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 Figures 12 and 13. 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 12. 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 13. 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)
specifies the 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 2-12 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 14.

Byte	Contents
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 selection of purge options. (If the caller is in problem state, these actions are taken regardless of the bit setting.) .... 1... Ensure that I/O requests will be reprocessed (restored) under their original TCB. (If zero and this byte is meaningful (bit 7 of byte 0 is on), the I/O requests will be reprocessed under the TCB of the program making the 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 resotred, 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 14). (The address of the PIRL will be at the location pointed to by bytes 9-11 of the purge parameter list.)

An IOB representing an I/O request that completed in error can be added to the IOB chain as a means of retrying the I/O request. Only an abnormal-end appendage, however, can add an IOB to an IOB chain; it cannot add the IOB directly, but must call a system routine to do it.

To get the address of the routine, an abnormal-end appendage must:

1. Get the address of the I/O supervisor appendage table from an offset of X'14' bytes into the CVT.
2. Pick up the routine's address from an offset of X'14' bytes into the I/O supervisor appendage table.

The routine requires this input:

- Zeros in register 0.
- The address of an RQE in register 1.
- The address of a 16-word save area in register 13.
- A return address in register 14.
- Its entry-point address in register 15.

When the routine returns to the abnormal-end appendage, the IOB associated with the RQE that was passed will have been added to the end of the IOB chain.

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)
the same address you specified at byte 9 of the purge parameter list.

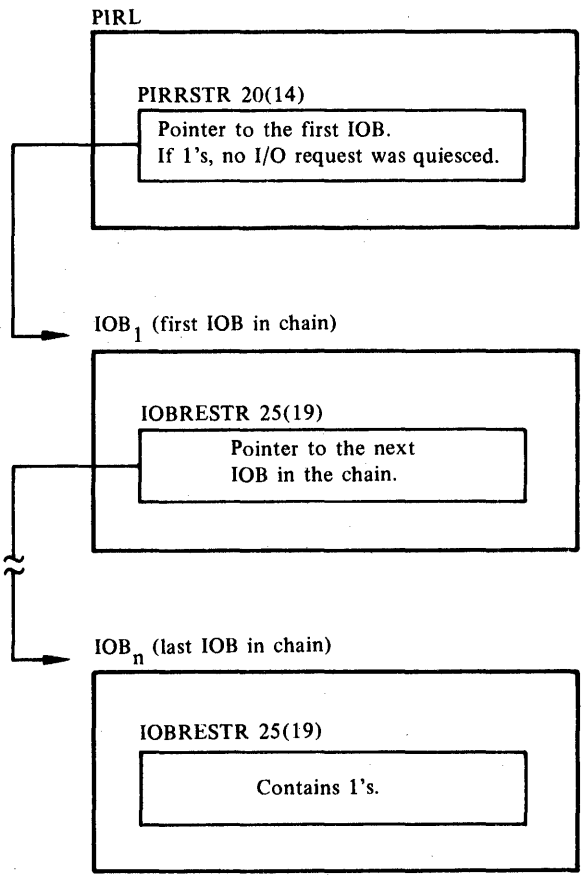


Figure 14. The PIRL and IOB Chain

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.

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

- *IBM 2821 Control Unit*, GA24-3312, contains the information necessary to create a user-designed chain/train for the 1403 Printer.
- *OS/VS Data Management Macro Instructions*, GC26-3793, describes the SETPRT macro instruction that loads a UCS image and an FCB image into their respective buffers.
- *OS/VS2 JCL*, GC28-0692, 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 3211 Printer and 3811 Control Unit Component Description*, GA24-3543, contains the information necessary to create a user-designed train for the 3211 Printer.

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 or UCS2 for the 3211 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. X'80' indicates a default image; X'00' indicates that the image is not to be used as a default. (Default images may be used by the system for jobs that do not request a specific image.)
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 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 (byte 512) of X'00'.) 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.

The following code is an example of adding a 1403 UCS image, IM, to the image library.

```
//ADDIM      JOB MSGLEVEL=1
//STEP       EXEC PROC=ASMFCL,PARM.ASM='NODECK,LOAD',
//                PARM.LKED='LIST,NCAL,NE,OL'
//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
```

The following example shows the code used to add a 3211 UCS image (IMG) to the image library. A 3211 UCS image has 432 characters, followed by 15 bytes of X'00', 64 bytes of associative bits, and a reserved byte (byte 512) of X'00'. 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 chain, see *IBM 3211 Printer, 3216 Interchangeable Train Cartridge, and 3811 Printer Control Unit Component Description and Operator's Guide*, GA24-3543.

Note: 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.

```
//ADDIMG        JOB MSGLEVEL=1
//STEP          EXEC PROC=ASMFCL,PARM.ASM='NODECK,LOAD',
//                   PARM.LKED='LIST,NCAL,OL'
//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<.+IHGFEDCBA*$-RQPONMLKJ%,&&ZYXWVUTS/@#098765432'
                DC   C'1<.+IHGFEDCBA*$-RQPONMLKJ%,&&ZYXWVUTS/@#098765432'
                DC   C'1<.+IHGFEDCBA*$-RQPONMLKJ%,&&ZYXWVUTS/@#098765432'
                DC   C'1<.+IHGFEDCBA*$-RQPONMLKJ%,&&ZYXWVUTS/@#098765432'
                DC   C'1<.+IHGFEDCBA*$-RQPONMLKJ%,&&ZYXWVUTS/@#098765432'
                DC   C'1<.+IHGFEDCBA*$-RQPONMLKJ%,&&ZYXWVUTS/@#098765432'
                DC   C'1<.+IHGFEDCBA*$-RQPONMLKJ%,&&ZYXWVUTS/@#098765432'
                DC   15X'00'         RESERVED FIELD, BITS 433-447
*  THE FOLLOWING FOUR DC INSTRUCTIONS DEFINE THE ASSOCIATIVE BITS,
*  UCSB BYTE POSITIONS 448-511
                DC   X'C0101010101010101010100040404240004010'
                DC   X'101010101010101010100040410000404010 10'
                DC   X'1010101010100040400000001010101010 10'
                DC   X'10101010004040444800'
                DC   X'00'       RESERVED FIELD, BYTE 512
                END
/*
//LKED.SYSLMOD      DD      DSNAME=SYS1.IMAGELIB(UCS2IMG),DISP=OLD
```

Adding an FCB Image to the Image Library

Two standard FCB images, STD1 and STD2, can be included in SYS1.IMAGELIB during system generation for a 3211 printer. STD1 prints six lines per inch on a 8 1/2 inch form. STD2 prints six lines per inch on an eleven 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 "S"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,* GA24-3543.

   The form image begins with lines per inch byte and must be as long as the form. For example, if you are printing six lines per inch on an eleven inch form, the form image must be 66 bytes long. The lines per inch byte defines the number of lines per inch and a channel:

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

```
//ADDFCB        JOB    MSGLEVEL=1
//STEP          EXEC PROC=ASMFCL,PARM.ASM='NODECK,LOAD',
//                     PARM.LKED='LIST,NCAL,OL'
//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
```

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/VS 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.

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.

INDEX

For additional information about any subject listed in this index, refer to the publications that are listed under the same subject in *OS/VS2 Master Index*, GC28-0693.

E

F

G

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

123

OS/VS2 System Programming Library: Data Management (File No. S370-30) Printed in U.S.A. GC26-3830-1

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

OS/VS2 System Programming Library:
Data Management
GC26-3830-1

Reader's Comment Form

Your comments about this publication will help us to improve it for you. Comment in the space below, giving specific page and paragraph references whenever possible. All comments become the property of IBM.

Please do not use this form to ask technical questions about IBM systems and programs or to request copies of publications. Rather, direct such questions or requests to your local IBM representative.

If you would like a reply, please provide your name, job title, and business address (including ZIP code).

Fold on two lines, staple, and mail. No postage necessary if mailed in the U.S.A. (Elsewhere, any IBM representative will be happy to forward your comments.) Thank you for your cooperation.

Fold and Staple

First Class Permit
Number 439
Palo Alto, California

Business Reply Mail

No postage necessary if mailed in the U.S.A.

Postage will be paid by:

IBM Corporation
System Development Division
LDF Publishing—Department J04
1501 California Avenue
Palo Alto, California 94304

Fold and Staple

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

OS/VS2 System Programming Library: Data Management (File No. S370-30) Printed in U.S.A. GC26-3830-1

OS/VS2 System Programming Library:
Data Management
GC26-3830-1

Reader's Comment Form

Your comments about this publication will help us to improve it for you. Comment in the space below, giving specific page and paragraph references whenever possible. All comments become the property of IBM.

Please do not use this form to ask technical questions about IBM systems and programs or to request copies of publications. Rather, direct such questions or requests to your local IBM representative.

If you would like a reply, please provide your name, job title, and business address (including ZIP code).

Fold on two lines, staple, and mail. No postage necessary if mailed in the U.S.A. (Elsewhere, any IBM representative will be happy to forward your comments.) Thank you for your cooperation.

Fold and Staple

First Class Permit
Number 439
Palo Alto, California

Business Reply Mail

No postage necessary if mailed in the U.S.A.

Postage will be paid by:

IBM Corporation
System Development Division
LDF Publishing—Department J04
1501 California Avenue
Palo Alto, California 94304

Fold and Staple

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

OS/VS2 System Programming Library: Data Management (File No. S370-30) Printed in U.S.A. GC26-3830-1