File No. S360-20
Form C28-6550-2

OS

IBM Systems Reference Library

IBM System/360 Operating System

System Programmer's Guide

This publication describes:

- How to maintain the data set cataloging facility of the Operating System and the volume-table-of-contents (VTOC) of a direct-access volume.
- How to modify or extend capabilities of the Operating System in the areas of job and job-step accounting, processing of magnetic tape labels, and SVC routines.
- How to implement the data set protection feature of the Operating System.
- How to write a shift initiator program for use when the multiprogramming with a fixed number of tasks (MFT) option is incorporated in the Operating System.
- How to use the EXCP (Execute a Channel Program) and XDAP (Execute a Direct-Access Program) macro-instructions, the Resident Access-Method and BLDL Table options, and the tracing routine.

Also included are descriptions of system macro-instructions used in modifying the control program.

PREFACE

This publication consists of self-contained chapters, each of which provides information on how to modify or extend the capabilities of the IBM System/360 Operating System control program. Although the information in one chapter is sometimes related to information in another, all chapters have been written as separate and complete units. Each chapter contains its own introductory section and list of pre-requisite publications. This organization has been used to reduce cross-referencing and to facilitate the addition of new chapters.

Third Edition (March 1967)

This publication is a revision of Forms C28-6550-0 and C28-6550-1 and obsoletes the previous editions. This publication retains the content of the previous editions, as amended by Technical Newsletters N28-2145, N28-2157, N28-2162, N28-2183, N28-2188, and N28-2207, with the following exceptions:

The format and field description of the Data Set Control Blocks, the Data Extent Block, and the Job File Control Block are deleted.

The chapter Cataloged Procedures is deleted.

Additions and changes to existing material that are effective as of this edition, are indicated by a vertical bar in the left margin.

Specifications contained herein are subject to change from time to time. Any such change will be reported in subsequent revisions or Technical Newsletters.

This publication was prepared for production using an IBM computer to update the text and to control the page and line format. Page impressions for photo-offset printing were obtained from an IBM 1403 Printer using a special print chain.

Copies of this and other IBM publications can be obtained through IBM Branch Offices.

A form for readers' comments appears at the back of this publication. It may be mailed directly to IBM. Address any additional comments concerning this publication to the IBM Corporation, Programming Systems Publications, Department D58, PO Box 390, Poughkeepsie, N. Y. 12602

© International Business Machines Corporation 1966, 1967

CONTENTS

ILLUSTRATIONS

FIGURES

TABLES

MAINTAINING THE CATALOG AND THE VOLUME TABLE OF CONTENTS

This chapter provides detailed information on how to maintain and modify the catalog and volume table of contents.

Before reading this chapter, you should be familiar with the information contained in the prerequisite publications listed below.

Documentation of the internal logic of the routines used to maintain and modify the catalog and volume table of contents can be obtained through your IBM Branch Office.

PREREQUISITE PUBLICATIONS

The IBM System/360 Operating System: Assembler Language publication (Form C28-6514) contains the information necessary to code programs in the assembler language.

The IBM System/360 Operating System: Data Management publication (Form C28-6537) contains a general description of the structure of catalog indexes, as well as a brief discussion of the volume table of contents (VTOC).

The IBM System/360 Operating System: Control Program Services publication (Form C28-6541) contains an explanation of the notation conventions used to describe the macro-instructions contained in this chapter.

The IBM System/360 Operating System: System Control Blocks publication (Form C28-6628) contains format and field descriptions of the system control blocks referred to in this chapter.

RECOMMENDED PUBLICATIONS

The IBM System/360 Operating System: Utilities publication (Form C28-6586) describes how to maintain and modify the catalog and the volume table of contents through the use of utility programs.

MAINTAINING THE CATALOG AND THE VOLUME TABLE OF CONTENTS

This chapter describes how to maintain and modify the catalog and the volume table of contents through the use of macro-instructions. Most of the maintenance and modification functions can also be performed using utility statements. The utility statements are described in the publication IBM System/360 Operating System: Utilities.

The functions you can perform using the macro-instructions are described in text, and the formats of the macro-instructions are tabulated on a fold-out sheet at the back of this chapter. The chart on the fold-out sheet associates the function described in text with the macro-instructions needed to perform the function. You should keep the fold-out sheet open when reading the text.

The functions that are described in text are as follows:

- How to read a block from the catalog.
- How to build an index.
- How to build a generation index.
- How to delete an index.
- How to assign an alias.
- How to delete an alias.
- How to connect control volumes.
- How to disconnect control volumes.
- How to catalog a data set.
- How to remove data set references from the catalog.
- How to recatalog a data set.
- How to read a data set control block from the volume table of contents.
- How to delete a data set.
- How to rename a data set.

Accompanying the function descriptions in text are coding examples and programming notes; exceptional-return condition codes for the macro-instructions are tabulated on the back of the fold-out sheet.

HOW TO READ A BLOCK FROM THE CATALOG

To read either an index block or a block indicating the volumes on which a data set is stored (volume-list block), you use the LOCATE and CAMLST macro-instructions. There are two ways to specify the block that you want read into main storage: by using the name of the index level or data set, or by using the block's location relative to the beginning of the catalog (TTR).

Specifying the Name of an Index Level or Data Set

If you specify an index level name, the first block of the named index is read into main storage, and an exceptional return code is set. Index block formats are contained in Appendix A of this chapter.

If you specify a data set name, a 256-byte volume-list block is read into main storage. The block contains up to 20 volume pointers, each of which points to a volume on which part of the data set is stored. The first two bytes of the block contain the number of volume pointers for the data set. Each volume pointer is a 12-byte field that contains a 4-byte device code, a 6-byte volume serial number, and a 2-byte data set sequence number. (Device codes are contained in Appendix B of this chapter.)

If the named data set is stored on more than 20 volumes, bytes 253-255 of the block contain the relative track address of the next block of volume pointers. Byte 256 contains a binary zero.

Example: In the following example, the list of volumes that contain data set A.B is read into main storage. The search for the volume-list block starts on the system residence volume.

```
┌─────────┬──────────┬──────────────────────────────────────────────┐
| Name    | Operation| Operand                                      |
├─────────┼──────────┼──────────────────────────────────────────────┤
|         | LOCATE   | INDAB                  READ VOLUME-LIST BLOCK FOR |
|         | Check Exceptional Returns       CATALOGED DATA SET A.B INTO|
| INDAB   | CAMLST   | NAME,AB,,LOCAREA       MAIN STORAGE AREA NAMED     |
| AB      | DC       | CL44'A.B'              LOCAREA. LOCAREA ALSO       |
| LOCAREA | DS       | 0D                     CONTAINS 3-BYTE TTR AND     |
|         | DS       | 265C                   6-BYTE SERIAL NUMBER        |
└─────────┴──────────┴──────────────────────────────────────────────┘
```

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 volume-list block by using the name of a data set. AB, the second operand, specifies the main 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 main storage.

After execution of these macro-instructions, the 265-byte area contains: the 256-byte volume-list block for data set A.B, the 3-byte relative track address (TTR) of the block following the one read into main storage, and the 6-byte serial number of the volume on which the block was found.

Specifying the Name of a Generation Data Set

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 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 block is read into main 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. A volume-list block is not read, since none exists for these data sets.

Example: In the following example, the list of volumes that contain generation data set A.PAY(-3) is read into main storage. The search for the volume-list block starts on the system residence volume.

```
| Name     | Operation | Operand                                                |
|----------|-----------|--------------------------------------------------------|
|          | LOCATE    | INDGX                  READ VOLUME-LIST BLOCK FOR      |
|          | Check Exceptional Returns     DATA SET A.PAY(-3) INTO      |
| INDGX    | CAMLST    | NAME,APAY,,LOCAREA     MAIN STORAGE AREA NAMED         |
| APAY     | DC        | CL44'A.PAY(-3)'        LOCAREA. LOCAREA ALSO CON-      |
| LOCAREA  | DS        | 0D                     TAINS 3-BYTE TTR AND            |
|          | DS        | 265C                   6-BYTE SERIAL NUMBER            |
```

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 volume-list block by using the name of a data set. APAY, the second operand, specifies the main 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 in main storage.

After execution of these macro-instructions, the 265-byte area contains: the 256-byte volume-list block for generation data set A.PAY(-3), the 3-byte relative track address (TTR) of the block following the one read into main storage, and the 6-byte serial number of the volume on which the block was found. 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.

Specifying a Name Using an Alias

For each of the preceding functions, you can specify an alias as the first name in the qualified name of an index level, data set, or generation data set. Each function is performed exactly as previously described, with one exception: the alias name specified is replaced by the true name.

Specifying by TTR

You can read any block in the catalog by specifying, in the form TTR, the identification of the block and its location relative to the beginning of the catalog. TT is the number of tracks defining the position, relative to the beginning of the catalog, of the track on which the block to be read resides; R is the identification of the block on that track. (Formats of each type of catalog block are contained in Appendix A of this chapter.)

Example: In the following example, the block at the location indicated by TTR is read into main storage. The specified block is in the catalog on the system residence volume.

```
| Name     | Operation | Operand                                                |
|----------|-----------|--------------------------------------------------------|
|          | LOCATE    | BLK                    READ A BLOCK INTO MAIN          |
|          | Check Exceptional Returns     STORAGE AREA NAMED LOCAREA   |
| BLK      | CAMLST    | BLOCK,TTR,,LOCAREA                                     |
| TTR      | DC        | H'5'                   RELATIVE TRACK 5                |
|          | DC        | X'03'                  BLOCK 3 ON TRACK                |
| LOCAREA  | DS        | 0D                     LOCAREA ALSO CONTAINS 3-BYTE    |
|          | DS        | 265C                   TTR AND 6-BYTE SERIAL NO.       |
```

The LOCATE macro-instruction points to the CAMLST macro-instruction. BLOCK, the first operand of CAMLST, specifies that the system is to search the catalog for the block indicated by TTR, the second operand. LOCAREA, the fourth operand, specifies a 265-byte area you have reserved in main storage.

After execution of these macro-instructions, the 265-byte area contains: the 256-byte index block, the 3-byte relative track address (TTR) of the block following the one read into main storage, and the 6-byte serial number of the volume on which the block was found.

HOW TO BUILD AN INDEX

To build a new index structure and add it to the catalog, you must create each level of the index separately. You create each level of the index by using the INDEX and CAMLST macro-instructions.

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

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

```
┌─────────┬─────────┬──────────────────────────────────────────────┐
│ Name    │ Operation│ Operand                                     │
├─────────┼─────────┼──────────────────────────────────────────────┤
│         │ INDEX   │ INDEXA              BUILD INDEX A            │
│         │ Check Exceptional Returns                              │
│         │ INDEX   │ INDEXB              BUILD INDEX STRUCTURE A.B│
│         │ Check Exceptional Returns                              │
│         │ INDEX   │ INDEXC              BUILD INDEX STRUCTURE A.B.C
│         │ Check Exceptional Returns                              │
│ INDEXA  │ CAMLST  │ BLDX,ALEVEL,VOLNUM                           │
│ INDEXB  │ CAMLST  │ BLDX,BLEVEL,VOLNUM                           │
│ INDEXC  │ CAMLST  │ BLDX,CLEVEL,VOLNUM                           │
│ VOLNUM  │ DC      │ CL6'000045'         VOLUME SERIAL NUMBER     │
│ ALEVEL  │ DC      │ CL2'A'              INDEX STRUCTURE NAMES    │
│ BLEVEL  │ DC      │ CL4'A.B'            FOLLOWED BY BLANKS       │
│ CLEVEL  │ DC      │ CL6'A.B.C'          WHICH DELIMIT FIELDS     │
└─────────┴─────────┴──────────────────────────────────────────────┘
```

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

HOW TO BUILD A GENERATION INDEX

You build a generation index by using the INDEX and CAMLST macro-instructions. All higher levels of the index must exist. If the higher levels of the index are not in the catalog, you must build them. How to build an index has been explained previously. In the following example, the generation index D is built on the control volume whose serial number is 000045. The higher level indexes A.B.C already exist. When the number of generation data sets in the generation index D exceeds four, the oldest data set in the group is uncataloged and scratched.

```
┌─────────┬─────────┬──────────────────────────────────────────────┐
│ Name    │ Operation│ Operand                                     │
├─────────┼─────────┼──────────────────────────────────────────────┤
│         │ INDEX   │ GENINDX                    BUILD GENERATION  │
│         │ Check Exceptional Returns          INDEX             │
│ GENINDX │ CAMLST  │ BLDG,DLEVEL,VOLNUM,,DELETE,,4                │
│ DLEVEL  │ DC      │ CL8'A.B.C.D'               BLANK DELIMITER   │
│ VOLNUM  │ DC      │ CL6'000045'                                  │
└─────────┴─────────┴──────────────────────────────────────────────┘
```

The INDEX macro-instruction points to the CAMLST macro-instruction. BLDG, the first operand of CAMLST, specifies that a generation index be built. DLEVEL, the second operand, specifies the main storage location of an area into which you have placed the fully qualified name of a generation index. VOLNUM, the third operand, specifies the main storage location of an area into which you have placed the 6-byte serial number of the volume on which the generation index is to be built. DELETE, the fifth operand, specifies that all data sets dropped from the generation data group are to be deleted. The final operand, 4, specifies the number of data sets that are to be maintained in the generation data group.

Note: A model DSCB for the generation data set must be placed on the control volume containing the index prior to creation of the data set. The system will take information from the model DSCB when you create a data set for the group. A model DSCB is created by specifying, in a DD statement: the name of the data set; zero space allocation, i.e., SPACE=(TRK,(0)); and appropriate DCB=parameters.

HOW TO DELETE AN INDEX

You can delete any number of index levels from an existing index structure. Each level of the index is deleted separately. You delete each level of the index by using the INDEX and CAMLST macro-instructions.

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

Example: In the following example, index level C is deleted from index structure A.B.C. The search for the index level starts on the system residence volume.

```
┌─────────┬─────────┬──────────────────────────────────────────────┐
│ Name    │ Operation│ Operand                                     │
├─────────┼─────────┼──────────────────────────────────────────────┤
│         │ INDEX   │ DELETE                DELETE INDEX LEVEL C FROM│
│         │ Check Exceptional Returns       INDEX STRUCTURE A.B.C │
│ DELETE  │ CAMLST  │ DLTX,LEVELC                                  │
│ LEVELC  │ DC      │ CL6'A.B.C'            ONE BLANK FOR DELIMITER │
└─────────┴─────────┴──────────────────────────────────────────────┘
```

The INDEX macro-instruction points to the CAMLST macro-instruction. DLTX, the first operand of CAMLST, specifies that an index level be deleted. LEVELC, the second operand, specifies the main storage location of an area into which you have placed the fully qualified name of the index structure whose lowest level is to be deleted.

HOW TO ASSIGN AN ALIAS

You assign an alias to an index level by using the INDEX and CAMLST macro-instructions. An alias can be assigned only to a high level index; e.g., index A of index structure A.B.C can have an alias, but

index B cannot. Assigning an alias to a high level index effectively provides aliases for all data sets cataloged under that index.

Example: In the following example, index level A is assigned an alias of X. The search for the index level starts on the system residence volume.

```
| Name     | Operation | Operand                                          |
|          | INDEX     | ALIAS                  BUILD AN ALIAS FOR A HIGH |
|          | Check Exceptional Returns      LEVEL INDEX                    |
| ALIAS    | CAMLST    | BLDA,DSNAME,,DSALIAS                          |
| DSNAME   | DC        | CL8'A'                 MUST BE 8-BYTE FIELDS  |
| DSALIAS  | DC        | CL8'X'                                        |
```

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

HOW TO DELETE AN ALIAS

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

Example: In the following example, alias X, previously assigned as an alias for index level A, is deleted. The search for the alias starts on the system residence volume.

```
| Name     | Operation | Operand                                          |
|          | INDEX     | DELALIAS               DELETE AN ALIAS FOR A  |
|          | Check Exceptional Returns      HIGH LEVEL INDEX               |
| DELALIAS | CAMLST    | DLTA,ALIAS                                    |
| ALIAS    | DC        | CL8'X'                 MUST BE 8-BYTE FIELD   |
```

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

HOW TO CONNECT CONTROL VOLUMES

You connect two control volumes by using the INDEX and CAMLST macro-instructions. If a control volume is to be connected to the system residence volume, you need supply only the serial number of the volume to be connected and the name of a high level index associated with the volume to be connected.

If a control volume is to be connected to a control volume other than the system residence volume, you must supply the serial numbers of both volumes and the name of a high level index associated with the volume to be connected.

The result of connecting control volumes is that the volume serial number of the control volume connected and the name of a high level

index are entered into the volume index of the volume to which it was connected. This entry is called a control volume pointer. A control volume pointed to by a control volume cannot, in turn, point to another control volume.

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

```
┌─────────┬─────────┬──────────────────────────────────────────────┐
│ Name    │Operation│ Operand                                      │
├─────────┼─────────┼──────────────────────────────────────────────┤
│         │ INDEX   │ CONNECT                      CONNECT TWO CON-│
│         │Check Exceptional Returns               TROL VOLUMES WHOSE│
│ CONNECT │ CAMLST  │ LNKX,INDXNAME,OLDCVOL,NEWCVOL SERIAL NUMBERS ARE│
│ INDXNAME│ DC      │ CL8'HIGHINDX'                 000155 AND 001555.│
│ OLDCVOL │ DC      │ CL6'000155'                                  │
│ NEWCVOL │ DC      │ CL6'001555'                                  │
└─────────┴─────────┴──────────────────────────────────────────────┘
```

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

HOW TO DISCONNECT CONTROL VOLUMES

You disconnect two control volumes by using the INDEX and CAMLST macro-instructions. If a control volume is to be disconnected from the system residence volume, you need supply only the name of the high level index associated with the volume to be disconnected.

If a control volume is to be disconnected from a control volume other than the system residence volume, you must supply, in addition to the name of the high level index, the serial number of the control volume from which you want to disconnect.

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

Example: In the following example, the control volume that contains the high level index HIGHINDX is disconnected from the system residence volume.

```
┌─────────┬─────────┬──────────────────────────────────────────────┐
│ Name    │Operation│ Operand                                      │
├─────────┼─────────┼──────────────────────────────────────────────┤
│         │ INDEX   │ DISCNECT                DISCONNECT TWO CONTROL│
│         │Check Exceptional Returns          VOLUMES              │
│ DISCNECT│ CAMLST  │ DRPX,INDXNAME                                │
│ INDXNAME│ DC      │ CL8'HIGHINDX'           MUST BE 8-BYTE FIELD │
└─────────┴─────────┴──────────────────────────────────────────────┘
```

The INDEX macro-instruction points to the CAMLST macro-instruction. DRPX, the first operand of CAMLST, specifies that control volumes be disconnected. INDEXNAME, the second operand, specifies the main storage

location of an 8-byte area into which you have placed the name of the high level index of the control volume to be disconnected.

HOW TO CATALOG A DATA SET

You catalog a data set by using the CATALOG and CAMLST macro-instructions. All index levels required to catalog the data set must exist in the catalog, or an exceptional return code is set.

You must build a complete volume list in main storage. This volume list consists of volume pointers for all volumes on which the data set is stored. The first two bytes of the list indicate the number of volume pointers that follow. 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 contained in Appendix B of this chapter.)

Example: In the following example, the data set named A.B.C is cataloged under an existing index structure A.B. The data set is stored on two volumes.

```
┌─────────┬─────────┬──────────────────────────────────────────────┐
│ Name    │Operation│ Operand                                      │
├─────────┼─────────┼──────────────────────────────────────────────┤
│         │ CATALOG │ ADDABC              CATALOG DATA SET A.B.C. THE │
│         │ Check Exceptional Returns     INDEX STRUCTURE A.B. EXISTS │
│ ADDABC  │ CAMLST  │ CAT,DSNAME,,VOLUMES                          │
│ DSNAME  │ DC      │ CL6'A.B.C'          ONE BLANK FOR DELIMITER  │
│ VOLUMES │ DC      │ H'2'                TWO VOLUMES              │
│         │ DC      │ X'30002001'         2311 DISK STORAGE        │
│         │ DC      │ CL6'000014'         VOLUME SERIAL NUMBER     │
│         │ DC      │ H'0'                DATA SET SEQUENCE NUMBER │
│         │ DC      │ X'30002001'         2311 DISK STORAGE        │
│         │ 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 be cataloged. DSNAME, the second operand, specifies the main storage location of an area into which you have placed the fully qualified name of the data set to be cataloged. VOLUMES, the fourth operand, specifies the main storage location of the volume list you have built.

HOW TO REMOVE DATA SET REFERENCES FROM THE CATALOG

You remove data set references from the catalog by using the CATALOG and CAMLST macro-instructions.

Example: In the following example, references to data set A.B.C are removed from the catalog.

```
┌─────────┬─────────┬──────────────────────────────────────────────┐
│ Name    │Operation│ Operand                                      │
├─────────┼─────────┼──────────────────────────────────────────────┤
│         │ CATALOG │ REMOVE              REMOVE REFERENCES TO DATA │
│         │ Check Exceptional Returns     SET A.B.C FROM THE CATALOG │
│ REMOVE  │ CAMLST  │ UNCAT,DSNAME                                 │
│ DSNAME  │ DC      │ CL6'A.B.C'          ONE BLANK FOR DELIMITER  │
└─────────┴─────────┴──────────────────────────────────────────────┘
```

The CATALOG macro-instruction points to the CAMLST macro-instruction. UNCAT, the first operand of CAMLST, specifies that references to a data

set be removed from the catalog. DSNAME, the second operand, specifies the main storage location of an area into which you have placed the fully qualified name of the data set whose references are to be removed.

HOW TO RECATALOG A DATA SET

You recatalog a cataloged data set by using the CATALOG and CAMLST macro-instructions. Recataloging is usually performed when new volume pointers must be added to the volume list of a data set.

You must build a complete volume list in main storage. This volume list consists of volume pointers for all volumes on which the data set is stored. The first two bytes of the list indicate the number of volume pointers that follow. 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 contained in Appendix B of this chapter.)

Example: In the following example, the data set named A.B.C is recataloged. A new volume pointer is added to the volume list, which previously contained only two volume pointers.

```
┌─────────┬─────────┬──────────────────────────────────────────────┐
│ Name    │ Operation│ Operand                                     │
├─────────┼─────────┼──────────────────────────────────────────────┤
│         │ CATALOG │ RECATLG                RECATALOG DATA SET A.B.C, │
│         │ Check Exceptional Returns      ADDING A NEW VOLUME   │
│         │         │                        POINTER TO THE VOLUME │
│ RECATLG │ CAMLST  │ RECAT,DSNAME,,VOLUMES  LIST.                 │
│ DSNAME  │ DC      │ CL6'A.B.C'             ONE BLANK FOR DELIMITER │
│ VOLUMES │ DC      │ H'3'                   THREE VOLUMES         │
│         │ DC      │ X'30002001'            2311 DISK STORAGE     │
│         │ DC      │ CL6'000014'            VOLUME SERIAL NUMBER  │
│         │ DC      │ H'0'                   SEQUENCE NUMBER       │
│         │ DC      │ X'30002001'            2311 DISK STORAGE     │
│         │ DC      │ CL6'000015'            VOLUME SERIAL NUMBER  │
│         │ DC      │ H'0'                   SEQUENCE NUMBER       │
│         │ DC      │ X'30002001'            2311 DISK STORAGE     │
│         │ 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 main 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 main storage location of the volume list you have built.

HOW TO READ A DATA SET CONTROL BLOCK FROM THE VOLUME TABLE OF CONTENTS

You can read a data set control block (DSCB) into main storage by using the OBTAIN and CAMLST macro-instructions. There are two ways to specify the DSCB that you want read: by using the name of the data set associated with the DSCB, or by using the absolute track address of the DSCB.

When you specify the name of the data set, a format 1 DSCB is read into main storage. To read a DSCB other than a format 1 DSCB, you must specify an absolute track address. (DSCB formats and field descriptions are contained in the System Control Block publication).

When a data set name is specified, the 96-byte data portion of the format 1 DSCB, and the absolute track address of the DSCB are read into main storage. When the absolute track address of a DSCB is specified, the 44-byte key portion and the 96-byte data portion of the DSCB are read into main storage.

Example: In the following example, the format 1 DSCB for data set A.B.C is read into main storage. The serial number of the volume containing the DSCB is 770655.

```
┌─────────┬─────────┬──────────────────────────────────────────────┐
│ Name    │ Operation│ Operand                                     │
├─────────┼─────────┼──────────────────────────────────────────────┤
│         │ OBTAIN  │ DSCBABC                    READ DSCB FOR DATA │
│         │ Check Exceptional Returns          SET A.B.C INTO MAIN│
│ DSCBABC │ CAMLST  │ SEARCH,DSABC,VOLNUM,WORKAREA STORAGE AREA NAMED│
│ DSABC   │ DC      │ CL44'A.B.C'                WORKAREA. 96-BYTE │
│ VOLNUM  │ DC      │ CL6'770655'                DATA PORTION IS   │
│ WORKAREA│ DS      │ 0D                         READ. THE REST OF │
│         │ DS      │ 350C                       THE AREA IS USED BY│
│         │         │                            THE OBTAIN ROUTINE │
└─────────┴─────────┴──────────────────────────────────────────────┘
```

The OBTAIN macro-instruction points to the CAMLST macro-instruction. SEARCH, the first operand of CAMLST, specifies that a DSCB be read into main storage. DSABC, the second operand, specifies the main storage location of a 44-byte area into which you have placed the fully qualified name of the data set whose associated DSCB is to be read. VOLNUM, the third operand, specifies the main 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 main storage location of a 350-byte work area that is to contain the DSCB.

After execution of these macro-instructions, the first 96 bytes of the work area contain the data portion of the format 1 DSCB; the next five bytes contain the absolute track address of the DSCB. The OBTAIN routine uses the rest of the area as a work area.

HOW TO DELETE A DATA SET

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 (DSCB) for the data set to be deleted, and all space occupied by the data set to be made available for reallocation. If the data set to be deleted is sharing a split cylinder, the space will not be made available for reallocation until all data sets on the split cylinder are deleted.

A data set cannot be deleted if the expiration date in the format 1 DSCB has not passed, unless you choose to ignore the expiration date. You can ignore the expiration date 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. Certain volumes, such as the system residence volume, must always be mounted.

When deleting a data set, you must build a complete volume list in main storage. This volume list consists of volume pointers for all volumes on which the data set is stored. The first two bytes of the list indicate the number of volume pointers that follow. 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 contained in Appendix B of this chapter.)

Volumes are processed in the order that they appear in the volume list. Those volumes that are pointed to at the beginning of the list are processed first. If a volume is not mounted, a message is issued to the operator requesting him to mount the volume. You can indicate the I/O 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. When the volume is mounted, processing continues. If you do not load register 0 with a UCB address, its contents must be zero.

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 for the unavailable volume, and the next volume indicated in the volume list is processed or requested.

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

```
┌─────────┬─────────┬──────────────────────────────────────────────┐
│ Name    │ Operation│ Operand                                     │
├─────────┼─────────┼──────────────────────────────────────────────┤
│         │ SR      │ 0,0                          SET REG 0 TO ZERO│
│         │ SCRATCH │ DELABC                       DELETE DATA SET  │
│         │ Check Exceptional Returns              A.B.C. FROM TWO  │
│ DELABC  │ CAMLST  │ SCRATCH,DSABC,,VOLIST,,OVRD  VOLUMES, IGNORING│
│ DSABC   │ DC      │ CL44'A.B.C'                  THE EXPIRATION   │
│ VOLIST  │ DC      │ H'2'                         DATE IN THE DSCB.│
│         │ DC      │ X'30002001'                  2311 DISK STORAGE│
│         │ DC      │ CL6'000017'                  VOLUME SERIAL NO.│
│         │ DC      │ H'0'                         SEQUENCE NUMBER  │
│         │ DC      │ X'30002001'                  2311 DISK STORAGE│
│         │ DC      │ CL6'000018'                  VOLUME SERIAL NO.│
│         │ DC      │ H'0'                         SEQUENCE NUMBER  │
└─────────┴─────────┴──────────────────────────────────────────────┘
```

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 main 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 main storage location of the volume list you have built. OVRD, the sixth operand, specifies that the expiration date be ignored in the DSCB of the data set to be deleted.

HOW TO RENAME A DATA SET

You rename a data set stored on direct-access volumes by using the RENAME and CAMLST macro-instructions. This causes the data set name in all format 1 data set control blocks (DSCB) for the data set to be replaced by the new name that you supply.

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 required volumes must be serially mountable. Certain volumes, such as the system residence volume, must always be mounted.

When renaming a data set, you must build a complete volume list in main storage. This volume list consists of volume pointers for all

volumes on which the data set is stored. The first two bytes of the list indicate the number of volume pointers that follow. 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 contained in Appendix B of this chapter.)

Volumes are processed in the order they appear in the volume list. Those volumes that are pointed to at the beginning of the list are processed first. If a volume is not mounted, a message is issued to the operator requesting him to mount the volume. You can indicate the I/O 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. When the volume is mounted, processing continues. If you do not load register 0 with a UCB address, its contents must be zero.

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 for the unavailable volume, and the next volume indicated in the volume list is processed or requested.

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

```
┌─────────┬─────────┬──────────────────────────────────────────────┐
│ Name    │Operation│ Operand                                      │
├─────────┼─────────┼──────────────────────────────────────────────┤
│         │ SR      │ 0,0                          SET REG 0 TO ZERO│
│         │ RENAME  │ DSABC                        CHANGE DATA SET │
│         │ Check Exceptional Returns              NAME A.B.C. TO  │
│ DSABC   │ CAMLST  │ RENAME,OLDNAME,NEWNAME,VOLIST D.E.F          │
│ OLDNAME │ DC      │ CL44'A.B.C'                                  │
│ NEWNAME │ DC      │ CL44'D.E.F'                                  │
│ VOLIST  │ DC      │ H'2'                         TWO VOLUMES     │
│         │ DC      │ X'30002001'                  2311 DISK STORAGE│
│         │ DC      │ CL6'000017'                  VOLUME SERIAL NO.│
│         │ DC      │ H'0'                         SEQUENCE NUMBER │
│         │ DC      │ X'30002001'                  2311 DISK STORAGE│
│         │ DC      │ CL6'000018'                  VOLUME SERIAL NO.│
│         │ DC      │ H'0'                         SEQUENCE NUMBER │
└─────────┴─────────┴──────────────────────────────────────────────┘
```

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 main 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 main 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 main storage location of the volume list you have built.

Macro-Instructions Required to Maintain and Modify the Catalog and VTOC

Function	Macro-Instructions Required to Perform Function		
	Name	Operation	Operands
Read a block from the catalog - by name	[symbol] [list-name]	LOCATE CAMLST	list-addrx[1] NAME,dsname-relexp[6],[cvol-relexp[7]],area-relexp[9]
Read a block from the catalog - by location	[symbol] [list-name]	LOCATE CAMLST	list-addrx[1] BLOCK,ttr-relexp[3],[cvol-relexp[7]],area-relexp[9]
Build an index	[symbol] [list-name]	INDEX CAMLST	list-addrx[1] BLDX,name-relexp[2],[cvol-relexp[7]]
Build a generation index	[symbol] [list-name]	INDEX CAMLST	list-addrx[1] BLDG,name-relexp[2],[cvol-relexp[7]],,[DELETE[15]],[EMPTY[16]],number-absexp[17]
Assign an alias	[symbol] [list-name]	INDEX CAMLST	list-addrx[1] BLDA,index name-relexp[5],[cvol-relexp[7]],alias name-relexp[10]
Delete an index	[symbol] [list-name]	INDEX CAMLST	list-addrx[1] DLTX,name-relexp[2],[cvol-relexp[7]]
Delete an alias	[symbol] [list-name]	INDEX CAMLST	list-addrx[1] DLTA,index name-relexp[5],[cvol-relexp[7]]
Connect control volumes	[symbol] [list-name]	INDEX CAMLST	list-addrx[1] LNKX,index name-relexp[5],[cvol-relexp[7]],new cvol-relexp[12]
Disconnect control volumes	[symbol] [list-name]	INDEX CAMLST	list-addrx[1] DRPX,index name-relexp[5],[cvol-addrx[7]]
Catalog a data set	[symbol] [list-name]	CATALOG CAMLST	list-addrx[1] CAT,name-relexp[2],[cvol-relexp[7]],vol list-relexp[13]
Remove data set references from the catalog	[symbol] [list-name]	CATALOG CAMLST	list-addrx[1] UNCAT,name-relexp[2],[cvol-relexp[7]]
Recatalog a data set	[symbol] [list-name]	CATALOG CAMLST	list-addrx[1] RECAT,name-relexp[2],[cvol-relexp[7]],vol list-relexp[13]
Read a DSCB from the VTOC - by name	[symbol] [list-name]	OBTAIN CAMLST	list-addrx[1] SEARCH,dsname-relexp[6],vol-relexp[8],wk area-relexp[14]
Read a DSCB from the VTOC - by location	[symbol] [list-name]	OBTAIN CAMLST	list-addrx[1] SEEK,cchhr-relexp[4],vol-relexp[8],wk area-relexp[14]
Delete a data set	[symbol] [list-name]	SCRATCH CAMLST	list-addrx[1] SCRATCH,dsname-relexp[6],,vol list-relexp[13],,[OVRD[18]]
Change the data set name in a DSCB	[symbol] [list-name]	RENAME CAMLST	list-addrx[1] RENAME,dsname-relexp[6],new name-relexp[11],vol list-relexp[13]

Note: The superscript numbers refer to the enumerated list of explanations for the operands.

[1] list-addrx
points to the parameter list (labe list-name) set up by the CAMLST mac instruction.

[2] name-relexp
specifies the main storage location of fully qualified name of a data set or in level. The name cannot exceed 44 characte If the name is less than 44 characters, must be followed by a blank. The name must defined by a C-type Define Constant (instruction.

[3] ttr-relexp
specifies the main storage location of 3-byte relative track address (TTR). T address indicates the position, relative the beginning of the catalog data set, of track containing the block (TT), and the bl identification on that track (R).

[4] cchhr-relexp
specifies the main storage location of 5-byte absolute track address (CCHHR) c DSCB.

[5] index name-relexp
specifies the main storage location of name of a high level index. The area t contains the name must be eight bytes lc The name must be defined by a C-type Def Constant (DC) instruction.

[6] dsname-relexp
specifies the main storage location of a fu qualified data set name. The area t contains the name must be 44 bytes long. name must be defined by a C-type Def Constant (DC) instruction.

[7] cvol-relexp
specifies the main storage location o 6-byte volume serial number for the volum be processed. If this parameter is not sp fied, the system residence volume is p essed.

[8] vol-relexp
specifies the main storage location of 6-byte serial number of the volume on w the required DSCB is stored.

[9] area-relexp
specifies the main storage location o 265-byte work area that you must define. work area must begin on a double-word bou ry. The first 256 bytes of the work area contain the block that is read from catalog, and the last nine bytes of the area will contain the relative track add and block identification (in the form TTR the block following the one read into storage and the serial number of the volum which the block was found.

Macro-Instructions Required to Perform Function		
Name	Operation	Operands
[symbol] [list-name]	LOCATE CAMLST	list-addrx[1] NAME,dsname-relexp[6],[cvol-relexp[7]],area-relexp[9]
[symbol] [list-name]	LOCATE CAMLST	list-addrx[1] BLOCK,ttr-relexp[3],[cvol-relexp[7]],area-relexp[9]
[symbol] [list-name]	INDEX CAMLST	list-addrx[1] BLDX,name-relexp[2],[cvol-relexp[7]]
[symbol] [list-name]	INDEX CAMLST	list-addrx[1] BLDG,name-relexp[2],[cvol-relexp[7]],,[DELETE[15]],[EMPTY[16]],number-absexp[17]
[symbol] [list-name]	INDEX CAMLST	list-addrx[1] BLDA,index name-relexp[5],[cvol-relexp[7]],alias name-relexp[10]
[symbol] [list-name]	INDEX CAMLST	list-addrx[1] DLTX,name-relexp[2],[cvol-relexp[7]]
[symbol] [list-name]	INDEX CAMLST	list-addrx[1] DLTA,index name-relexp[5],[cvol-relexp[7]]
[symbol] [list-name]	INDEX CAMLST	list-addrx[1] LNKX,index name-relexp[5],[cvol-relexp[7]],new cvol-relexp[12]
[symbol] [list-name]	INDEX CAMLST	list-addrx[1] DRPX,index name-relexp[5],[cvol-addrx[7]]
[symbol] [list-name]	CATALOG CAMLST	list-addrx[1] CAT,name-relexp[2],[cvol-relexp[7]],vol list-relexp[13]
[symbol] [list-name]	CATALOG CAMLST	list-addrx[1] UNCAT,name-relexp[2],[cvol-relexp[7]]
[symbol] [list-name]	CATALOG CAMLST	list-addrx[1] RECAT,name-relexp[2],[cvol-relexp[7]],vol list-relexp[13]
[symbol] [list-name]	OBTAIN CAMLST	list-addrx[1] SEARCH,dsname-relexp[6],vol-relexp[8],wk area-relexp[14]
[symbol] [list-name]	OBTAIN CAMLST	list-addrx[1] SEEK,cchhr-relexp[4],vol-relexp[8],wk area-relexp[14]
[symbol] [list-name]	SCRATCH CAMLST	list-addrx[1] SCRATCH,dsname-relexp[6],,vol list-relexp[13],,[OVRD[18]]
[symbol] [list-name]	RENAME CAMLST	list-addrx[1] RENAME,dsname-relexp[6],new name-relexp[11],vol list-relexp[13]

cs refer to the enumerated list of explanations for the operands.

[1] list-addrx
points to the parameter list (labeled list-name) set up by the CAMLST macro-instruction.

[2] name-relexp
specifies the main 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 a blank. The name must be defined by a C-type Define Constant (DC) instruction.

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

[4] cchhr-relexp
specifies the main storage location of the 5-byte absolute track address (CCHHR) of a DSCB.

[5] index name-relexp
specifies the main storage location of the name of a high level index. The area that contains the name must be eight bytes long. The name must be defined by a C-type Define Constant (DC) instruction.

[6] dsname-relexp
specifies the main 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.

[7] cvol-relexp
specifies the main storage location of a 6-byte volume serial number for the volume to be processed. If this parameter is not specified, the system residence volume is processed.

[8] vol-relexp
specifies the main storage location of the 6-byte serial number of the volume on which the required DSCB is stored.

[9] area-relexp
specifies the main storage location of a 265-byte work area that you must define. The work area must begin on a double-word boundary. The first 256 bytes of the work area will contain the block that is read from the catalog, and the last nine bytes of the work area will contain the relative track address and block identification (in the form TTR) of the block following the one read into main storage and the serial number of the volume on which the block was found.

[10] alias name-relexp
specifies the main storage location of the name that is to be used as an alias for a high level index. The area that contains the name must be eight bytes long. The name must be defined by a C-type Define Constant (DC) instruction.

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

[12] new cvol-relexp
specifies the main storage location of the 6-byte volume serial number of the control volume that is to be connected to another control volume.

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

[14] wk area-relexp
specifies the main storage location of a 350-byte work area that you must define. The work area must begin on a double-word boundary.

If a data set name was specified, the first 96 bytes contain the data portion of a format 1 DSCB, and the next five bytes contain the absolute track address of the DSCB. The rest of the area is used as a work area by the OBTAIN routine.

If an absolute track address was specified, the first 140 bytes contain the key portion and data portion of the DSCB. The rest of the area is used as a work area by the OBTAIN routine.

[15] DELETE
specifies that all data sets dropped from a generation data group are to be deleted, i.e., the space allocated to the data sets is to be made available for reallocation.

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

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

[18] OVRD
specifies that the expiration date in the DSCB should be ignored.

EXCEPTIONAL RETURN CONDITION CODES

Control is always returned to the instruction that follows the LOCATE, INDEX, CATALOG, OBTAIN, SCRATCH, or RENAME macro-instruction. If the function has been performed successfully, register 15 contains zeros. Otherwise, register 15 contains a condition code that indicates the reason for the failure. The condition codes for the macro-instructions are as follows:

LOCATE Macro-Instruction

Code	Interpretation
4	Either the required control volume was not mounted or the specified volume does not contain a catalog data set (SYSCTLG). The volume serial number of the required volume is contained in bytes 260-265 of the work area.
8	One of the names of the qualified name was not found. Register 0 contains the number of the last valid name in the qualified name. For example, if the qualified name A.B.C.D were specified, but name C did not exist at the level specified, register 0 would contain the binary code 2. The work area contains the first index block of the last valid index name, the serial number of the volume containing the index (in bytes 260-265), and the relative track address (in bytes 257-259) of the next index block.
12	Either an index, an alias, or a control volume pointer was found when the list of qualified names was exhausted.
16	A data set resides at some level of index other than the lowest index level specified. Register 0 contains the number of simple names referred to before the data set was found. For example, if the qualified name A.B.C.D were specified, and a data set were found cataloged at A.B.C, register 0 would contain the binary code 3.
20	A syntax error exists in the name (e.g., nine characters, a double delimiter, blank name field, etc.).
24	A permanent I/O error was found when processing the catalog.

If the LOCATE macro-instruction fails to perfom its function for any of the reasons indicated above, register 0 contains the number of indexes searched before the failure was encountered.

OBTAIN Macro-Instruction

Code	Interpretation
4	The required volume was not mounted.
8	The DSCB was not found in the VTOC of the specified volume.
12	A permanent I/O error was found when processing the specified volume.

INDEX Macro-Instruction

Code	Interpretation
4	Either the required control volume was not mounted, or the specified volume does not contain a catalog data set (SYSCTLG).
8	The existing catalog structure is inconsistent with the operation performed. Because the INDEX macro-instruction uses the search routine of the LOCATE macro-instruction, register 1 contains the condition code that would be given by the LOCATE macro-instruction, and register 0 contains the number of index levels referred to during the search.
12	An attempt was made to delete an index or generation index that has an alias or has indexes or data sets cataloged under it. The index is unchanged.
16	The qualified name specified when building an index or generation index implies an index structure that does not exist; the high level index, specified when connecting control volumes, does not exist.
20	Space is not available on the specified control volume.
24	Not used with the INDEX macro-instruction.
28	A permanent I/O error was found when processing the catalog.

SCRATCH Macro-Instruction

Code	Interpretation
4	No volumes containing any part of the data set were mounted, nor was a UCB address contained in register 0.
8	An unusual condition was encountered on one or more volumes.

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

Code	Interpretation
0	The DSCB for the data set has been deleted from the VTOC on the volume pointed to.
1	The VTOC of this volume does not contain the DSCB to be deleted.
3	The DSCB was not deleted because either the OVRD option was not specified or the retention cycle has not expired.
4	A permanent I/O error was found when processing this volume.
5	A device for mounting this volume was unavailable.
6	The operator was unable to mount this volume.

CATALOG Macro-Instruction

Code	Interpretation
4	Either the required control volume was not mounted, or the specified volume does not contain a catalog data set (SYSCTLG).
8	The existing catalog structure is inconsistent with the operation performed. Because the INDEX macro-instruction uses the search routine of the LOCATE macro-instruction, register 1 contains the condition code that would be given by the LOCATE macro-instruction, and register 0 contains the number of index levels referred to during the search.
12	Not used with the CATALOG macro-instruction.
16	The index structure necessary to catalog the data set does not exist.
20	Space is not available on the specified control volume.
24	An attempt was made to catalog an improperly named generation data set.
28	A permanent I/O error was found when processing the catalog.

RENAME Macro-Instruction

Code	Interpretation
4	No volumes containing any part of the data set were mounted, nor was a UCB address contained in register 0.
8	An unusual condition was encountered on one or more volumes.

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

Code	Interpretation
0	The 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 DSCB to be renamed.
3	A DSCB containing the new name already exists in the VTOC of this volume.
4	A permanent I/O error was found when processing this volume.
5	A device for mounting this volume was unavailable.
6	The operator was unable to mount this volume.

APPENDIX A: CATALOG BLOCK ENTRIES

This section describes the contents of all catalog entries.

Control Entries

A volume index control entry is always the first entry in a volume index. The volume index control entry is 22 bytes long and contains eight fields.

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

Field 2: Last Block Address (3 bytes) -- contains the relative track address of the last block in the volume index. The address is in the form TTR.

Field 3: Half-word Count (1 byte) -- contains a binary five to indicate that five half words follow.

Field 4: Catalog Upper Limit (3 bytes) -- contains the relative track address of the last block in the catalog data set. The address is in the form TTR.

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

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

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

Field 8: Unused Bytes in Last Block (2 bytes) -- contains the binary count of the number of unused bytes in the last block of the volume index.

An index control entry is the first entry in all indexes except volume indexes. The index control entry is 18 bytes long and contains six fields.

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

Field 2: Last Block Address (3 bytes) -- contains the relative track address of the last block assigned to the index. The address is in the form TTR.

Field 3: Half-word Count (1 byte) -- contains a binary three to indicate that three half words follow.

Field 4: Index Lower Limit (3 bytes) -- contains the relative track address of the block in which this entry appears. The address is in the form TTR.

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

Field 6: Unused Bytes in Last Block (2 bytes) -- contains the binary count of the number of unused bytes remaining in the last block of the index.

An index link entry is the last entry in all index blocks. The entry is 12 bytes long and contains three fields.

Field 1: Name Field (8 bytes) -- contains only the hexadecimal number FF to ensure that this entry, because it has the highest binary name value, will appear as the last entry in any index block.

Field 2: Link Address (3 bytes) -- contains the relative track address of the next block of the same index, if there is a next block in the index. Otherwise, the field contains binary zeros.

Field 3: Half-word Count (1 byte) -- contains a binary zero to indicate that no additional fields follow.

Pointer Entries

An index pointer entry can appear in all indexes except generation indexes. The entry is 12 bytes long and contains three fields.

Field 1: Name Field (8 bytes) -- contains the name of the index being pointed to by field 2.

Field 2: Index Address (3 bytes) -- contains the relative track address of the first block of the index named in field 1. The address is in the form TTR.

Field 3: Half-word Count (1 byte) -- contains a binary zero to indicate that no additional fields follow.

A data set pointer entry can appear in any index. It contains the simple name of a data set and from one to five 12-byte fields that each identify a volume on which the named data set resides. If the data set resides on more than five volumes, a volume control block must be used to point to the volumes. The volume control block is identified by a volume control block pointer entry, not a data set pointer entry.

The data set pointer entry varies in length. The length is determined by the formula (14+12m), where m is the number of volumes containing the data set. The variable m can be from 1 through 5. The data set pointer entry can appear in any index, and it contains five fields.

Field 1: Name Field (8 bytes) -- contains the simple name of the data set whose volumes are identified in field 5.

Field 2: Address Field (3 bytes) -- contains a binary zero.

Field 3: Half-word Count (1 byte) -- contains the binary count of the number of half words that follow. The number is found by the formula (6m+1), where m is the number of volumes on which the data set resides. The variable m can be from 1 through 5.

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

Field 5: Volume Entries (12 to 60 bytes) -- contains from one to five 12-byte entries, each of which identifies a volume on which the data set resides. Each entry contains a 4-byte device code, a 6-byte volume serial number, and a 2-byte data set sequence number. The data set sequence number is zero for direct-access volumes.

A volume control block pointer entry can appear in any index. It can identify up to 20 volumes. The entry is 14 bytes long and contains four fields.

Field 1: Name Field (8 bytes) -- contains the last name of the qualified name of the data set identified by this entry. The data set resides on the volumes whose serial numbers are given in the volume control block pointed to by field 2.

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

Field 3: Half-word Count (1 byte) -- contains a binary one to indicate that one half word follows.

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

A control volume pointer entry can appear only in volume indexes. It is 18 bytes long and contains four fields.

Field 1: Name Field (8 bytes) -- contains a high level index name that appears in the volume index of the control volume identified in field 4.

Field 2: Address Field (3 bytes) -- contains binary zeros.

Field 3: Half-word Count (1 byte) -- contains a binary three to indicate that three half words follow.

Field 4: Control Volume Serial Number (6 bytes) -- contains the serial number of the control volume whose volume index contains an entry identifying the high level index name in field 1.

An alias entry can appear in volume indexes only. An alias entry is 20 bytes long and contains four fields.

Field 1: Name Field (8 bytes) -- contains the alias of the high level index identified in field 2.

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

Field 3: Half-word Count (1 byte) -- contains a binary four to indicate that four half words follow.

Field 4: True Name Field (8 bytes) -- contains the name of the index whose alias appears in field 1. The address of the index is in field 2.

A generation index pointer entry can appear in all indexes except generation indexes. The entry is 16 bytes long and contains six fields.

Field 1: Name Field (8 bytes) -- contains the name of the generation index whose address is contained in field 2.

Field 2: Address Field (3 bytes) -- contains the relative track address of the generation index named in field 1. The address is in the form TTR.

Field 3: Half-word Count (1 byte) -- contains a binary two to indicate that two half words follow.

Field 4: Flags (1 byte) -- contains flags that govern the uncataloging of data sets as specified by the DELETE and EMPTY options of the INDEX macro-instruction. The options and their hexadecimal codes are as follows:

EMPTY=01 DELETE=02 EMPTY and DELETE=03

Field 5: Maximum Generations Allowed (1 byte) -- contains the binary count of the maximum number of generations allowed in the index at one time as specified in the INDEX macro-instruction.

Field 6: Current Generation Count (2 bytes) -- contains the binary count of the number of generations cataloged in the index.

The Volume Control Block Contents

A volume control block is composed of one or more volume-list blocks. Each volume-list block contains an 8-byte key and a 256-byte data portion. The data portion of the volume-list block can identify up to 20 volumes on which a data set is recorded. The format of the volume list block is as follows:

Field 1: Number of volumes (2 bytes) -- the first volume-list block contains the binary count of volumes on which the data set is stored; the value of this field is reduced by 20 for each subsequent volume-list block. If a data set is on 61 volumes, for example, it has four volume-list blocks. The first field of each block contains 61,41,21, and 1, respectively.

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

Field 3: Zero Field (10 bytes) -- contains binary zeros.

Field 4: Chain Address (3 bytes) -- contains the relative track address of the next block of this volume control block, if additional blocks exist. The address is in the form TTR. If this is the last block of the volume control block, the field contains a binary zero. If this field is not zero, this block must contain twenty 12-byte fields identifying volumes of the data set.

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

APPENDIX B: DEVICE CODE DESIGNATIONS

Device	Features	Device Code Designation (In Hexadecimal)
IBM 2400 Series Magnetic Tape Units		30008001
IBM 2400 Series Magnetic Tape Units	7-track Compatibility	30808001
IBM 2400 Series Magnetic Tape Units	7-track Compatibility Data Conversion	30C08001
IBM 2400 Series Magnetic Tape Units	Phase Encoding	34008001
IBM 2400 Series Magnetic Tape Units	Phase Encoding with Dual Density	34208001
IBM 2311 Disk Storage Drive		30002001
IBM 2301 Drum Storage		30402002
IBM 2302 Disk Storage		30002004
IBM 2303 Drum Storage		30002003
IBM 2314 Direct Access Storage Facility		30C02008

ADDING SVC ROUTINES TO THE CONTROL PROGRAM

This chapter provides detailed information on how to write an SVC routine and insert it into the control program portion of the System/360 Operating System.

Before reading this chapter, you should be familiar with the information contained in the prerequisite publications listed below.

Documentation of the internal logic of the supervisor and its relationship to the remainder of the control program can be obtained through your IBM Branch Office.

PREREQUISITE PUBLICATIONS

The IBM System/360 Operating System: Assembler Language publication (Form C28-6514) contains the information necessary to code programs in the assembler language.

The IBM System/360 Operating System: Control Program Services publication (Form C28-6541) describes the system macro-instructions that can be used in programs coded in the assembler language.

WRITING SVC ROUTINES

Because your SVC routine will be a part of the control program, you must follow the same programming conventions used in SVC routines supplied with System/360 Operating System.

Four types of SVC routines are supplied with System/360 Operating System, and the programming conventions for each type differ. The general characteristics of the four types are described in the following text, and the programming conventions for all types are shown in tabular form.

Characteristics of SVC Routines

All SVC routines operate in the supervisor state. You should keep the following characteristics in mind when deciding what type of SVC routine to write:

- Location of the routine - Your SVC routine can be either in main storage at all times as part of the resident control program, or on a direct-access device as part of the SVC library. Type 1 and 2 SVC routines are part of the resident control program, and types 3 and 4 are in the SVC library.

- Size of the routine - Types 1, 2, and 4 SVC routines are not limited in size. However, you must divide a type 4 SVC routine into load modules of 1024 bytes or less. The size of a type 3 SVC routine must not exceed 1024 bytes.

- Design of the routine - Type 1 SVC routines must be reenterable or serially reusable; all other types must be reenterable.

- Interruption of the routine - When your SVC routine receives control, the CPU is masked for all maskable interruptions but the machine check interruption. All type 1 SVC routines must execute in this masked state. If you want to allow interruptions to occur during the execution of a type 2, 3, or 4 SVC routine, you must change the appropriate masks. If you expect that a type 2, 3, or 4 SVC routine will run for an extended period of time, it is recommended that you allow interruptions to be processed where possible.

Programming Conventions for SVC Routines

The programming conventions for the four types of SVC routines are summarized in Table 1. Details about many of the conventions are in the reference notes that follow the table. The notes are referred to by the numbers in the last column of the table. If a reference note for a convention does not pertain to all types of SVC routines, an asterisk indicates the types to which the note refers.

Table 1. Programming Conventions for SVC Routines

Conventions	Type 1	Type 2	Type 3	Type 4	Reference Code
Part of resident control program	Yes	Yes	No	No	
Size of routine	Any	Any	≤ 1024 bytes	Each load module ≤ 1024 bytes	
Reenterable routine	Optional, but must be serially reusable	Yes	Yes	Yes	1
May allow interruptions	No	Yes	Yes	Yes	2
Entry point	Must be the first byte of the routine or load module, and must be on a double-word boundary				
Number of routine	Numbers assigned to your SVC routines should be in descending order from 255 through 200				
Name of routine	IGCnnn	IGCnnn	IGC00nnn	IGCssnnn	3
Register contents at entry time	Registers 3, 4, 5, and 14 contain communication pointers; registers 0, 1, and 15 are parameter registers				4
May contain relocatable data	Yes	Yes	No*	No*	5
Can supervisor request block (SVRB) be extended	Not applicable	Yes*	Yes*	Yes*	6
May issue WAIT macro-instruction	No	Yes*	Yes*	Yes*	7
May issue XCTL macro-instruction	No	No	No	Yes*	8
May pass control to what other types of SVC routines	None	Any	Any	Any	
Type of linkage with other SVC routines	Not applicable	Issue supervisor call (SVC) instruction			
Exit from SVC Routine	Branch using return register 14				
Method of abnormal termination	Use resident abnormal termination routine	Use ABEND macro-instruction or resident abnormal termination routine			9

Reference Code	SV Routine Types	Reference Notes
1	all	If your SVC routine is to be reenterable, you cannot use macro-instructions whose expansions store information into an in-line parameter list.
2	all	You should write SVC routines so that program interruptions cannot occur. If a program interruption does occur during execution of an SVC routine, the routine loses control and the task that called the routine terminates. If a program interruption occurs and you are modifying a serially reusable SVC routine, a system queue, control blocks, etc., the modification will never complete; the next time the partially modified code is used, the results will be unpredictable.
3	all	You must use the following conventions when naming SVC routines: • Types 1 and 2 must be named IGCnnn; nnn is the decimal number of the SVC routine. You must specify this name in an ENTRY, CSECT, or START instruction. • Type 3 must be named IGC00nnn; nnn is the signed decimal number of the SVC routine. This name must be the name of a member of a partitioned data set. • Type 4 must be named IGCssnnn; nnn is the signed decimal number of the SVC routine, and ss is the number of the load module minus one, e.g., ss is 01 for the second load module of the routine. This name must be the name of a member of a partitioned data set.
4	all	Before your SVC routine receives control, the contents of all registers are saved. For type 4 routines, this applies only to the first load module of the routine. In general, the location of the register save area is unknown to the routine that is called. When your SVC routine receives control, the status of the registers is as follows: • Register 0 and 1 contain the same information as when the SVC routine was called. • Register 2 contains unpredictable information. • Register 3 contains the starting address of the communication vector table. • Register 4 contains the address of the task control block (TCB) of the task that called the SVC routine.

Reference Code	SVC Routine Types	Reference Notes
		• Register 5 contains the address of the supervisor request block (SVRB), if a type 2, 3, or 4 SVC routine is in control. If a type 1 SVC routine is in control, register 5 contains the address of the last active request block. • Register 6 through 12 contain unpredictable information. • Register 13 contains the same information as when the SVC routine was called. • Register 14 contains the return address. • Register 15 contains the same information as when the SVC routine was called. You must use registers 0, 1, and 15 if you want to pass information to the calling program. The contents of registers 2 through 14 are restored when control is returned to the calling program.
5	3,4	Because relocatable address constants are not relocated when a type 3 or 4 SVC routine is loaded into main storage, you cannot use them in coding these routines; nor can you use macro-instructions whose expansions contain relocatable address constants. Types 1 and 2 are not affected by this restriction since they are part of the resident control program.
6	2,3,4	You can extend the SVRB, in 8-byte increments, from 96 bytes up to 144 bytes. The extended area is available as a work area during execution of your routine only if you specify the extension during the system generation process. When your SVC routine receives control, register 5 contains the address of the SVRB to which the extended save area is appended.
7	2,3,4	You cannot issue the WAIT macro-instruction unless you have changed the system mask to allow I/O and external interruptions. If you have allowed these interruptions, you can issue WAIT macro-instructions that await either single or multiple events. The event control block (ECB) for single-event waits or the ECB list and ECBs for multiple-event waits must be in dynamic main storage.
8	4	When you issue an XCTL macro-instruction in a routine under control of a type 4 SVRB, the new load module is brought into a transient area. The contents of registers 2 through 13 are unchanged when control is passed to the load module; register 15 contains the entry point of the called load module.

Reference Code	SVC Routine Types	Reference Notes
9	all	Type 1 SVC routines must use the resident abnormal termination routine to terminate any task. The entry point to the abnormal termination routine is in the communication vector table (CVT). The symbolic name of the entry point is CVTBTERM. Type 2, 3, and 4 SVC routines must use the ABEND macro-instruction to terminate the current task, and must use the resident abnormal termination routine to terminate a task other than the current task. Before the resident abnormal termination routine is entered, the CPU must be masked for all maskable interruptions but the machine check interruption, and registers 0, 1, and 14 must contain the following: • Register 0 contains the address of the TCB of the task to be terminated. • Register 1 contains the following information: Bit 0 is a 1 if you want a dump taken. Bit 1 is a 1 if you want to terminate a job step. Bits 2-7 are zero. Bits 8-19 contain the error code. Bits 20-31 are zero. • Register 14 contains the return address. The resident abnormal termination routine exits by branching to the address contained in register 14. The contents of register 15 are destroyed by the abnormal termination routine.

INSERTING SVC ROUTINES INTO THE CONTROL PROGRAM

You insert SVC routines into the control program during the system generation process.

Before your SVC routine can be inserted into the control program, the routine must be a member of a cataloged partitioned data set. You must name this data set SYS1.name.

The following text gives a description of the information you must supply during the system generation process. You will find a description of the macro-instructions required during the system generation process in the publication IBM System/360 Operating System: System Generation, Form C28-6554.

Specifying SVC Routines

You use the SVCTABLE macro-instruction to specify the SVC number, the type of SVC routine, and, for type 2, 3, or 4 routines, the number of double words in the extended save area.

Inserting SVC Routines During the System Generation Process

To insert a type 1 or 2 SVC routine into the resident control program, you use the RESMODS macro-instruction. You must specify the name of the partitioned data set and the names of the members to be inserted into the control program. Each member can contain more than one SVC routine.

To insert a type 3 or 4 SVC routine into the SVC library, you use the SVCLIB macro-instruction. You must specify the name of the partitioned data set and the names of members to be included in the SVC library. The member names must conform to the conventions for naming type 3 and 4 routines, i.e., IGC00nnn and IGCssnnn.

ADDING AN ACCOUNTING ROUTINE TO THE CONTROL PROGRAM

This chapter provides detailed information on how to write an accounting routine and insert it into the control program portion of System/360 Operating System.

Before reading this section, you should be familiar with the information contained in the prerequisite publications listed below.

Documentation of the internal logic of the scheduler can be obtained through your IBM Branch Office.

PREREQUISITE PUBLICATIONS

The IBM System/360 Operating System: Assembler Language publication (Form C28-6514) contains the information necessary to code programs in the assembler language.

The IBM System/360 Operating System: Control Program Services publication (Form C28-6541) describes the system macro-instructions that can be used in programs coded in the assembler language.

WRITING AN ACCOUNTING ROUTINE

The conventions that you must follow when writing an accounting routine and the accounting information that is supplied by the operating system are described in the following text.

Entry to the Accounting Routine: Your accounting routine receives control during job and step termination. The entry point of your accounting routine must be named IEFACTRT. You can specify this name in either a CSECT statement or an ENTRY statement.

The first sequence of instructions in your routine must save the contents of registers 0 through 14. You can use the SAVE macro-instruction to perform this function; register 13 contains the address of a register save area.

Input to the Accounting Routine: Register 1 contains the starting address of a list of pointers to accounting information. Each pointer is on a full-word boundary. The items in the list and the order in which they appear are as follows:

1. 4-byte pointer to the 8-byte job name area.

2. 4-byte pointer to the 8-byte step name area. The pointer to the step name is zero when the job is terminated.

3. 4-byte pointer to the 20-byte programmer's name area.

4. 4-byte pointer to the job running time.[1] The job running time is contained in the first three bytes of a 4-byte area. The last byte contains the number of job accounting data fields.

5. 4-byte pointer to the job accounting data fields. If the JOB statement did not contain accounting information this pointer indicates a four-byte field whose last byte is zeroed. These data fields contain the accounting information that was specified in the JOB statement. The first byte of each data field contains the number of bytes of data that follow. All job accounting data fields are contiguous in main storage. The last data field is followed by a byte of zeros.

6. 4-byte pointer to the step running time.[1] The step running time is contained in the first three bytes of a 4-byte area. The last byte contains the number of step accounting data fields. The pointer to the step running time is zero when the job is terminated.

7. 4-byte pointer to the step accounting data fields. These fields contain the accounting information that was specified in the EXEC statement. The first byte of each data field contains the number of bytes of data that follow. All step accounting data fields are contiguous in main storage. The last data field is followed by a byte of zeros. If the EXEC statement did not contain an ACCT=parameter, the pointer indicates a one-byte field of zeros. The pointer to the step accounting data fields is zero when the job is terminated.

---

[1]Job running time and step running time are not supplied unless you have selected multiprogramming with a variable number of tasks and interval timing. However, if only timing is selected, you can use the timer macro-instructions in your accounting routine. An explanation of these features is contained in the publication IBM System/360 Operating System: Storage Estimates, Form C28-6551.

Output from the Accounting Routine: You can write output in two ways: by issuing console messages or by using the standard system output.

1. Console messages -- You can use Write to Operator (WTO) or Write to Operator with Reply (WTOR) macro-instructions.

2. System output -- You must assemble the following calling sequence into your routine. The contents of register 12 must be the same as when your accounting routine was entered, and register 13 must contain the address of an area of 64 full words.

```
| Name     | Operation | Operand                                          |
|----------|-----------|--------------------------------------------------|
|          | MVC       | 36(4,12),MSGADDR     MOVE MESSAGE ADDRESS AND    |
|          | MVC       | 42(2,12),MSGLEN      LENGTH TO SYSTEM TABLE.     |
|          | L         | REG15,VCONYS         BRANCH AND LINK TO MESSAGE  |
|          | BALR      | REG14,REG15          ROUTINE                     |
|          |   .       |                                                  |
|          |   .       |                                                  |
| MSGADDR  | DC        | A(MSG)                                           |
| MSG      | DC        | C'text of message'                               |
| MSGLEN   | DC        | H'two character length of message'               |
| VCONYS   | DC        | V(IEFYS)                                         |
```

Exit from the Accounting Routine: You can use the RETURN macro-instruction to restore the contents of registers and return control to the operating system.

INSERTING AN ACCOUNTING ROUTINE INTO THE CONTROL PROGRAM

You can insert your accounting routine into the control program either before or after the system generation process. In either case, you must specify that you have an accounting routine. This is done during the system generation process by using the ACCTRTN option in the SCHEDULR macro-instruction.

Inserting an Accounting Routine Before System Generation

To insert your accounting routine before system generation, you link edit it into the module library (SYS1.MODLIB) to replace the existing module named IEFACTRT. Detailed information about using the linkage editor is contained in the publication IBM System/360 Operating System: Linkage Editor, Form C28-6538.

Inserting an Accounting Routine After System Generation

To insert your accounting routine after system generation, you link edit your routine into the IEFSTERM and IEFJTERM modules of the 18K scheduler; the IEFSTERM and IEFCNTRL modules of the 44K scheduler; and the IEFCNTRL module of the 100K scheduler. You will be replacing the existing control section IEFACTRT in these modules. The scheduler modules are in the linkage library (SYS1.LINKLIB). The linkage editor control statements you must use to insert your accounting routine in the step and job termination modules of each scheduler follow.

18K Scheduler

```
     Step Termination                 Job Termination
ALIAS  GO,IEFYN                  ALIAS  IEFZA,IEFW23SD
ENTRY  IEFSD011                  ENTRY  IEFZA
NAME   IEFSTERM(R)               NAME   IEFJTERM(R)
```

44K Scheduler

```
     Step Termination                    Job Termination
ALIAS  GO,IEFYN,IEFSD009          ALIAS  IEFKA,IEFMC,IEFSD008,
ENTRY  IEFSD011                          IEFZA,IEF5DDHD
NAME   IEFSTERM(R)                ENTRY  IEFKA
                                  NAME   IEFCNTRL(R)
```

100K Scheduler

```
     Step and Job Termination
ALIAS  GO,IEFKA,IEFYN
ENTRY  IEFSD011
NAME   IEFCNTRL(R)
```

The following sequence of job control language and linkage editor statements shows insertion of your accounting routine into the 18K scheduler modules. Card input is assumed.

```
//jobname      JOB     (parameters)
//stepname     EXEC    PGM=IEWL, (parameters)
//SYSPRINT     DD      SYSOUT=A
//SYSUT1       DD      UNIT=SYSDA,SPACE=(parameters)
//SYSLMOD      DD      DSNAME=SYS1.LINKLIB,DISP=OLD
//SYSLIN       DD      *
                  .
                  .
                  .
       (accounting routine object deck)
                  .
                  .
                  .
       INCLUDE  SYSLMOD(IEFSTERM)
         ALIAS  GO,IEFYN
         ENTRY  IEFSD011
         NAME   IEFSTERM(R)
                  .
                  .
                  .
       (accounting routine object deck)
                  .
                  .
                  .
       INCLUDE  SYSLMOD(IEFJTERM)
         ALIAS  IEFZA,IEFW23SD
         ENTRY  IEFZA
         NAME   IEFJTERM(R)
```

This same statement sequence, with substitution of proper parameters, can be used with the 44K scheduler. The 100K scheduler requires only the one set of linkage editor statements and one object deck.

NONSTANDARD LABEL PROCESSING ROUTINES; VOLUME LABEL AND DUAL-DENSITY TAPE DEVICE EDITOR ROUTINES

Section 1 of this chapter explains how to write routines to process nonstandard labels on magnetic tape, and how to add them to the control program.

Section 2 of this chapter explains how to write tape volume label and dual-density tape device editor routines and insert them in the control program in place of IBM supplied editor routines.

Before reading this chapter, you should be familiar with the information contained in the prerequisite publications listed below. Documentation of the internal logic of the standard label processing routines can be obtained through your IBM Branch Office.

Prerequisite Publications

The IBM System/360 Operating System: Assembler Language publication (Form C28-6514) contains the information necessary to code programs in the assembler language.

The IBM System/360 Operating System: Control Program Services publication (Form C28-6541) describes the macro-instructions that are used to request services from the control program.

The IBM System/360 Operating System: System Control Block publication (Form C28-6628) contains format illustrations and field descriptions of the system control blocks referred to in this chapter. This publication also describes the format and fields of standard magnetic tape labels.

The IBM System/360 Operating System: Data Management publication (Form C28-6537) discusses magnetic tape labeling.

SECTION 1: WRITING NONSTANDARD LABEL PROCESSING ROUTINES

For magnetic tape volumes with nonstandard labels (i.e., input or output header or trailer labels that do not conform to the standard label format), you must write your own label processing routines. You must also perform such functions as volume identification and volume positioning.

The appropriate nonstandard label processing routine is selected, brought into main storage, and executed when the data control block is either opened or closed, or when an end-of-volume or end-of-data set condition occurs. When a nonstandard label processing routine has completed, it must return control to the OPEN, CLOSE, or EOV routine, which then continues its normal processing. The following paragraphs explain this flow of control between the control program and each type of nonstandard label processing routine. Information on tape positioning and volume identification is also provided.

Input Header Label Routines

An input header label routine is brought into main storage when either the OPEN or EOV routine is executed. Your routine is entered after the control program has determined that a tape does not have standard labels. When your routine receives control, the tape will have been positioned at the interrecord gap preceding the nonstandard label.

If your routine determines that the wrong volume is mounted, you must place a 1 in the high-order bit position of the SRTEDMCT field of the unit control block (UCB), and return control to the control program. The control program then issues a message to the operator requesting that the correct volume be mounted. When the new volume is mounted, your routine is entered again after the control program has checked the initial label.

Before returning control to the control program, you should, for forward read operations, position the tape at the interrecord gap that precedes the initial record of the data set; for backward read operations, the tape should be positioned after the last record of the data set.

Input Trailer Label Routine

An input trailer label routine is brought into main storage when the EOV routine is executed. Your routine is entered when a tape mark is encountered. When your routine receives control, the tape will have been positioned, for forward read operations, immediately after the tape mark at the end of the data set; for backward read operations, immediately before the tape mark at the beginning of the data set.

You need not reposition the tape before returning control to the control program.

Output Header Label Routines

An output header label routine is brought into main storage when either the OPEN or EOV routine is executed. Your routine is entered after the control program has determined that a tape does not have standard labels. When your routine receives control, the tape will have been positioned at the interrecord gap preceding the nonstandard label.

If your routine determines that the wrong volume is mounted, you must place a 1 in the high-order bit position of the SRTEDMCT field of the unit control block (UCB), and return control to the control program. The control program issues a message to the operator requesting that the correct volume be mounted. When the new volume is mounted, your routine

is entered again after the control program has checked the initial label and positioned the tape.

You need not reposition the tape before returning control to the control program.

Output Trailer Label Routines

An output trailer label routine is brought into main storage when either the EOV or CLOSE routine is executed. When your routine receives control, the tape will have been positioned at the interrecord gap following the last data set record that was written. After you have written the tape mark and label, you need not reposition the tape before returning control to the control program.

The output trailer label routine is also brought into main storage when input data sets are closed. This allows you to position the tapes if necessary.

Programming Conventions

This section describes the conventions to be followed when writing your routines.

- Size of the routine - Nonstandard label processing routines are not limited in size. However, if the size of such a routine exceeds 1024 bytes, you must divide the routine into load modules, each of which is 1024 bytes or less. To pass control between load modules, you must use the XCTL macro-instruction.

- Design of the routine - Nonstandard label processing routines must be reenterable. Relocatable address constants cannot be used in coding these routines or in coding any channel command words (CCW) to be used in the routine; nor can macro-instructions be used whose expansions contain relocatable address constants.

- Register usage - The contents of registers 2 through 14 must be saved when your routine receives control from the control program. The contents of these registers must be restored before your routine returns control.

- Entry Point of the routine - The entry point of the routine must be the first byte of the load module, and must be on a double-word boundary.

- Exit from the routine - The XCTL macro-instruction (E-form) must be used to exit from your routine and return control to a specific control program module. These modules differ depending both upon the control program routine from which control was received, and the type of label processing being performed.

  Module names are shown below for each control program routine and for each type of label processing routine.

Label Processing Routine	Control Program Routine	Module Name
Input Header	OPEN	IGG0190B
	EOV	IGG0550D
Input Trailer	EOV	IGG0550B
Output Header	OPEN	IGG0190R
	EOV	IGG0550H
Output Trailer	EOV	IGG0550F
	CLOSE	IGG0200B

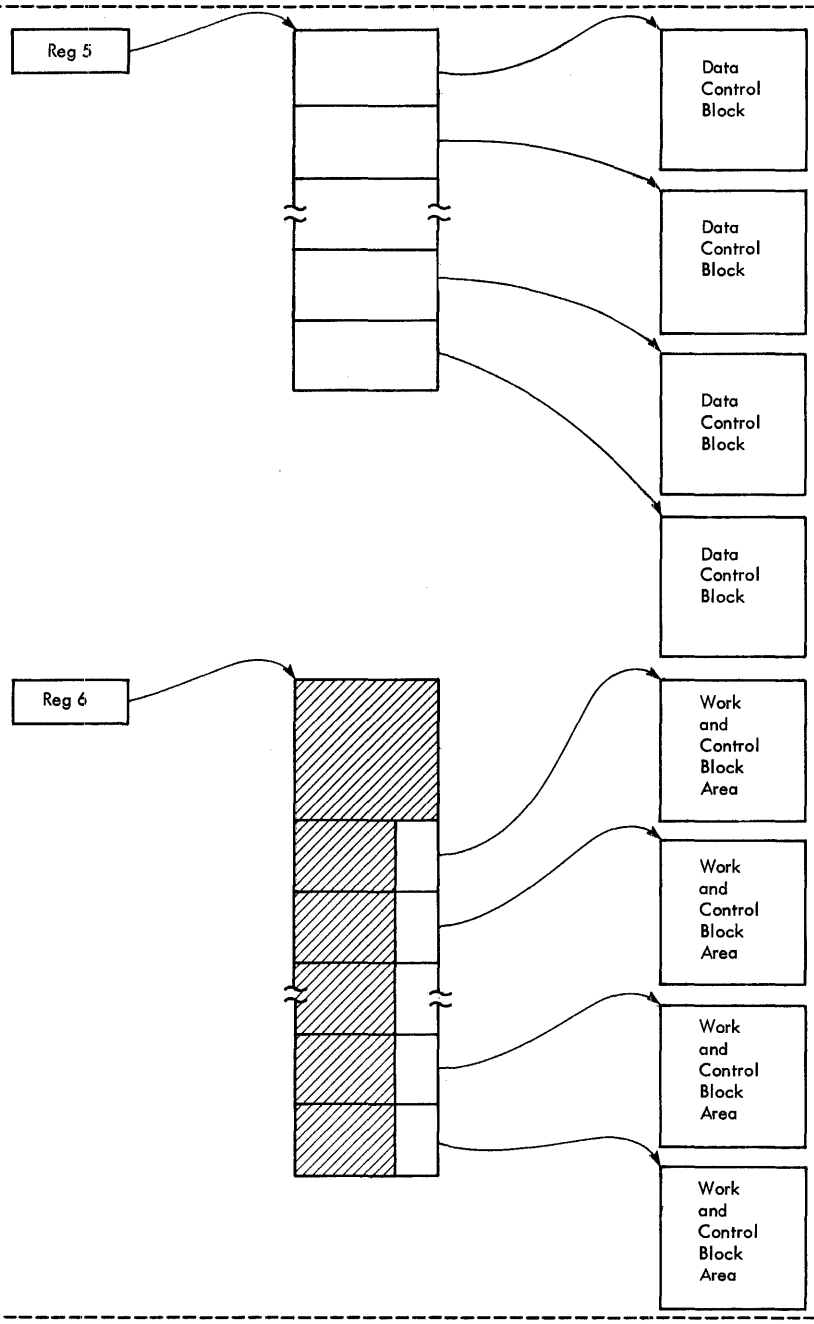
- Work areas - The GETMAIN macro-instruction must be used to obtain main storage for all of your work areas, including those areas used to read in or create a label. The FREEMAIN macro-instruction must be used to release this main storage.

Program Functions

In processing nonstandard labels, you must perform many of the functions that the control program performs in processing standard labels. All I/O operations, as, for example, those required to read labels, to write labels, and to position volumes, must be performed by using the execute channel program (EXCP) macro-instruction. The use of EXCP normally requires that you build several control blocks in your work area. However, you can save much coding effort by using existing control blocks built by the control program.

When your routine receives control from the OPEN or CLOSE routine, the status of control information and pointers is as shown in Figure 1.

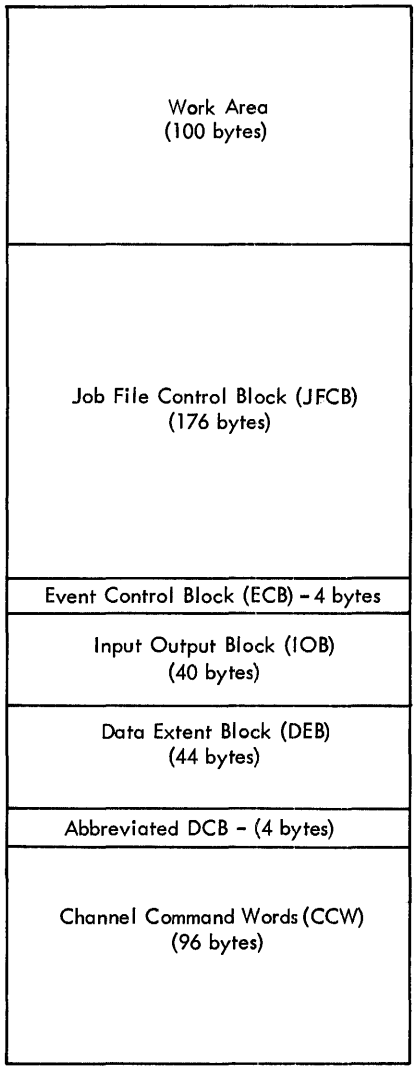
When your routine receives control from the EOV routine, register 2 contains the address of a DCB, and register 4 contains the address of a combined work and control block area. The format of this area is shown in Figure 2.



Register 5 contains the starting address of a list of DCB addresses. Each DCB specified in the OPEN or CLOSE macro-instruction has a 4-byte entry in the list. The DCBs to which the entries point are in the problem program.

For each DCB specified in the OPEN or CLOSE macro-instruction, a combined work and control block area is built. Register 6 contains the starting address of a table that contains an address for each work and control block area. The addresses of the areas are contained in the low-order three bytes of 8-byte entries. The list of 8-byte entries begins 32 bytes from the starting address of the table. The format of the combined work and control block area is shown in Figure 2.

Figure 1. Status of Control Information and Pointers



Each of the fields within the work and control block area can be addressed by your nonstandard label processing routines. The IECDSECT macro-instruction defines the symbolic names of all these fields. (The macro-definition and how to add it to the macro-library are in the Appendix of this chapter.) Code this macro-instruction (with a null operand field and immediately preceded by a DSECT statement) in the list of constants for each of your nonstandard label processing routines. Using the starting address of the work area as a base, you are able to address any field symbolically.

Figure 2. Format of Combined Work and Control Block Area

General Logic Flow of a Nonstandard Label Processing Routine: The general flowcharts in Figures 3, 4, and 5 show the logic that you could use in your routines. Each block in the flowcharts is numbered, and the number corresponds to an item in the list of explanations of the blocks.

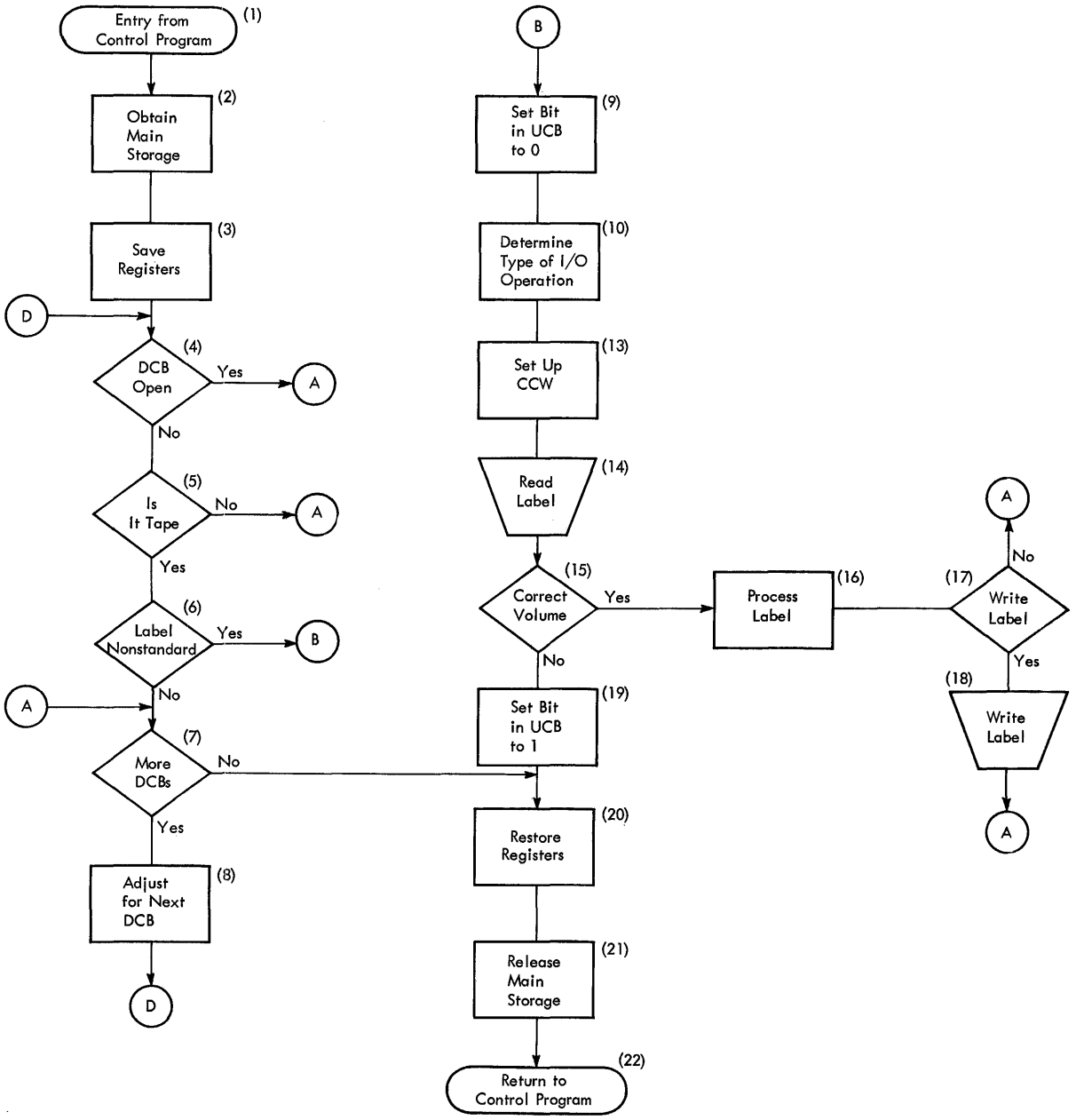


Figure 3. General Flow of a Nonstandard Label Processing Routine After Receiving Control From the OPEN Routine

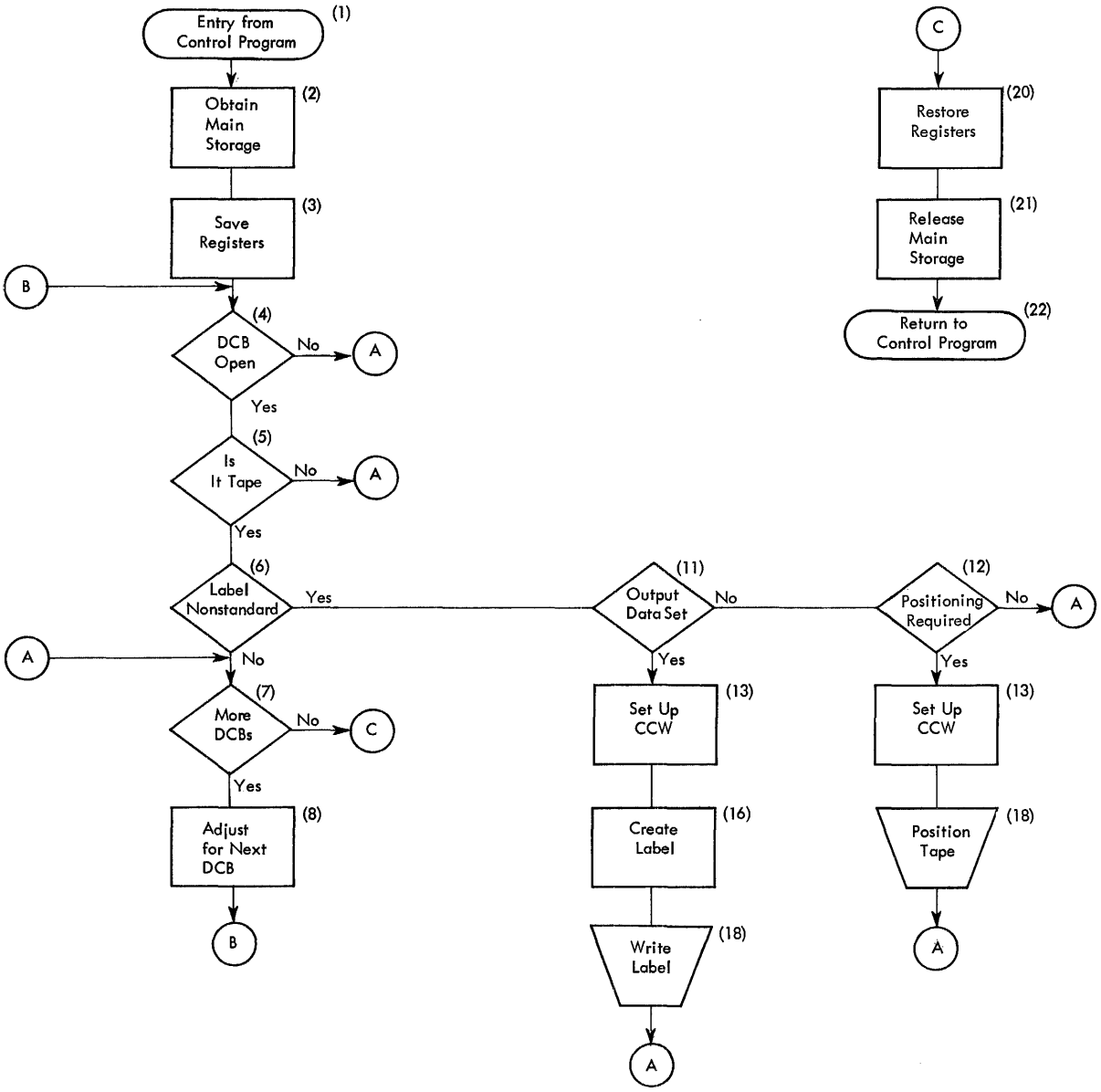


Figure 4. General Flow of a Nonstandard Label Processing Routine After Receiving Control From the CLOSE Routine

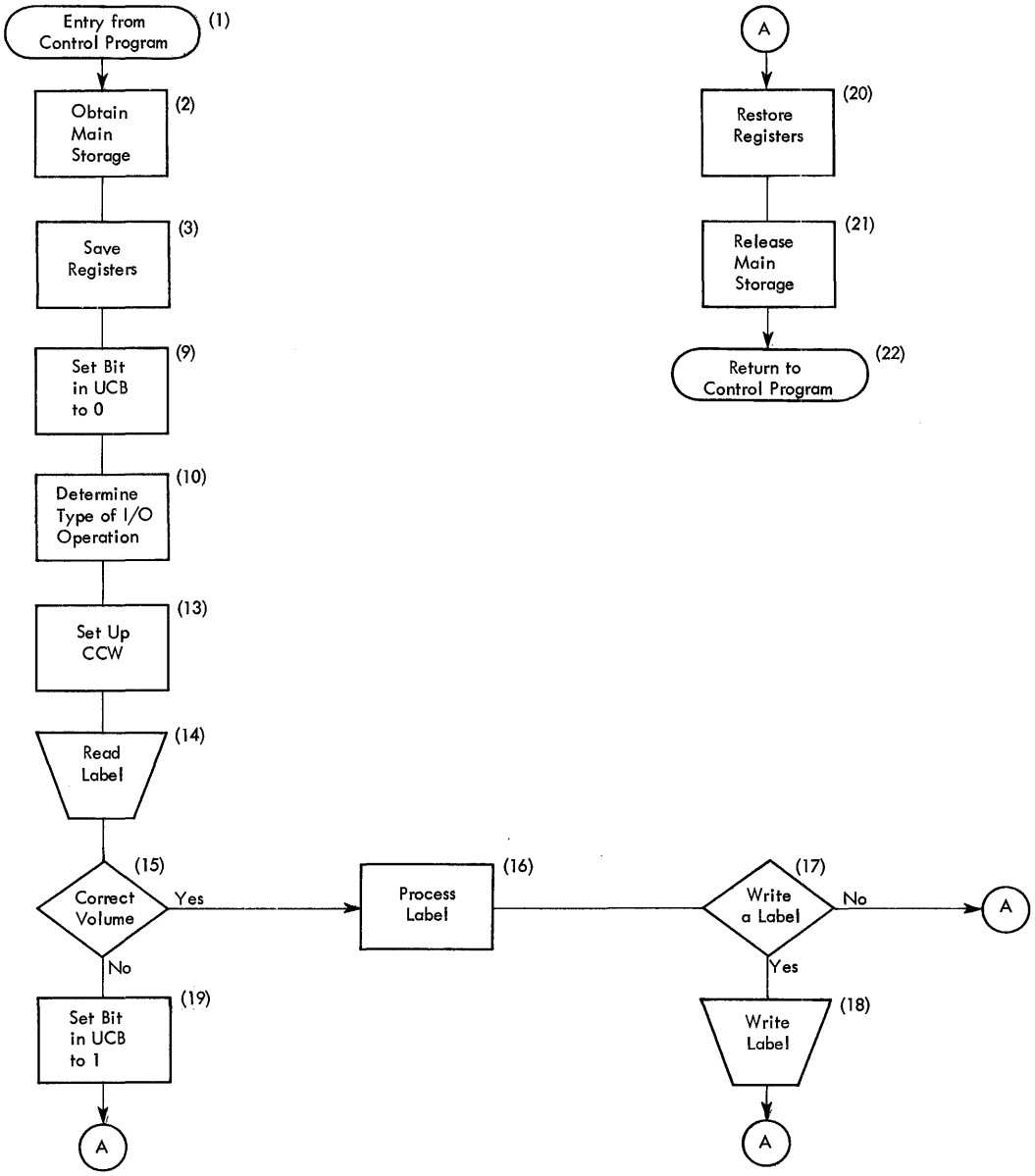


Figure 5. General Flow of a Nonstandard Label Processing Routine After Receiving Control From the EOV Routine

Explanations of Logic Blocks

1. The entry is in the form of an XCTL macro-instruction issued by the control program.

2. Use the GETMAIN macro-instruction to obtain main storage.

3. Use the store multiple instruction (STM).

4. To locate the address of the data control block (DCB), use the contents of register 5. To determine if the DCB is open, test bit 3 of the DCBOFLGS field of the DCB; if this bit is zero, the DCB has not been opened. (The symbolic names of all fields in the DCB are defined by the DCBD macro-instruction.)

5. To determine if a tape data set is being processed, test the UCB3TAPE field of the unit control block (UCB); this bit is one for a tape data set. The symbolic names of all fields in the UCB are defined by the IEFUCBOB macro-instruction. (The macro-definition and how to add it to the macro-library are in the Appendix of this chapter.) The address of the UCB is contained in the DXDEBUCB field of the data extent block (DEB) as defined by the IECDSECT macro-instruction. (The macro-definition and how to add it to the macro-library are in the Appendix of this chapter.)

6. To determine if nonstandard labels have been specified, test the JFCBLTYP field of the job file control block (JFCB); this field contains a hexadecimal 04 when nonstandard labels have been specified.

7. The final DCB entry in the list of DCB addresses contains a one in its high-order bit position.

8. Add 4 to the contents of register 5; add 8 to the contents of register 6.

9. Set the high-order bit to zero in the SRTEDMCT field of the UCB.

10. To determine the type of I/O operation specified in the OPEN macro-instruction, check the bit configuration of the high-order byte of the DCB entry in the list of DCB addresses. The bit configuration for each type of I/O operation is shown below. (The high-order four bits correspond to the disposition of the data set; the low-order four bits correspond to the I/O operation itself. For example, the bit configuration x0110000 indicates a data set opened for input whose disposition is LEAVE.)

Bits	0	1	2	3	4	5	6	7	
	x	0	0	1	x	x	x	x	REREAD
	x	0	1	1	x	x	x	x	LEAVE
	x	0	0	0	x	x	x	x	Neither REREAD nor LEAVE
	x	x	x	x	0	0	0	0	INPUT
	x	x	x	x	1	1	1	1	OUTPUT
	x	x	x	x	0	0	1	1	INOUT
	x	x	x	x	0	1	1	1	OUTIN
	x	x	x	x	0	0	0	1	RDBACK
	x	x	x	x	0	1	0	0	UPDAT

11. To determine the mode of the data set, test the high-order bit of the DCBOFLGS field of the DCB. If this bit is one, the data-set mode is output; if this bit is zero, the data-set mode is input. (The symbolic names of all fields in the DCB are defined by the DCBD macro-instruction.)

12. You may want to position the tape if you have closed an input data set before all data has been read.

13. Move your CCW into the channel program area of the control program's work area. (The symbolic name of the first entry in the channel program area is DXCCW.)

14. Issue an EXCP macro-instruction specifying the address of the control program's IOB. (The symbolic name of the IOB is DXIOB.)

15. Techniques used to check for correct volume will differ depending on the label formats used in the installation.

16. Label processing routines will differ by label format.

17. If a write operation is required, this block can be used.

18. Issue an EXCP macro-instruction specifying the address of the control program's IOB. (The symbolic name of the IOB is DXIOB.)

19. Set the high-order bit to 1 in the SRTEDMCT field of the UCB.

20. Use the load multiple instruction (LM).

21. Use the FREEMAIN macro-instruction to free the work area obtained in step 2.

22. Use the XCTL macro-instruction, specifying the appropriate operand.

The following coding sequence illustrates an exit from your routine during OPEN or CLOSE operations.

```
      [1] MVC   0(8,6),MODNAME
      [2] LA    15,DXCCW12
          XCTL  EPLOC=(6),DCB=0,SF=(E,(15))
MODNAME DC      C'IGGxxxxx'
```

[1] The name of the module you are transferring tc is moved to the first 8 bytes of the table pointed to by register 6 (see Figure 1).

[2] DXCCW12 addresses an eight-byte field that may be used for the remote supervisor parameter list required by the E-form of the XCTL macro-instruction.

The following coding sequence illustrates an exit from your routine during end-of-volume operations.

```
        MVC   DXCCW12+16(8),MODNAME
         LA   15,DXCCW12+8
         LA   0,DXCCW12+16
       XCTL   EPLOC=(0),DCB=0,SF=(E,(15))
MODNAME DC    C'IGGxxxxx'
```

The IECDSECT macro-instruction, discussed in the appendix to this chapter, defines the name DXCCW12. The DCB= parameter of the XCTL macro-instruction must be coded DCB=0 in this application.

INSERTING NONSTANDARD LABEL ROUTINES INTO THE CONTROL PROGRAM

Nonstandard label processing routines must be included in the control program as part of the SVC library (SYS1.SVCLIB), since they are load modules of type 4 SVC routines. You insert nonstandard label processing routines into the control program during the system generation process.

Before your nonstandard label routine can be inserted into the control program, each load module of the routine must be a member of a cataloged partitioned data set. You must name this data set SYS1.name.

To insert your nonstandard label routine load modules into the SVC library, you use the SVCLIB macro-instruction. Using this macro-instruction, you must specify the name of the partitioned data set and the names of members to be included in the SVC library. Member names for the first load module of each type of label processing routine are shown below. (Member names for additional load modules must begin with the letters NSL or IGC.) The format of the SVCLIB macro-instruction is contained in the publication IBM System/360 Operating System: System Generation, Form C28-6554.

Nonstandard Label Processing Routine	Control Program Routine	Member Name
Input header	OPEN	NSLOHDRI
	EOV	NSLEHDRI
Output header	OPEN	NSLOHDRO
	EOV	NSLEHDRO
Input trailer	EOV	NSLETRLI
Output trailer	EOV	NSLETRLO
	CLOSE	NSLCTRLO

SECTION 2: VOLUME LABEL AND DUAL-DENSITY TAPE DEVICE EDITOR ROUTINES

When writing data sets on magnetic tape (OUTPUT or OUTIN specified in the OPEN macro-instruction) conflicts may be detected between:

1. The label type specified by the program and the label type on the currently mounted volume.

2. The recording density specified by the program, and the density at which a dual-density tape device is set to record.

These volume label and density conflicts are detected by the OPEN or EOV control program routines during label verification procedures. IBM supplies editor routines that function when these conflicts are detected, resolving the conflicts by requesting dismounting of the currently mounted tape volume and the mounting of a new tape volume whose label and/or density conform to the program specifications. The editor routines you write replace the IBM supplied routines, and can be designed to resolve the label and density conflicts without operator intervention (i.e., tape handling).

Your editor routines can resolve label and density conflicts by writing labels, "cancelling labels," and by performing write operations to set the desired density on a dual-density tape device. Or, your editor routines can reset system control blocks (in effect, change the program specifications) to agree with the label type and/or density of the currently mounted volume. Or, your installation may desire a combination of these actions, including dismounting under certain conditions. All these possible actions may be included in your editor routines.

You may replace the standard IBM editor routine associated with the OPEN routine; the standard IBM editor routine associated with the EOV routine, or both.

The balance of this section provides you with the information necessary to write editor routines; programming conventions, entry conditions, a suggested logic for your OPEN and/or EOV editor routines, and how to insert your routines in the control program.

PROGRAMMING CONVENTIONS

Your editor routines must conform to the same general programming conventions as the nonstandard label processing routines discussed in Section I of this chapter (see Section I - Programming Conventions) in so far as size, design, register usage, entry points, and work areas are concerned.

You must name the first (or only) module of your routines as follows:

OMODVOL1 -- the editor routine associated with OPEN
EMODVOL1 -- the editor routine associated with EOV

If your editor routines consist of more than one load module, names for the additional modules must begin with the prefix OMODVOL for the OPEN routine, EMODVOL for the EOV routine. Transfer between the modules must be by name.

As discussed in Section I of this chapter, you must use EXCP programming to perform the needed input/output operations.

ENTRY CONDITIONS AND GENERAL LOGIC FLOW OF THE EDITOR ROUTINES

Figure 6 -- Editor Routine Entry Conditions -- and the logic flowcharts (Figures 7 and 8) with their accompanying text present the basic information necessary to design and write your editor routines. The logic charts depict a suggested logic -- oriented towards resolving label and density conflicts by altering the characteristics of the mounted volume.

ENTRY CONDITIONS

Figure 6 presents the four conditions under which the control program OPEN or EOV routines transfer control to your editor routines, and the general action the editor routine takes to permit processing on the current volume to continue. The first two conditions arise only when the tape volume is mounted on a dual-density tape device.

Entry from the EOV routines occurs when a volume switching operation is necessary and any condition noted in Figure 6 is present.

Program Specification	Mounted Volume Characteristics	Transfer Occurs on a	Possible Editor Routine Action
SL - 800 or 1600 BPI density	SL -- CD	DC[1]	Overwrite the standard label with a standard label. The first write from loadpoint sets the recording density on a dual-density device. (See Figure 7 or 8 -- blocks 15b, 16 and explanation).
NSL[2] - 800 or 1600BPI density	NSL OR NL -- CD	DC[1]	Write a tapemark to set density. The program specification NSL will cause control to be given to your nonstandard label routines after return to OPEN or EOV. (See Figure 7 - blocks 15, 15b, 16. If your installation supports protection and retention data checking NSL volumes, see block 6).
SL	NSL or NL[3]	LC	Write a standard volume label. (See Figure 7 - blocks 15, 15a, 16. If your installation supports protection and retention date checking on NSL volumes, see block 6).
NL or NSL	SL[3]	LC	Overwrite standard label with a tapemark, i.e. "cancel." (See Figure 7 - blocks 15, 15a, 16). Depending on whether NL or NSL is specified by the program, OPEN or EOV will either position tape (NL) or transfer control to your nonstandard label routines (NSL) when control is returned to them.

Legend:
- SL - standard volume label
- NSL - nonstandard volume label
- NL - no volume label
- DC - density check
- LC - label check
- CD - Conflicting density. The volume has been previously written on, or in a, recording density other than that specified by the program. The recording density on a dual-density device is set by sensing the density of the mounted volume.

[1]Dual-Density devices only.
[2]If NL is specified, no density check is performed. For NL volumes, tape is positioned at load point and recording density is set by the first write command.
[3]If the volume is mounted on a dual-density device a density condition may also exist. If will be corrected by the write operation.

Note: The OPEN and EOV routines position the tape at load point before transferring control to the editor routines.

Figure 6. Editor Routine Entry Conditions

GENERAL LOGIC FLOW

Figures 7 and 8 depict a suggested logic you can use to develop your editor routines. Note that Figure 8 (the EOV editor routine) does not contain logic blocks corresponding to blocks 5, 18, and 19 in Figure 7 (the OPEN editor routine). These blocks represent functions that you must program when receiving control from the OPEN routine. You must test all the DCBs defined by the OPEN macro-instruction before returning control to the OPEN routine. When receiving control from the EOV routine, you only have to process one DCB.

Note: If your installation does not support protection and retention data checking on nonstandard label volumes and does not desire to maintain retention date checking on standard label volumes, you need not implement the functions of logic blocks 6 through 13 in Figures 7 and 8.

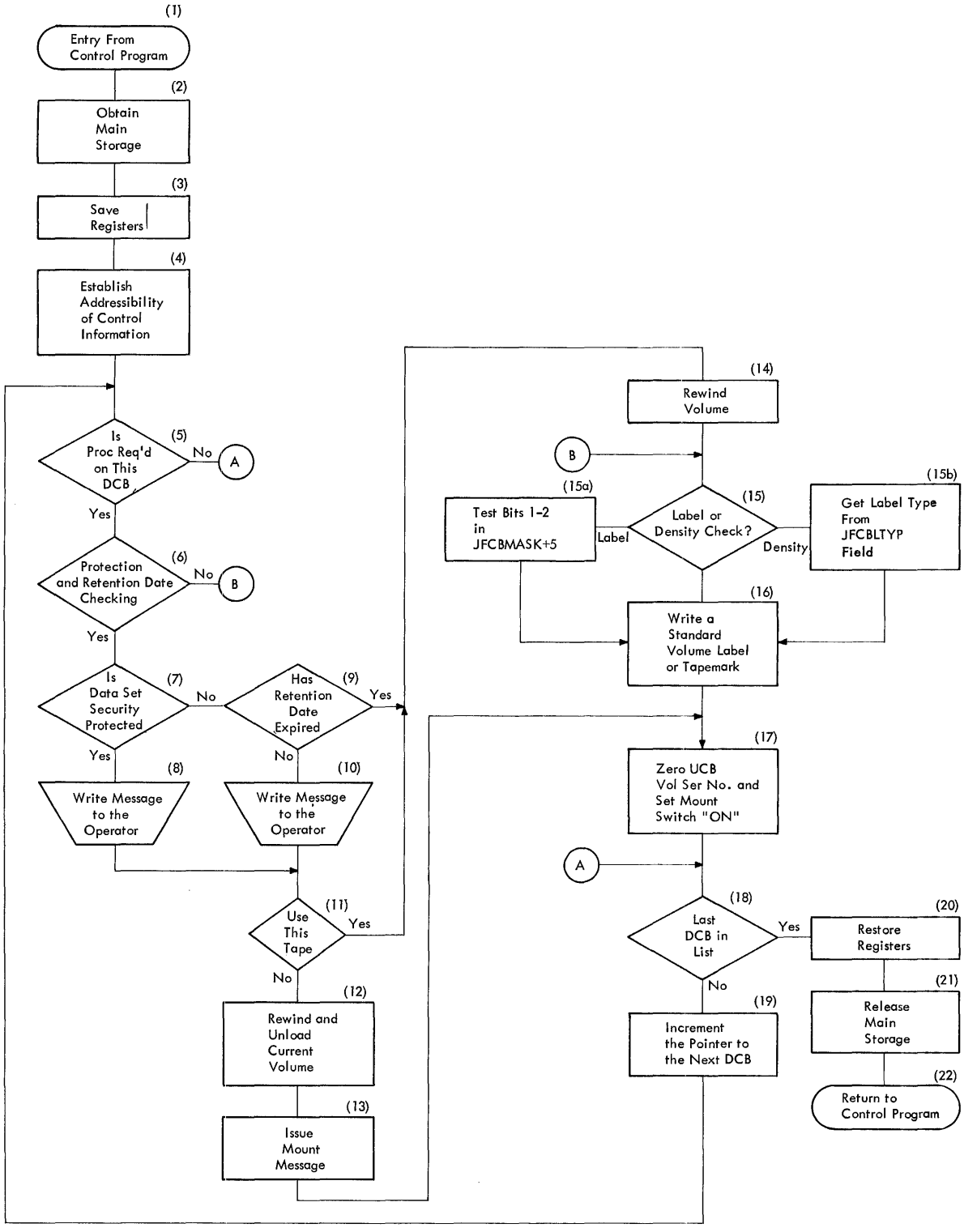


Figure 7. General Flow of an Editor Routine After Receiving Control From the OPEN Routine

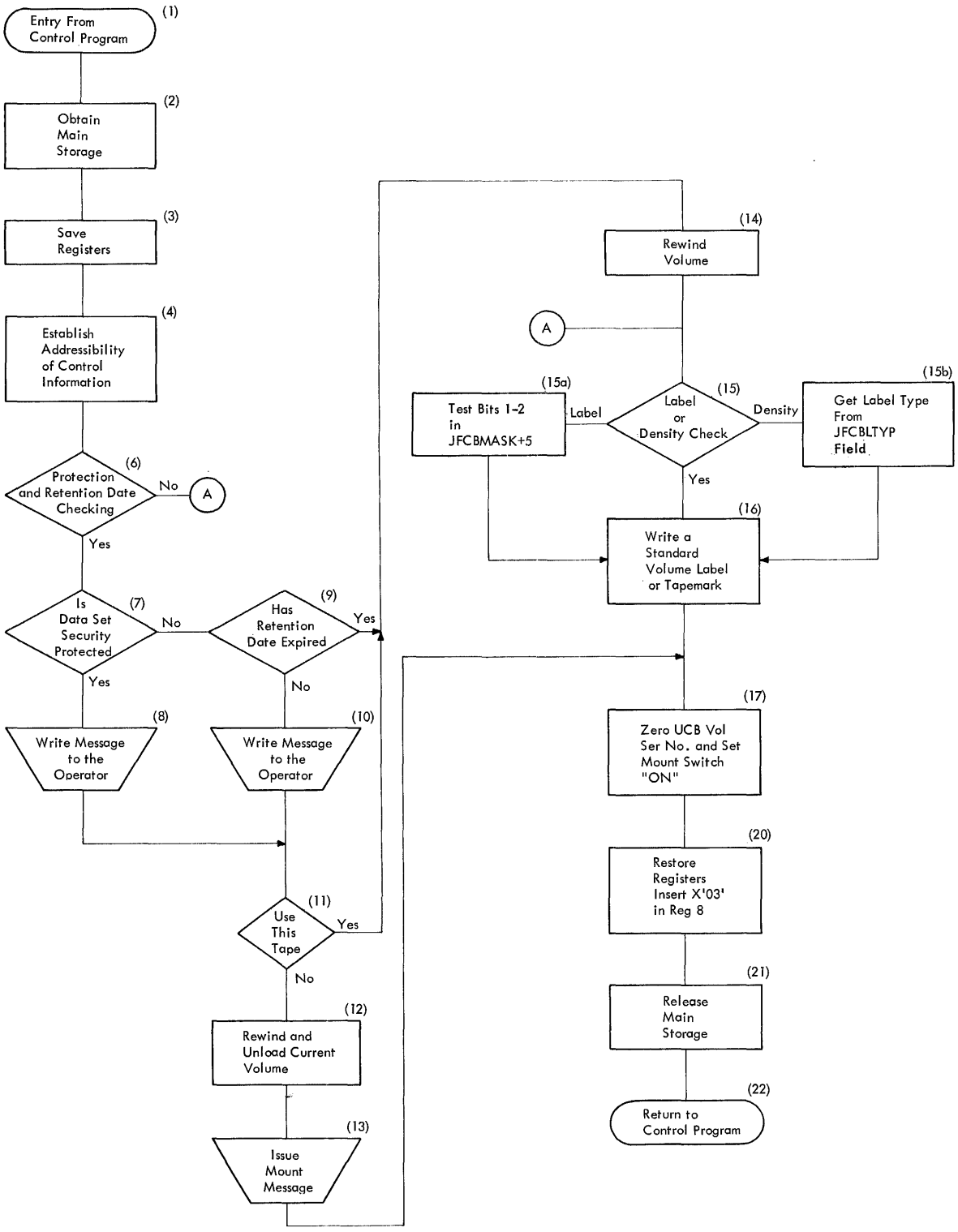


Figure 8. General Flow of an Editor Routine After Receiving Control From the End-of-Volume Routine

LOGIC BLOCK EXPLANATIONS

1. Your exception routine receives control via an XCTL macro-instruction issued by the OPEN or EOV routines of the control program.

2. Use the GETMAIN macro-instruction. The main storage you obtain must contain all your work areas, including those used to read in a label or write a label.

3. Use the store multiple instruction (STM).

4. Figure 1 in Section I of this chapter provides the information you need to establish addressability of the DCB address list and work and control block area for each DCB defined by the OPEN macro-instruction.

   When you receive control from the EOV routine, general register 2 contains the address of the DCB for the data set and general register 4 contains the address of the work and control block area associated with the DCB.

   The IECDSECT macro-instruction shown in the Appendix of this chapter symbolically defines the fields of the work and control block area (see Figure 2 in Section I).

   You will also need to address the unit control block (UCB) for the device on which the tape volume is mounted. The address of the UCB may be obtained from the DXDEBUCB field of the data extent block defined by the IECDSECT macro-instruction. The IEFUCBOB macro-instruction (see Appendix) defines the fields of the unit control block.

5. Bit configurations in the byte addressed by JFCBMASK+5 indicate whether label checks or density checks have occurred, and, in the case of a label check, the condition that caused the check. At this point, you test bits 0 and 3. If either bit is set to 1, processing is required.

   The field JFCBMASK is defined by the IECDSECT macro-instruction. Bit settings in the byte at JFCBMASK+5 are defined as:

Bits	Setting	Meaning
0	1	Label check has occurred.
1	1	Standard label (SL) specified; no label/nonstandard label on mounted volume.
2	1	No label (NL) or nonstandard label (NSL) specified; standard label on mounted volume.
3	1	Density check has occurred.
4-7	-	Reserved for future use.

6. If your installation supports a protection and retention date scheme involving nonstandard labels, and/or you want to maintain retention date and protection checking on standard labels, you must incorporate code in your editor routines to check for protection and retention date expiration.

   If checking is desired, you must, at this point, read the first record and determine the label type.

   To perform the I/O operation, move your CCWs into the channel program field of the work and control block area. The symbolic name for the first entry in this field is DXCCW. Then issue an EXCP macro-instruction specifying the address of the control

programs input/output block (IOB). The symoolic name for the IOB is DXIOB. These fields (DXCCW, DXIOB) are defined by the IECDSECT macro-instruction. Note: There are twelve CCW locations in the DXCCW field. Do not place a CCW at the location defined by DXCCW7 in your editor routine for OPEN. Do not place a CCW in locations DXCCW11 or DXCCW12 in your editor routine for EOV.

7. To check the retention date and/or protection fields in a standard label you must read the data set header 1 record into a workarea. The format of the nonstandard label defined by your installation determines how you access the protection and retention date fields in the nonstandard label. Step 6 provides directions for handling the I/O operation.

8. Write a message to the operator to inform him that the volume is protected and to determine if it is to be used.

9. See step 7 above.

10. Write a message to the operator to inform him that the expiration date for the mounted volume has not elapsed and to determine if it is to be used.

11. If the volume is to be used, continue processing to resolve label or density conditions.

12. Rewind and unload the currently mounted volume. Step 6 provides directions for handling the I/O operation.

13. Write a message to the operator requesting dismounting of the current volume and mounting of a new volume. The device name (in EBCDIC) may be obtained from the UCBNAME field of the unit control block.

14. Step 6 provides directions for handling the I/O operation.

15. Test bit 3 of the byte at JFCBMASK+5. If set to one, control was received as a result of a density check.

    Test bit 0 of the byte at JFCBMASK+5. If set to one, control was received as the result of a label check.

15a. If control was received as the result of a label check, test bits 1 and 2 of the byte at JFCBMASK+5. See step 5.

15b. If control is received as the result of a density check, use the JFCBLTYP field in the job file control block (JFCB) to ascertain the type of label specified in the program. A hexadecimal 04 indicates a nonstandard label (NSL) has been specified, a hexadecimal 02 indicates that a standard label has been specified.

16. When correcting a density check or label check condition, and a nonstandard label (NSL) or no label (NL) is specified by the program, you must write some kind of record on the tape that will be interpreted by the OPEN or EOV routines as a nonstandard label or no label, i.e., it does not contain VOL1 in the first four bytes of the record. The easiest way to do this is to write a tapemark. Upon return to OPEN or EOV and re-verification of the label, the specification for label type and density will have been met, the OPEN or EOV will transfer control to your nonstandard label routines if NSL is specified or position the tape for writing if NL has been specified.

    The System Control Block publication contains the format and field descriptions for standard tape volume labels. You must supply

information for the label identifier (VOLIABI), the label sequence number (VOLNO), and the volume serial number (VOLSERNO) fields, and record the balance of the label as blanks.

You enter VOL in the label identifier field, a 1 in the label sequence number field, and a six character serial number in the volume serial number field. Note: To ensure that two or more tape volumes carrying the same serial number are not produced, write to the operator at this point for assignment of a serial number.

Data set header labels 1 and 2 are constructed by the OPEN or EOV routines after control is returned to them.

Note: If you desire, at this point, you may change the control block settings to conform to the characteristics of the tape volume mounted, i.e., reset the label type field in the JFCB to conform with the type of label on the volume mounted, and change the density field in the DCB to the density of the tape mounted.

17. The symbolic name for the volume serial number field in the unit control block is SRTEVOLI. The "mount switch" is the high order bit of the field named SRTEDMCT in the unit control block. These fields are defined by the IEFUCBOB macro-instruction. Exclusive OR(XC) the SRTEVOLI field with itself. OR(OI) the SRTEDMCT field with X'80'.

18. When receiving control from the OPEN routine, you must process the entire DCB list. The last entry in the list can be recognized by a "1" in bit 0 of the first byte in the entry.

19. You increment the pointer to the DCB address list by four bytes. You must also increment the pointer to the work and control block area for each DCB. You increment this pointer by 8 bytes.

20. Use the load multiple instruction (LM). Note: When preparing to return to EOV you must insert a hexadecimal 03 in Register 8.

21. Use the FREEMAIN macro-instruction.

22. Return control to the OPEN or EOV routines via an XCTL macro-instruction, specifying the module to be given control as follows:

Return From	To Module
OMODVOL1	IGG0190A (OPEN)
EMODVOL1	IGG0550P (EOV)

Note: OPEN and EOV will rewind the volume upon receiving control from OMODVOL1 or EMODVOL1.

Return is via the XCTL macro-instruction (E-form). See Section 1 -- Explanation of Logic Blocks -- item 22.

INSERTING YOUR LABEL EDITOR ROUTINES INTO THE CONTROL PROGRAM

You insert your editor routines into the control program after system generation by making a linkage editor run against the system library named SYS1.SVCLIB. You will be replacing the IBM supplied editor routines OMODVOL1 and/or EMODVOL1 with your routines.

The setup for making the linkage editor run is shown below.

```
//jobname   JOB          (parameters)
//stepname  EXEC         PGM=IEWL, (parameters)
//SYSPRINT  DD           SYSOUT=A
//SYSUT1    DD           UNIT=SYSDA,SPACE=(parameters)
//SYSLMOD   DD           DSNAME=SYS1.SVCLIB,DISP=OLD
//SYSLIN    DD           *
              .
              .
              .
    (object deck for OPEN)
              .
              .
              .
ENTRY         OMODVOL1
NAME          OMODVOL1(R)
              .
              .
              .
    (object deck for EOV)
              .
              .
              .
ENTRY         EMODVOL1
NAME          EMODVOL1(R)
/*
```

Caution: You should not attempt to insert routines into the SVC library when you are running in a multi-tasking environment. Also, if your label editor routines (for OPEN or EOV) consist of more than one module you must have requested space for the SVC library directory entries for the additional modules at the time the system was generated.

APPENDIX: IECDSECT,IEFJFCBN, AND IEFUCBOB MACRO-INSTRUCTIONS

If you want to use the IECDSECT,IEFJFCBN and IEFUCBOB macro-instructions, you must either add these macro-definitions to the macro-library (SYS1.MACLIB) or place them in a separate partitioned data set and concatenate this data set to the macro-library. This section contains the following:

- The formats of the macro-instructions.
- The Job Control and Utility statements needed to add the macro-definition to the library.
- The macro-definition to be added to the library.

IECDSECT MACRO-INSTRUCTION

This macro-instruction defines the symbolic names of all fields in the work area used by the OPEN, CLOSE, TCLOSE, and EOV routines. Code this macro-instruction with blank name and operand fields, and precede it with a DSECT statement. Note: The IEFJFCBN macro-instruction is used in the assembly of IECDSECT. The macro-definition for IEFJFCBN must be present in the macro-library (SYS1.MACLIB) for successful definition of all fields in the work area.

Name	Operation	Operand
	IECDSECT	

Control Statements Required

```
//jobname      JOB     {parameters}
//stepname     EXEC    PGM=IEBUPDAT,PARM=NEW
//SYSPRINT     DD      SYSOUT=A
//SYSUT2       DD      DSNAME=SYS1.MACLIB,DISP=OLD
//SYSIN        DD      DATA
./       ADD   IECDSECT,00,0,1
                .
                .
                .
               IECDSECT Macro-Definition
                .
                .
                .
./       ENDUP
/*
```

IECDSECT Macro-Definition

```
         MACRO
         IECDSECT
         SPACE 1
*                      THIS MACRO IS USED TO DEFINE THE WORK AREA
*                           FOR ALL MODULES OF OPEN,CLOSE,TCLOSE
*                           AND END OF VOLUME FOR O/S 360
         SPACE 1
*                      THIS MACRO DEFINES A WORK AREA WITH THE
*                           FOLLOWING FORMAT
         SPACE 1
*                           1.LABELS AND DSCB
```

```
*                                   LABELS
*                                        VOLUME LABEL
*                                        FILE LABEL 1
*                                        FILE LABEL 2
*                                   DSCB
*                                        FORMAT 1
*                                        FORMAT 3 KEY
*                                        FORMAT 3 DATA
*                                        CORE ADDRESS OF NEXT DSCB
*                                   MESSAGE AREA ............... 100 BYTES
*                              2.JFCB ............................ 176 BYTES
*                              3.ECB ..............................   4 BYTES
*                              4.IOB ..............................  40 BYTES
*                              5.DEB ..............................  44 BYTES
*                              6.DCB ..............................   4 BYTES
*                              7.CCW S ............................  96 BYTES
          SPACE 1
*                                                        TOTAL *** 464 BYTES
          SPACE 2
* ***
* ***
* ***
* ***
          SPACE 1
*                         VOLUME LABEL
          SPACE 1
DXLBL     DS    0CL80
VOLLABI   DS    CL3                 LABEL IDENTIFIER
VOLNO     DS    CL1                 VOLUME LABEL NUMBER
VOLSERNO  DS    CL6
VOLSEC    DS    CL1
          DS    0CL10               RESERVED
VOLVTOC   DS    CL5
          DS    CL5
          DS    CL10          RESERVED
          DS    CL10          RESERVED
VOLOWNER  DS    CL10          OWNER NAME AND ADDRESS CODE
          DS    CL29          RESERVED
          SPACE 1
*                         FILE LABEL 1
          SPACE 1
          ORG   DXLBL
FL1LABI   DS    CL3           LABEL IDENTIFIER
FL1NO     DS    CL1           FILE LABEL NUMBER
FL1ID     DS    CL17          FILE IDENTIFIER
FL1FILSR  DS    CL6           FILE SERIAL NUMBER
FL1VOLSQ  DS    CL4           VOLUME SEQUENCE NUMBER
FL1FILSQ  DS    CL4           FILE SEQUENCE NUMBER
FL1GNO    DS    CL4           GENERATION NUMBER
FL1VNG    DS    CL2           VERSION NUMBER OF GENERATION
FL1CREDT  DS    CL6           CREATION DATE
FL1EXPDT  DS    CL6           EXPIRATION DATE
FL1FSEC   DC    C'0'          FILE SECURITY INDICATOR
FL1BLKCT  DS    CL6           BLOCK COUNT
FL1SYSCD  DS    CL13          SYSTEM CODE
FL1RES    DS    0CL7          RESERVED FOR FUTURE USE
          DS    CL1
FL1RES1   DS    CL6
          SPACE 1
*                         FILE LABEL 2
          SPACE 1
          ORG   FL1ID
FL2RECFM  DS    CL1           RECORD FORMAT
FL2BLKL   DS    CL5           BLOCK LENGTH
FL2LRECL  DS    CL5           BLOCKING FACTOR/RECORD LENGTH
```

```
FL2DEN    DS    CL1          DENSITY
FL2FILP   DS    CL1          FILE POSITION
FL2JSID   DS    0CL17        JOB/STEP IDENTIFICATION
FL2JOBD   DS    CL8          JOB IDENTIFICATION
FL2JSSP   DC    C'/'         SLASH
FL2STEPD  DS    CL8          STEP IDENTIFICATION
FL2TRTCH  DS    CL2          TAPE RECORDING TECHNIQUE
FL2CNTRL  DS    CL1          CARRIAGE CONTROL CHARACTER
FL2RES    DS    CL43         RESERVED FOR FUTURE USE
          SPACE 1
*                        DATA SET CONTROL BLOCK
          SPACE 1
          ORG   DXLBL
DXDSCB    DS    0CL96
DSCFMTID  DC    C'1'
DSCFILSR  DS    CL6          FILE SERIAL NUMBER
DSCVOLSR  DS    CL2
DSCCREDT  DS    CL3          CREATION DATE IN DISCONTINUOUS BIN
DSCEXPDT  DS    CL3          EXPIRATION DATE IN DISCONTINUOUS BIN
DSCNOEXT  DS    CL1
DSCBLDBL  DS    CL1
          DS    CL1
DSCSYSCD  DS    CL13         SYSTEM CODE
          DS    CL7
DSCFILTY  DS    CL2          FILE TYPE
DSCRECFM  DS    CL1          RECORD FORMAT
DSCOPTCD  DS    CL1          OPTION CODE
DSCBLKL   DS    CL2          BLOCK LENGTH
DSCLRECL  DS    CL2          RECORD LENGTH
DSCKEYL   DS    CL1          KEY LENGTH
DSCRKP    DS    CL2          KEY LOCATION
DSCDSIND  DS    CL1
DSCSCALO  DS    CL4
DSCLSTAR  DS    CL5
DSCTRBAL  DS    CL2
DSCEXTYP  DS    CL1          EXTENT TYPE INDICATOR
DSCEXTSQ  DS    CL1          EXTENT SEQUENCE NUMBER
DSCLOWLM  DS    CL4
DSCUPPLM  DS    CL4
DSCEXT1   DS    CL10
DSCEXT2   DS    CL10
DSCNEXT   DS    CL5          POINTER TO NEXT RECORD
DSCCORE   DS    CL4          CORE ADDRESS OF NEXT DSCB RECORD
DSCBEND   EQU   *
          SPACE 1
*                        DATA SET CONTROL BLOCK -FORMAT 3- KEY PORTION
          SPACE 1
          ORG   DXDSCB
DXDSCB3K  DS    0CL40
DSCBF3C   DC    X'03030303'
DSCBEXSK  DS    0CL40
DSCBEXTY  DS    CL1          EXTENT TYPE INDICATOR
DSCBEXSQ  DS    CL1          EXTENT SEQUENCE NUMBER
DSCBLLMT  DS    CL4          CCHH LOWER LIMIT
DSCBULMT  DS    CL4          CCHH UPPER LIMIT
DSCBEX2   DS    CL10         ADDITIONAL EXTENT
DSCBEX3   DS    CL10         ADDITIONAL EXTENT
DSCBEX4   DS    CL10         ADDITIONAL EXTENT
          SPACE 1
*                        DATA SET CONTROL BLOCK -FORMAT 3- RECORD PORTION
          SPACE 1
          ORG   DXDSCB
DSCBFMID  DC    C'3'         FORMAT ID
DSCBEXSD  DS    0CL90        ADDITIONAL EXTENTS
DSCBEX5   DS    CL10         ADDITIONAL EXTENT
```

```
DSCBEX6   DS    CL10          ADDITIONAL EXTENT
DSCBEX7   DS    CL10          ADDITIONAL EXTENT
DSCBEX8   DS    CL10          ADDITIONAL EXTENT
DSCBEX9   DS    CL10          ADDITIONAL EXTENT
DSCBEXA   DS    CL10          ADDITIONAL EXTENT
DSCBEXB   DS    CL10          ADDITIONAL EXTENT
DSCBEXC   DS    CL10          ADDITIONAL EXTENT
DSCBEXD   DS    CL10          ADDITIONAL EXTENT
DSCBNEXT  DS    CL5           CCHHR OF NEXT FORMAT 3 DSCB
          SPACE 1
*                       MESSAGE AREA
          SPACE 1
          ORG   DXDSCB
REPLYLTH  DS    CL1
REPLYADR  DS    CL3
REPLYECB  DS    CL4
MSGLSTSZ  DS    CL4
MESSAGEA  DS    CL60
REPLY     DS    CL10
*
          ORG   MESSAGEA
*
*               DEFINITION OF LENGTH OF MESSAGE COMPONENTS
MSERL     EQU   3             MESSAGE SERIAL NUMBER LENGTH
MINSTL    EQU   6             MSG INSTRUCTION LTH INC MSG SER
MUNL      EQU   3             MESSAGE UNIT NAME LENGTH
MVCLL     EQU   6             MESSAGE VOLUME SERIAL LENGTH
* MTXTL          LENGTH MAY BE DEFINED BY EACH MODULE TO FIT REQUIREMENT
* MSGLTH         LENGTH OF FULL MSG DEFINED BY EACH MODULE
*               MESSAGE FORMAT IS 'IEC000A M 000,00000  (TEXT)      '
MSGIOSUP  DC    CL3'IEC'      I/O SUPPORT MESSAGE IDENTITY
MSGSER    DS    0CL3          MESSAGE SERIAL NUMBER
          ORG   MSGSER+MSERL-1
MSGSERLO  DS    CL1           VOLUME SERIAL LO ORDER BYTE
          ORG   MSGSER
MSGINSTR  DC    CL6'000A M'   MESSAGE INSTRUCTION INCL MSGSER
          ORG   MSGINSTR+MINSTL-1
MSGACTN   DS    CL1           MESSAGE ACTION REQD BY OPERATOR
          DC    C' '
MSGUN     DC    CL3'000'      UNIT NAME THAT MSG REFERS TO
          DC    C','
MSGVOLSR  DC    CL6'000000'   VOLUME SERIAL THAT MSG REFERS TO
          DC    C','
MSGTEXT   DS    0CL38
          SPACE 1
*                       JOB FILE CONTROL BLOCK
          SPACE 1
          ORG   DSCBEND
DXJBF     DS    0CL176
          IEFJFCBN
          SPACE 1
*                       EVENT CONTROL BLOCK
          SPACE 1
DXECB     DS    0CL4
          DC    X'00000000'
          SPACE 1
*                       INPUT/OUTPUT BLOCK
          SPACE 1
DXIOB     DS    0CL32
IOBFLAG1  DC    X'00'
IOBFLAG2  DC    X'00'
IOBSENSE  DS    0H
IOBSENS0  DS    CL1
IOBSENS1  DS    CL1           SENSE BYTE 1
IOBECBPT  DS    XL1
```

```
          DC    AL3(DXECB)
IOBCSW    DS    0D
IOBCOMAD  DC    X'00000000'  KEY,0000,COMMAND ADDRESS
IOBSTAT0  DC    X'00'        STATUS BYTE 0
IOBSTAT1  DC    X'00'        STATUS BYTE 1
IOBCNT    DC    X'0000'      COUNT
IOBSIOCC  DS    XL1
IOBSTART  DC    AL3(DXCCW)
IOBWGHT   DS    XL1
IOBDCBPT  DC    AL3(DXDCB)
          DS    XL1
          DS    XL3
IOBINCAN  DC    X'0000'
IOBERRCT  DS    XL2
DXDAADDR  DS    D            DIRECT ACCESS ADDRESS (MBBCCHHR)
          SPACE 1
*                        DATA EXTENT BLOCK
          SPACE 1
DYYYY     DS    0CL44
DXDEB     EQU   DYYYY-4
DXDEBDEB  DC    X'00C00000'
DXDEBOFL  DS    0CL1
DXDEBIRB  DC    X'00000000'
DXDEBSYS  DC    X'00000000'
DXDEBUSR  DC    X'00000000'
DXDEBECB  DC    X'00000000'
DXDEBID   DS    0CL1
DXDEBDCB  DC    AL4(DXDCB)
DXDCBAD   EQU   DXDEBDCB
DXDEBAPP  DS    CL4
DXDEBMOD  DS    0CL1
DXDEBUCA  DS    F
DXDEBBIN  DS    H
DXDEBSCC  DS    H
DXDEBSHH  DS    H
DXDEBECC  DS    H
DXDEBEHH  DS    H
DXDEBNTR  DS    H
          SPACE 1
*                        DATA CONTROL BLOCK
          SPACE 1
DXXXX     DS    0F
DXDCB     EQU   DXXXX-44     POINTER TO RELATIVE BEGINNING OF DCB
DXDBDEB   DC    A (DXDEB)
          SPACE 1
*                        CHANNEL CONTROL WORDS
          SPACE 1
          CNOP  3,8
DXCCW     DS    0CL96
DXCCW1    DS    D
DXCCW2    DS    D
DXCCW3    DS    D
DXCCW4    DS    D
DXCCW5    DS    D
DXCCW6    DS    D
DXCCW7    DS    D
DXCCW8    DS    D
DXCCW9    DS    D
DXCCW10   DS    D
DXCCW11   DS    D
DXCCW12   DS    D
          SPACE 1
DSECTSIZ  EQU   464          CORE AREA REQUIRED FOR THIS MACRO
          MEND
```

IEFUCBOB MACRO-INSTRUCTION

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.

Name	Operation	Operand
	IEFUCBOB	

Control Statements Required

```
//jobname       JOB     {parameters}
//stepname      EXEC    PGM=IEBUPDAT,PARM=NEW
//SYSPRINT      DD      SYSOUT=A
//SYSUT2        DD      DSNAME=SYS1.MACLIB,DISP=OLD
//SYSIN         DD      DATA
./        ADD   IEFUCBOB,00,0,1
                .
                .
                .
                IEFUCBOB Macro-Definition
                .
                .
                .
./        ENDUP
/*
```

IEFUCBOB Macro-Definition

```
          MACRO
          IEFUCBOB
UCBOB     EQU *                        UNIT CONTROL BLOCKS
          DS    0F
SRTEJBNR  DS    XL1                        JOB INTERNAL NUMBER
SRTECHAN  DS    XL1                  ALLOC.CHANNEL MASK
UCBID     DS    XL1                  UCB IDENTIFICATION
SRTESTAT  DS    XL1                  STATUS BITS
SRTEONLI  EQU   128                        ONLINE
SRTECHGS  EQU   64                         CHANGE ONLINE/OFFLINE
SRTERESV  EQU   32                         RESERVED DEVICE
SRTEUNLD  EQU   16                         UNLOAD THIS DEVICE
SRTEALOC  EQU   8                          BIT 4 ALLOCATED
SRTEPRES  EQU   4                          BIT 5 PERMANENTLY RESIDENT
SRTESYSR  EQU   2                          BIT 6 SYSRES
*                                                OR PRIMARY CONSOLE
SRTEDADI  EQU   1                          BIT 7 DADSM INTERLOCK
*                                                OR TAPE CONTAINS
*                                                STANDARD LABELS, OR
*                                                ALTERNATE CONSOLE
UCBCHA    DS    XL1                  FLAG1 AND CHANNEL ADDRESS
UCBUA     DS    XL1                  UNIT ADDRESS
UCBFL2    DS    XL1                  FLAG2
UCBDTI    DS    XL1                  DEVICE TABLE
UCBETI    DS    XL1                  ERROR TABLE
UCBSTI    DS    XL1                  STATUS TABLE
UCBLCI    DS    XL1                  LOGICAL CHANNEL TABLE
UCBATI    DS    XL1                  ATTENTION TABLE
UCBWGT    DS    XL1                  WEIGHT
```

```
UCBNAME   DS    CL3                UNIT NAME IN 3 EBCDIC CHARACTERS
UCBTYP    DS    XL4                DEVICE TYPE
UCBTBYT1  EQU   UCBTYP                    BYTE 1 OF UCBTYPE-MODEL
UCB1FEA0  EQU   128                       BIT 0 OF OPTION FIELD
UCB1FEA1  EQU   64                        BIT 1 OF OPTION FIELD
UCB1FEA2  EQU   32                        BIT 2 OF OPTION FIELD
UCB1FEA3  EQU   16                        BIT 3 OF OPTION FIELD
UCB1FEA4  EQU   8                         BIT 4 OF OPTION FIELD
UCB1FEA5  EQU   4                         BIT 5 OF OPTION FIELD
UCB1FEA6  EQU   2                         BIT 6 OF OPTION FIELD
UCB1FEA7  EQU   1                         BIT 7 OF OPTION FIELD
UCBTBYT2  EQU   UCBTYP+1                  BYTE 2 OF UCBTYPE-OPTIONS
UCBTBYT3  EQU   UCBTYP+2                  BYTE 3 OF UCBTYPE-CLASS
UCB3TAPE  EQU   128                       BIT 0 OF CLASS - TAPE
UCB3COMM  EQU   64                        BIT 1 OF CLASS - COMMUNIC.
UCB3DACC  EQU   32                        BIT 2 OF CLASS - DIRECT AC
UCB3DISP  EQU   16                        BIT 3 OF CLASS - DISPLAY
UCB3UREC  EQU   8                         BIT 4 OF CLASS - UNIT REC.
UCB3CHAR  EQU   4                         BIT 5 OF CLASS - CHAR.READ
UCBTBYT4  EQU   UCBTYP+3                  BYTE 4 OF UCBTYPE-DEVICE
UCBLTS    DS    XL2                LAST 12*
UCBSNS    DS    XL6                SENSE INFORMATION
SRTEVOLI  DS    CL6                VOLUME SERIAL
SRTESTAB  DS    XL1                STATUS B
SRTEBSVL  EQU   128                       BIT 0 SHARED VOLUME
SRTEBVSC  EQU   64                        BIT 1 VOLUME SECURITY
SRTEBALB  EQU   32                        BIT 2 ADDIT.VOL.LABEL PROC
SRTEBPRV  EQU   16                        BIT 3 PRIVATE
SRTEBPUB  EQU   8                         BIT 4 PUBLIC
SRTEBVQS  EQU   4                         BIT 5 VOLUME TO BE QUIESCE
*                                         BIT   TO MOUNT ANOTHER
SRTEBJLB  EQU   2                         BIT 6 JOBLIB VOLUME
SRTEBNUL  EQU   1                         BIT 7 CONTROL VOLUME
SRTEDMCT  DS    XL1                DATA MANAGEMENT COUNT
SRTEFSCT  DS    XL2                FILE SEQ. COUNT
SRTEFSEQ  DS    XL2                FILE SEQ. NUMBER
UCBSQC    DS    2F                 SEEK QUEUE CONTROL WORD
UCBSKA    DS    2F                 MBBCCHHR FOR LAST SEEK
SRTEUSER  DS    XL1                CURRENT NUMBER OF USERS
SRTEECBA  DS    XL3                DA ECB ADDRESS
DATACELL  EQU *                    9 OF THESE BLOCKS WILL BE PRESENT
DCELJBNR  DS    XL1                JOB INTERNAL
DCELUSER  DS    XL1                CURRENT NUMBER OF USERS
DCELSTAB  DS    XL1                STATUS B
DCELSTAT  DS    XL1                STATUS A
DCELVOLI  DS    CL6                VOLUME SERIAL
DCELVTOC  DS    XL3                VTOC ADDRESS
DCELECBA  DS    XL3                DA ECB ADDRESS
          MEND
```

IEFJFCBN MACRO-INSTRUCTION

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.

Name	Operation	Operand
	IEFJFCBN	

Control Statements Required

```
//jobname      JOB     (parameters)
//stepname     EXEC    PGM=IEBUPDAT,PARM=NEW
//SYSPRINT     DD      SYSOUT=A
//SYSUT2       DD      DSNAME=SYS1.MACLIB,DISP=OLD
//SYSIN        DD      DATA
./       ADD   IEFJFCBN,00,0,1
                .
                .
                .
               IEFJFCBN macro-definition
                .
                .
                .
./       ENDUP
/*
```

IEFJFCBN Macro-Definition

```
         MACRO
         IEFJFCBN
INFMJFCB EQU   *
JFCBDSNM DS    CL44      DATA SET NAME
JFCBELNM DS    CL8       ELEMENT NAME OR VERSION
JFCBISDM DS    CL1       TASK SCHEDULER - DATA
*                        MANAGEMENT INTERFACE BYTE
JFCBSYSC DS    CL13      SYSTEM CODE
JFCBLTYP DS    CL1       LABEL TYPE AND USER'S-LABEL
*                        INDICATOR
         DS    CL1       NOT USED
JFCBFLSQ DS    CL2       FILE SEQUENCE NUMBER
JFCBVLSQ DS    CL2       VOLUME SEQUENCE NUMBER
JFCBMASK DS    CL8       DATA MANAGEMENT MASK
JFCBCRDT DS    CL3       DATA SET CREATION DATE
JFCBXPDT DS    CL3       DATA SET EXPIRATION DATE
JFCBIND1 DS    CL1       INDICATOR BYTE 1
JFCBRLSE EQU   64         BITS 0 AND 1 - EXTERNAL
*                        STORAGE RELEASE INDICATOR
JFCBLOCT EQU   16         BITS 2 AND 3 - DATA SET
*                        HAS BEEN LOCATED
JFCBNEWV EQU   4          BITS 4 AND 5 - NEW VOLUME
*                        ADDED TO DATA SET
JFCBPMEM EQU   1          BITS 6 AND 7 - DATA SET IS
*                        A MEMBER OF A PODS OR GDG
JFCBIND2 DS    CL1       INDICATOR BYTE 2
JFCBSTAT EQU   64         BITS 0 AND 1 - DATA SET
*                        STATUS (NEW, OLD, OR MOD)
JFCBSCTY EQU   16         BITS 2 AND 3 - DATA SET
*                        SECURITY INDICATOR
JFCBUFNO DS    0AL1
JFCBUFRQ DS    AL1
JFCBFTEK DS    0BL1
JFCBFALN DS    BL1
JFCBUFL  DS    AL2
JFCEROPT DS    BL1
JFCTRTCH DS    0BL1
JFCKEYLE DS    0AL1
JFCMODE  DS    0BL1
JFCCODE  DS    0BL1
JFCSTACK DS    0BL1
JFCPRTSP DS    BL1
```

```
JFCDEN   DS    BL1
JFCLIMCT DS    AL3
JFCDSORG DS    BL2
JFCRECFM DS    BL1
JFCOPTCD DS    BL1
JFCBLKSI DS    AL2
JFCLRECL DS    AL2
JFCNCP   DS    AL1
JFCNTM   DS    AL1
JFCRKP   DS    AL2
JFCCYLOF DS    AL1
JFCDBUFN DS    AL1
JFCINTVL DS    AL1
JFCCPRI  DS    BL1
JFCSOWA  DS    AL2
JFCBNTCS DS    CL1                NUMBER OF OVERFLOW TRACKS
JFCBNVOL DS    CL1                NUMBER OF VOLUME SERIAL
*                                 NUMBERS
JFCBVOLS DS    CL30               VOLUME SERIAL NUMBERS (THE
*                                 FIRST FIVE)
JFCBEXTL DS    CL1                LENGTH OF BLOCK OF EXTRA
*                                 VOLUME SERIAL NUMBERS
*                                 (BEYOND FIVE)
JFCBEXAD DS    CL3                TRACK ADDRESS OF BLOCK OF
*                                 EXTRA VOLUME SERIAL NUMBERS
JFCBPQTY DS    CL3                PRIMARY QUANTITY OF D.A.
*                                 STORAGE REQUIRED
JFCBCTRI DS    CL1                INDICATES WHETHER CYLINDERS
*                                 TRACKS, OR RECORDS ARE
*                                 SPECIFIED IN JFCBPQTY AND
*                                 JFCBSQTY
JFCBSQTY DS    CL3                SECONDARY QUANTITY OF D.A.
*                                 STORAGE REQUIRED
JFCBIND3 DS    CL1                INDICATOR BYTE 3
JFCBCNTG EQU   64                  BITS 0 AND 1 - CONTIGUOUS
*                                  STORAGE INDICATOR
JFCBMXIG EQU   16                 BITS 2 AND 3 - MAXIMUM
*                                  AVAILABLE EXTENT INDICATOR
JFCBALXI EQU   4                   BITS 4 AND 5 - ALL EXTENTS
*                                  INDICATOR
JFCBRNDC EQU   1                   BITS 6 AND 7 - ROUND
*                                  CYLINDER INDICATOR
JFCBDQTY DS    CL3                QUANTITY OF D.A. STORAGE
*                                 REQUIRED FOR A DIRECTORY
JFCBSPNM DS    CL3                CORE ADDRESS OF THE JFCB
*                                 WITH WHICH CYLINDERS ARE
*                                 SPLIT
JFCBABST DS    CL2                RELATIVE ADDRESS OF FIRST
*                                 TRACK TO BE ALLOCATED
JFCBSBNM DS    CL3                CORE ADDRESS OF THE JFCB
*                                 FROM WHICH SPACE IS TO BE
*                                 SUBALLOCATED
JFCBDRLH DS    CL3                AVERAGE DATA RECORD LENGTH
JFCBVLCT DS    CL1                VOLUME COUNT
JFCBSPTN DS    CL1                NUMBER OF TRACKS PER
*                                 CYLINDER TO BE USED BY THIS
*                                 DATA SET WHEN SPLIT
*                                 CYLINDERS IS INDICATED
JFCBLGTH EQU 176                LENGTH OF JFCB
JFCBEND  EQU   *
         MEND
```

EXECUTE CHANNEL PROGRAM (EXCP) MACRO-INSTRUCTION

This chapter contains a general description of the function and application of the Execute Channel Program (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.

The EXCP macro-instruction provides you with a device-dependent means of performing the I/O operations. 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 System Reference Library publications for each device.

Documentation of the internal logic of the input/output supervisor can be obtained through your IBM Branch Office.

PREREQUISITE PUBLICATIONS

The IBM System/360 Operating System: Data Management publication (Form C28-6537) explains the standard procedures for I/O processing under the operating system.

The IBM System/360 Operating System: Assembler Language publication (Form C28-6514) contains the information necessary to code programs in the assembler language.

The IBM System/360 Operating System: Control Program Services publication (Form C28-6541) describes the system macro-instructions that can be used in programs coded in the assembler language.

The IBM System/360 Operating System: System Control Block publication (Form C28-6628) contains format and field descriptions of the system control blocks referred to in this chapter.

EXECUTE CHANNEL PROGRAM (EXCP) MACRO-INSTRUCTION

Execute Channel Program (EXCP) is a macro-instruction of System/360 Operating System that causes a supervisor-call interruption to pass control to the input/output supervisor. EXCP also provides the input/output supervisor with control information regarding a channel program to be executed. When the IBM standard data access methods are being used, the access method routines are responsible for issuing EXCP. If you are not using the standard access methods, you may issue EXCP directly. Direct use of EXCP provides you with device dependence in organizing data and controlling I/O devices.

You issue EXCP primarily for I/O programming situations to which the standard access methods do not apply. When you are writing your own data access methods, you must include EXCP for I/O operations. EXCP must also be used for processing of nonstandard labels, including the reading and writing of labels and the positioning of 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 input/output 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.

USE OF EXCP IN SYSTEM AND PROBLEM PROGRAMS

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

SYSTEM USE OF EXCP

When using a standard data access method to perform I/O operations, the programmer is relieved of coding channel programs, and of 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 that 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 supplied 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 input/output supervisor will then:

5. Schedule the I/O request.

6. Issue a start input/output (SIO) instruction to activate the I/O device.

7. Process I/O interruptions and schedule error recovery procedures, when necessary.

8. Place a completion code in the event control block after the channel program has been executed.

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.

PROGRAMMER USE OF EXCP

If you wish 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 the EXCP macro-instruction. The input/output supervisor always handles items 5 through 8.

After issuing the EXCP macro-instruction, you should issue a WAIT macro-instruction specifying 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 REQUIREMENTS

This section describes the channel program that you must provide in order to issue the EXCP macro-instruction. 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 double-word 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 Systems Reference Library publications for each device. All channel command words described in these publications can be used, with the exception of REWIND and UNLOAD (RWU).

Data and Command Chaining

Chaining is the successive loading of channel command words into a channel from contiguous double-word locations in main 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 the IBM System/360: Principles of Operation publication (Form A22-6821).

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.

When 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 the EXCP macro-instruction, you must be familiar with the function and structure of an input/output block (IOB), an event control block (ECB), a data control block (DCB), and a data extent block (DEB). Brief descriptions of these control blocks follow. Their fields are illustrated in the section "EXCP Programming Specifications."

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 for synchronizing I/O operations with the problem program must be directed to 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. 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. (Unit control blocks are set up at system generation time and need not concern you.) 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 the EXCP macro-instruction has been issued.

INITIATION OF CHANNEL PROGRAM

By issuing the EXCP macro-instruction, you request the execution of the channel program specified in the input/output block. The input/output supervisor checks the request for validity by ensuring that the required control blocks contain the correct information. If they do not, abnormal termination procedures are initiated. A program check occurs if the control blocks are not on correct boundaries.

The input/output supervisor obtains the address of the data control block from the input/output block and the address of the data extent block from the data control block. From the data extent block, the system obtains the address of the unit control block (UCB) for the desired I/O device. To protect and facilitate reference to the addresses of the IOB, DEB, and UCB, the input/output supervisor places these addresses, along with other information about the channel program, into an area called a request element. The request element is used by the input/output supervisor for forming queues to keep track of I/O requests. A channel program's request element is "available" if the information it contains is no longer to be used by the input/output supervisor and if it is ready to receive information about another request. When a request element is "made available", it is removed from all request queues and placed on a queue of available request elements. You are not concerned with the contents of the request element unless you have provided appendage routines, as explained in the section "Appendages."

After completing the request element for the channel program, the input/output supervisor determines whether a channel and the requested I/O device are ready for the channel program. If they are not ready, the request element is placed into the appropriate queue, and control is returned to the problem program. The channel program is subsequently executed when the channel and device are ready.

To initiate execution of the channel program, the system obtains its address from the input/output block, places this address into the channel address word (CAW), and issues a start input/output (SIO) instruction.

Before issuing the SIO instruction for direct-access devices, the system issues the initial seek, which is overlapped with other operations. You specify the seek address in the input/output block. When the seek has completed, the system constructs a command chain to reissue the seek, set the file mask specified in the data extent block, and pass control to your channel program. (When using the operating system, you cannot issue the initial seek or set the file mask yourself.)

Before issuing SIO for magnetic tape devices, the system constructs a command chain to set the mode specified in the data extent block and pass control to your channel program. (When using the operating system, you cannot set the mode yourself.)

COMPLETION OF CHANNEL PROGRAM

The system considers the channel program completed when it receives an indication of a channel end condition. When channel end occurs, the request 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.

Device End Errors

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. It should be noted that when 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 NOPCCW 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 the IBM System/360: Principles of Operation publication. For descriptions of interruptions by specific devices, refer to IBM Systems Reference Library publications for each device.

If error conditions are associated with an interruption, the input/output supervisor schedules the appropriate device-dependent error routine. The channel is then restarted with another request that is not related[1] to the channel program in error. 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 places into the event control block.

Error Recovery Procedures for Related Channel Programs

Related channel programs are requests that are associated with a particular data control block and data extent block in the same job step. They must be executed in a definite order, i.e., the order in which the requests are received by the input/output supervisor. A channel program is not started until all previous requests for related

---

[1]Related channel programs are discussed in the next section.

channel programs have been completed. You specify, in the input/output block, whether the channel program is related to others.

If a permanent error occurs in a channel program that is related to other requests, the request elements for all the related channel programs are removed from their queue and made available. This process is called purging. The addresses of the input/output blocks for the related channel programs are chained together, with the address of the first input/output block in the chain placed into the "User Purge IOB Address" field of the data extent block. The address of the second input/output block is placed into the "Restart Address" field of the first input/output block, and so on. The last input/output block in the chain is indicated by all ones in its Restart Address field. The chain defines the order in which the request elements for the related channel programs are removed from the request queue.

For all requests that are related to the channel program in error, the system places completion codes into the event control blocks. The IFLGS field of the data control block is also flagged. Any requests for a data control block with error flags are posted complete without execution. If you wish to reissue requests that are related to the channel program in error, you must reset the first two bits of the IFLGS field of the data control block to zeros. You then issue a RESTORE macro-instruction, specifying, as the only parameter, the address of the "User Purge IOB Address" field of the data extent block. This causes execution of all the related channel programs. (The RESTORE macro-definition and how to add it to the macro-library are in the Appendix of this chapter.) Alternatively, if you wish to restart only particular channel programs rather than all of them, you may reissue the EXCP macro-instruction for each channel program desired.

APPENDAGES

This section discusses the appendages that you may optionally code when using the EXCP macro-instruction. Before a programmer-written appendage can be executed, it must be included in the SVC library. These procedures are explained first; descriptions of the routines themselves and of their coding specifications follow.

DEFINING APPENDAGES

An appendage must be defined in a DD statement as a member of a SYS1 partitioned data set. The full member name of an appendage is eight bytes in length, but the first six bytes are required by IBM standards to be the characters IGG019. The last two characters must be provided by you as an identification; they may range in collating sequence from WA to Z9.

ENTERING APPENDAGES INTO SVC LIBRARY

The SVC library is a partitioned data set named SYS1.SVCLIB. You can insert an appendage into the SVC library during the system generation process. In either case, the routine must be a member of a cataloged partitioned data set whose name begins with SYS1.

To enter a routine into the SVC library during system generation, you use the SVCLIB macro-instruction. The format of this macro-instruction is given in the publication IBM System/360 Operating System: System Generation, Form C28-6554.

CHARACTERISTICS OF APPENDAGES

An appendage is a programmer-written routine that provides additional control over I/O operations during channel program execution. By providing 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 before a start input/output (SIO) instruction is issued, or when one of the following occurs:

- Program controlled interruption.
- End of extent.
- Channel end.
- Abnormal end.

Appendages are executed in supervisor state. However, you must not issue, in an appendage, any SVC instructions or instructions that change the status of the computing or operating system (e.g., WTO, LPSW, SSM, etc.). Since appendages are disabled for all types of interruptions except machine checks, you also must not enter loops that test for completion of I/O operations. An appendage must not alter storage used by either the supervisor or the input/output supervisor.

The identification of an appendage, which consists of the last two characters of its 8-character name, must be specified in the DCB macro-instruction, as described in the section "EXCP Programming Specifications." When the OPEN macro-instruction for the data control block is issued, any appendages specified in the DCB macro-instruction are loaded into main storage. The appendages are linked to the input/output supervisor when their addresses are placed into a table of addresses called an appendage vector table. This table is always constructed by the system when OPEN is issued; if an appendage is not provided, the table contains the address of a return branch instruction to the input/output supervisor. Using the appendage vector table, the input/output supervisor branches and links to an appendage at the appropriate time. The address of the starting location of the appendage is placed into register 15.

Parameters are passed to appendages by the input/output supervisor. These parameters are contained in registers, and are as follows:

- Register 1 contains the address of the request element for the channel program. The request element contains the following information:

  Bytes 1 and 2
  are a field used by the system for linking the request element into a queue.

  Bytes 3 and 4
  indicate the address of the unit control block (UCB) for the I/O device.

  Byte 5
  indicates the identification of the task control block (TCB) for the task. (In a multitasking environment, this field is not used. It contains all zeros if the request element is not available and all ones when the request element is available.)

  Bytes 6, 7, and 8
  indicate the address of the input/output block.

  Byte 9
  indicates the priority of the request.

Bytes 10, 11, and 12
indicate the address of the data extent block.

The request element is normally 12 bytes in length; however, in a multitasking environment, it includes 4 more bytes that indicate the address of the TCB.

- Register 2 contains the address of the input/output block (IOB).
- Register 3 contains the address of the data extent block (DEB).
- Register 4 contains the address of the data control block (DCB).
- Register 7 contains the address of the unit control block (UCB).

The system places, into register 14, the address of the location in the input/output supervisor to which control is to be returned after execution of the appendage. When passing control from an appendage to the system, you may use displacements to the return address in register 14 for optional return procedures. Some of these procedures differ in their treatment of the request element associated with the channel program.

You may not change register 1 in an appendage. Register 9, if used, must be set to binary zero 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.

The types of appendages are listed in the following paragraphs, with explanations of when they are entered, how they return control to the system, and which registers they may use without saving and restoring.

Start Input/Output (SIO) Appendage

This appendage is entered before the input/output supervisor issues a start input/output (SIO) instruction for an I/O operation.

If the return address in register 14 is used to return control to the input/output supervisor, the I/O operation is executed normally. You may optionally bypass the SIO instruction and prevent execution of the channel program by using the contents of register 14 plus 4 as the return address. In this case, the channel program is not posted complete, but its request element is made available.

You may use registers 10 and 11 in a start input/output appendage without saving and restoring their contents.

Program Controlled Interruption (PCI) Appendage

This appendage is entered when a program controlled interruption occurs. At the time of the interruption, the contents of the channel status word will not have been placed in the "channel status word" field of the input/output block. The channel status word can be obtained from location 64. You must use the return address in register 14 to allow the system to proceed with normal interruption processing.

You may use registers 10 through 13 in a program controlled interruption appendage without saving and restoring their contents.

End-of-Extent 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 is placed in the "ECB code" field of the input/output block for subsequent posting in the event control block.

You may use the following optional return addresses:

- Contents of register 14 plus 4 - The channel program is posted complete, and its request element is made available.
- Contents of register 14 plus 8 - The request is tried again.

You may use registers 10 through 12 in an end-of-extent appendage without saving and restoring their contents.

Note: If an end-of-cylinder or file-protect condition occurs, the input/output 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 a file protect condition occurs and was caused by a full seek (command code=07) imbedded within a channel program, the request is flagged as a permanent error, and the abnormal end appendage is entered.

Channel End Appendage

This appendage is entered when a channel end, channel end with unit exception, or channel end with wrong length record occur without any other abnormal end conditions.

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. In the case of unit exception or wrong length record, the error recovery procedure is performed before the channel program is posted complete.

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.
- 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 "Flags 1" and "Flags 2" fields of the input/output block and set the "Error Counts" field to zero.
- Contents of register 14 plus 12 - The channel program is not posted complete, and its request element is not made available. (The request element is assumed to be used in a subsequent asynchronous exit routine.)

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

Abnormal End Appendage

This appendage is entered when a unit check, channel chaining check, program check, or protection check is detected with normal ending conditions. In the case of error conditions that can be retried by the system's error routines, the appendage is entered a second time when the input/output supervisor determines that the error is permanent.

To determine if an error is permanent, you should check the "ECB code" field of the input/output block. To determine the type of error, check the channel status word and the sense information. 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.

- 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 "Flags 1" and "Flags 2" fields of the input/output block and set the "Error Counts" field to zero.

- Contents of register 14 plus 12 - The channel program is not posted complete, and its request element is not made available. (The request element is assumed to be used in a subsequent asynchronous exit.)

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

EXCP PROGRAMMING SPECIFICATIONS

This section describes the parameters of the macro-instructions that you must use with EXCP, and the fields of the required control blocks.

MACRO-INSTRUCTIONS

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 of the EXCP macro-instruction itself, are listed and 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 the publication IBM System/360 Operating System: Control Program Services, Form C28-6541. 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 the two 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 of this category. 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 9 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
: specifies the name of the data definition (DD) statement that describes the data set to be processed.

MACRF=(E)
: specifies that the EXCP macro-instruction is to be used in processing the data set.

EXCP INTERFACE PARAMETERS:

EOEA=symbol
: specifies the 2-byte identification of an end-of-extent appendage that you have entered into the SVC library.

PCIA=symbol
: specifies the 2-byte identification of a program controlled interruption (PCI) appendage that you have entered into the SVC library.

SIOA=symbol
: specifies the 2-byte identification of a start I/O (SIO) appendage that you have entered into the SVC library.

CENDA=symbol
: specifies the 2-byte identification of a channel end appendage that you have entered into the SVC library.

XENDA=symbol
: specifies the 2-byte identification of an abnormal end appendage that you have entered into the SVC library.

Note: The full name of an appendage is eight bytes in length, but the first six bytes are required by IBM standards to be the characters IGG019. You provide the last two characters as the 2-byte identification; they may range in collating sequence from WA to Z9.

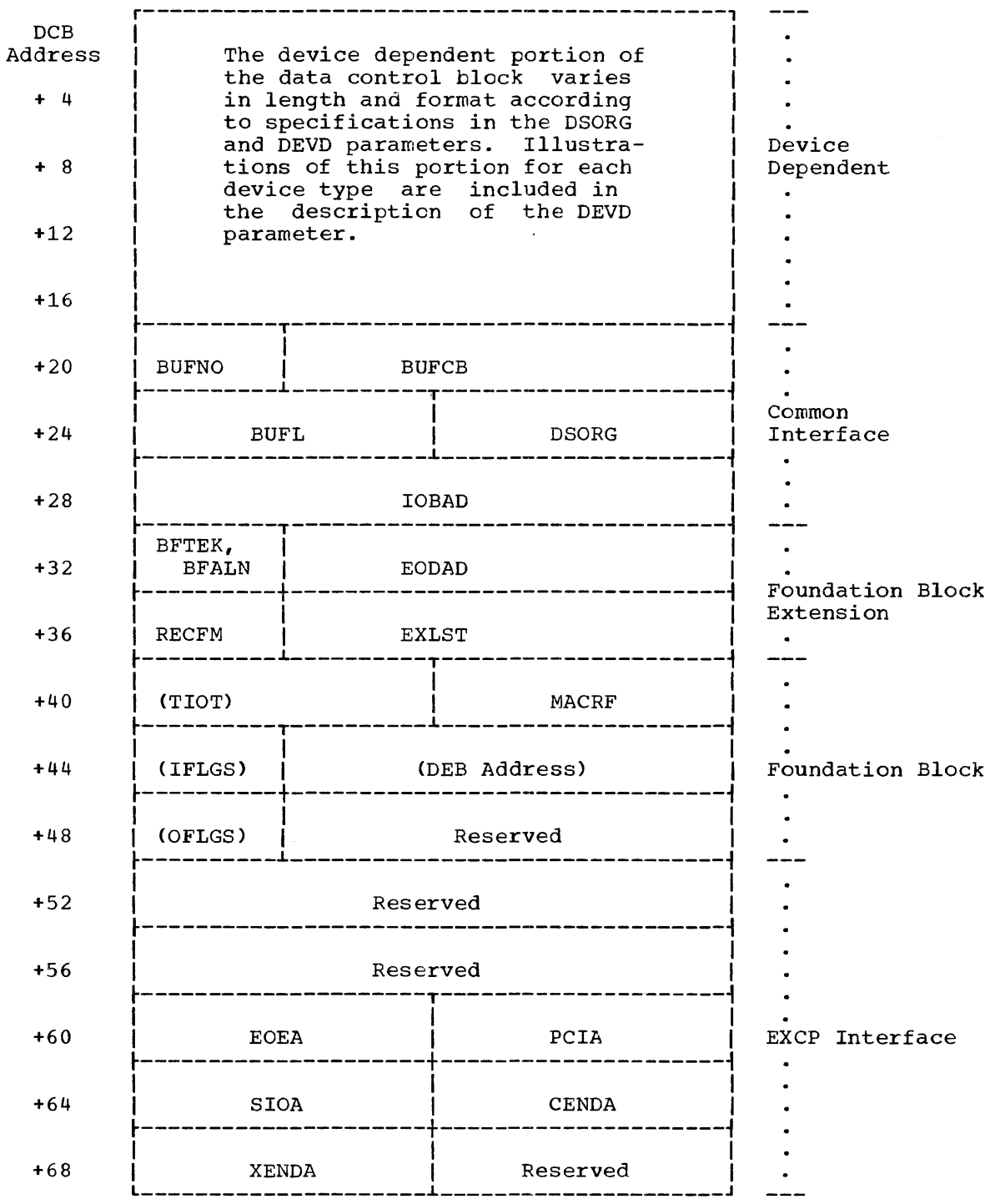


Figure 9. Data Control Block Format fcr EXCP (After OPEN)

FOUNDATION BLOCK EXTENSION AND COMMON INTERFACE PARAMETERS:

EXLST=relexp
specifies the address of an exit list that you have written for exceptional conditions. The format of this exit list is given in Appendix D of the publication IBM System/360 Operating System: Control Program Services.

EODAD=relexp
specifies the address of your end-of-data set routine. If this routine is not available when it is required, the task is abnormally terminated.

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

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

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

- The track balance (TRBAL) field, which contains a 2-byte binary number that indicates the remaining number of bytes on the current track.
- The full disk address (FDAD-MBBCCHHR) field, which indicates the location of the current record.

IOBAD=relexp
specifies 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 but provide cataloging information about the data set. This information can be used later by access method routines that read or update the data set.

RECFM=code
specifies the record format of the data set. Record format codes are given in the IBM System/360 Operating System: Control Program Services publication.

BFTEK={S|E}
specifies the buffer technique as either simple or exchange.

BFALN={F|D}
specifies the word boundary alignment of each buffer as either full word or double word.

BUFL=absexp
specifies the length in bytes of each buffer; the maximum length is 32,767.

BUFNO=absexp
specifies the number of buffers assigned to the associated data set; the maximum number is 255.

BUFCB=relexp
specifies the address of a buffer pool control block, i.e., the 8-byte field preceding the buffers in a buffer pool.

DEVICE DEPENDENT PARAMETERS:

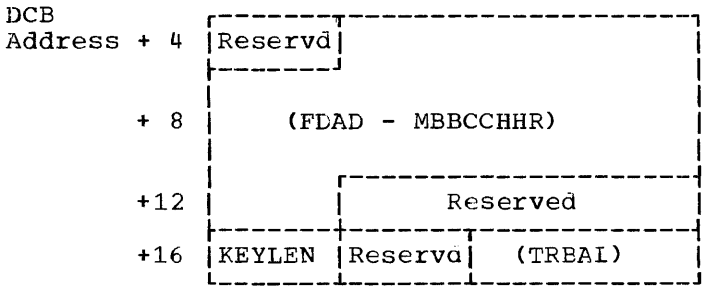
DEVD=code
specifies the device on which the data set may reside as one of the following codes. The codes are listed in order of descending space requirements for the data control block:

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

Note: 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.

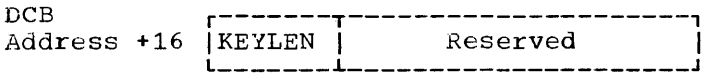
The following diagrams illustrate the device dependent portion of the data control block for each device type specified in the DEVD parameter, and for each data set organization specified in the DSORG parameter. Fields that correspond to device dependent parameters in addition to DEVD are indicated by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

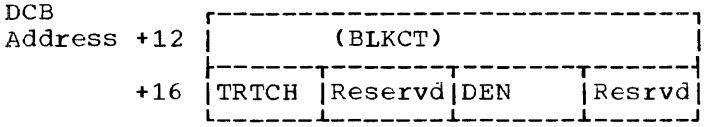


Note: For output data sets, the system uses the contents of the full disk address (FDAD-MBBCCHHR) field plus one to write a file mark when the data control block is closed, provided the track balance (TRBAL) field indicates that space is available. You must maintain the contents of these two fields yourself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

Device dependent portion of data control block when DEVD=DA and DSORG=IS or DA:

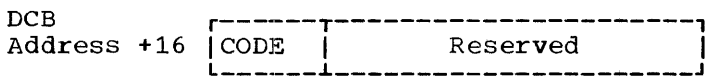


Device dependent portion of data control block when DEVD=TA and DSORG=PS:

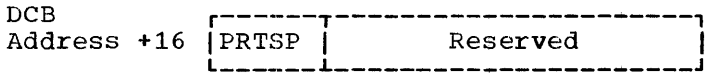


Note: For output data sets, the system uses the contents of the block count (BLKCT) field to write the block count in trailer labels when the data control block is closed, or when the EOV macro-instruction is issued. You must maintain the contents of this field yourself if the system is to write the correct block count.

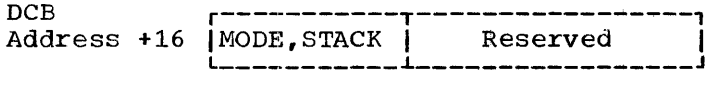
Device dependent portion of data control block when DEVD=PT and DSORG=PS:



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 parameters pertain to specific devices and may be specified only when the DEVD parameter is specified.

KEYLEN=value
specifies, for direct-access devices, the length in bytes of the

The following parameters pertain to specific devices and may be specified only when the DEVD parameter is specified.

KEYLEN=value
specifies, 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
specifies, for paper tape, the code in which records are punched as follows:

Value	Code
I	IBM BCD
F	Friden
B	Burroughs
C	National Cash Register
A	ASCII
T	Teletype
N	no conversion (format F records only)

If this parameter is omitted, N is assumed.

DEN=value
specifies, for magnetic tape, the tape recording density in bits per inch as follows:

Value	Density	
	Model 2400 7-track	Model 2400 9-track
0	200	-
1	556	-
2	800	800

If this parameter is omitted, the lowest density is assumed.

TRTCH=value
specifies, for 7-track magnetic tape, the tape recording technique as follows:

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
specifies, for a card reader or punch, the mode of operation. Either C (column binary mode) or E (EBCDIC code) may be specified.

STACK=value
specifies, for a card punch or card reader, the stacker bin to receive cards as either 1 or 2.

PRTSP=value
specifies, for a printer, the line spacing as either 0, 1, 2, or 3.

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. 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 data control block.
- Verification or creation of standard labels.
- Tape positioning.
- Loading of programmer-written appendage routines.

The three parameters of the OPEN macro-instruction are:

dcb-addr
specifies the address of the data control block to be initialized. (More than one data control block may be specified.)

opt_1
specifies 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.

opt_2
specifies 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 — The disposition indicated on the DD statement is tested and appropriate positioning provided. This service is assumed if this operand is omitted and volume positioning is applicable. If there is no disposition specified in the DD statement when this operand is specified, LEAVE is assumed.

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 only parameter of the EXCP macro-instruction is:

iob-addrx
specifies the address, or a register that contains the address of the input/output block of the channel program to be executed.

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. 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 tape mark is read or a reflective spot is written over. In these cases, bit settings in the 1-byte OFLGS 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 OFLGS. Bit positions 2,3,6, and 7 of OFLGS are used only by the system; you are concerned with bit positions 0,1,4, and 5. The use of these OFLGS bit positions is as follows:

Bit 0
indicates that a tape mark is to be written.

Bit 1
indicates that a backwards 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 OFLGS are both off when EOV is executed, the tape is spaced past a tape mark, 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 tape mark 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.

When issuing EOV for sequentially organized output data sets on direct-access volumes, you can determine whether additional space has been obtained on the same or a different volume. You do this by checking the volume serial number in the unit control block (UCB) both before and after issuing EOV.

The only parameter of the EOV macro-instruction is:

dcb-addrx
: specifies 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.

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.

The two parameters of the CLOSE macro-instruction are:

dcb-addr
: specifies the address of the data control block to be restored. More than one data control block may be specified.

opt
: specifies the type of volume disposition intended for the data set. You may specify this parameter as either LEAVE or REREAD. The corresponding volume disposition when CLOSE is executed is as follows:

  LEAVE - Volume is positioned at logical end of data set.

  REREAD - Volume is positioned at logical beginning of data set.

  DISP - The disposition indicated on the DD statement is tested, and appropriate positioning is provided. This service is assumed if this operand is omitted and volume positioning is applicable. If there is no disposition specified in the DD statement when this operand is specified, LEAVE is assumed.

  This parameter is ignored if specified for volumes other than magnetic tape or direct-access.

Note: When CLOSE is issued for data sets on magnetic tape volumes, labels are processed according to bit settings in the OFLGS field of the data control block. Before issuing CLOSE for magnetic tape, you must set the appropriate bits in OFLGS. The OFLGS bit positions that you are concerned with are listed in the description of the EOV macro-instruction.

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 instruction in the section "EXCP Programming Specifications."

Input/Output Block Fields

The input/output block is not automatically constructed by a macro-instruction; it must be defined as a series of constants and must be on a full-word boundary. For nondirect-access devices, the input/output block is 32 bytes in length. For direct-access devices, 8 additional bytes must be provided.

In Figure 10, the shaded areas indicate fields in which you must specify information. The other fields are used by the system and must be defined as all zeros. You may not place information into these fields, but you may examine them.

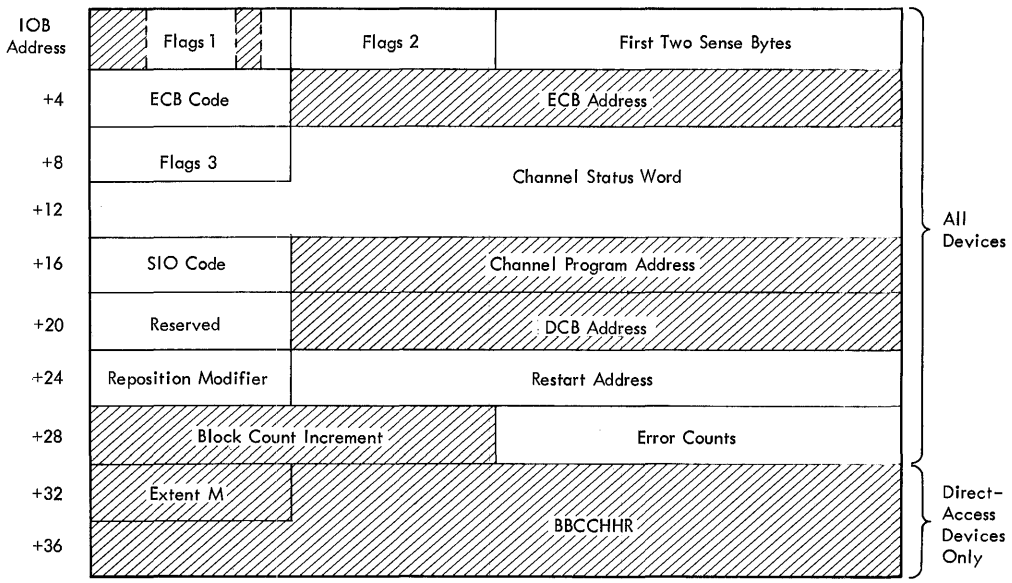


Figure 10. Input/Output Block Format

Flags 1 (1 byte)
specifies the type of channel program. 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 2311, 2671, 1052, and 2150.) A one bit in position 6 indicates that the channel program is not related to any other channel program. Bit positions 2, 3, 4, 5, and 7 are used only by the system.

Flags 2 (1 byte)
is used only by the system.

First Two Sense Bytes (2 bytes)
are placed into the input/output block by the system when a unit check occurs.

ECB Code (1 byte)
indicates 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."

ECB Address (3 bytes)
specifies the address of the 4-byte event control block that you have provided.

Flags 3 (1 byte)
is used only by the system.

Channel Status Word (7 bytes)
indicates the low order seven bytes of the channel status word, which are placed into this field each time a channel end occurs.

SIO Code (1 byte)
indicates, in the four low-order bits, the instruction length and condition code for the SIO instruction that the system issues to start the channel program.

Channel Program Address (3 bytes)
specifies the starting address of the channel program to be executed.

Reserved (1 byte)
is used only by the system.

DCB Address (3 bytes)
specifies the address of the data control block of the data set to be read or written by the channel program.

Reposition Modifier (1 byte)
is used by the system for volume repositioning in error recovery procedures.

Restart Address (3 bytes)
is used by the system to indicate the starting address of a channel program that performs special functions for error recovery procedures. The system also uses this field in procedures for making request elements available, as explained under "Error Recovery Procedures for Related Channel Programs."

Block Count Increment (2 bytes)
specifies, for magnetic tape, the amount by which the block count (BLKCT) 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.

Error Counts (2 bytes)
indicates the number of retries attempted during error recovery procedures.

Extent M (1 byte)
specifies, for direct-access or telecommunications devices, which extent entry in the data extent block is associated with the channel program. (0 indicates the first extent; 1 indicates the second, etc.)

BBCCHHR (7 bytes)
specifies, for direct-access devices, the seek address for the programmer's channel program.

Event Control Block Fields

You must define an event control block as a 4-byte area on a full-word boundary. When the channel program has been completed, the input/output supervisor places a completion code containing status information into the event control block (Figure 11). Before examining this information, you must test for the setting of the "Complete Bit." If the complete bit is not on, and the problem program cannot perform other useful operations, you should issue a WAIT macro-instruction that specifies the event control block. Under no circumstances may you construct a program loop that tests for the complete bit.

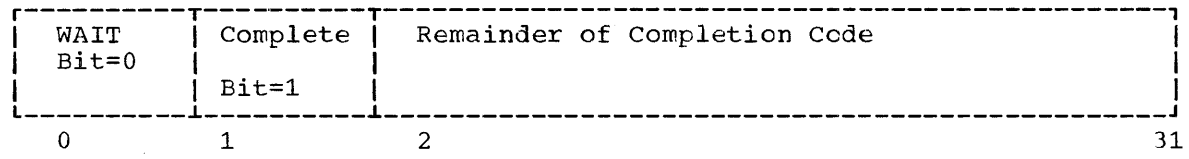


Figure 11. Event Control Block After Posting of Completion Code

WAIT Bit
A one bit in this position indicates that the WAIT macro-instruction has been issued, but that 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	Interpretation
7F000000	Channel program has terminated without error.
41000000	Channel program has terminated with permanent error.
42000000	Channel program has terminated because a direct-access extent address has been violated.
44000000	Channel program has been intercepted because of permanent error associated with device end for previous request. You may reissue the intercepted request.
48000000	Request element for channel program has been made available after it has been purged.
4F000000	Error recovery routines have been entered because of direct-access error but are unable to read home address or record 0.

Data Extent Block Fields

The data extent block 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 data extent block, but you may examine them. The Data Extent Block format and field description is contained in the System Control Block publication.

APPENDIX: RESTORE MACRO-INSTRUCTION

If you want to use the RESTORE macro-instruction, you must either add the macro-definition to the macro-library (SYS1.MACLIB) or place it in a separate partitioned data set and concatenate this data set to the macro-library. This section contains the following:

- The format of the macro-instruction.
- The Job Control and Utility statements needed to add the macro-definition to the library.
- The macro-definition to be added to the library.

RESTORE Macro-Instruction

This macro-instruction is used to return purged request elements to the request queues. The format of this macro-instruction is as follows:

Name	Operation	Operand
	RESTORE	User Purge IOB Address

The user purge IOB address is in the data extent block (DEB).

Control Statements Required

```
//jobname      JOB     {parameters}
//stepname     EXEC    PGM=IEBUPDAT,PARM=NEW
//SYSPRINT     DD      SYSOUT=A
//SYSUT2       DD      DSNAME=SYS1.MACLIB,DISP=OLD
//SYSIN        DD      DATA
./       ADD  RESTORE,00,0,1
                .
                .
                .
              RESTORE Macro-Definition
                .
                .
                .
./       ENDUP
/*
```

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
```

EXECUTE DIRECT ACCESS PROGRAM (XDAP) MACRO-INSTRUCTION

This chapter explains what the Execute Direct-Access Program (XDAP) macro-instruction does and how you can use it. The control block generated when XDAP is issued and the macro-instructions used with XDAP are also discussed.

The 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. Since most of the specifications for XDAP are similar to those for the Execute Channel program (EXCP) macro-instruction, it is recommended that you be familiar with the "EXCP Macro-Instruction" chapter of this publication, as well as with the information contained in the required publication.

PREREQUISITE PUBLICATION

The IBM System/360 Operating System: Data Management publication (Form C28-6537) explains the standard procedures for I/O processing under the operating system.

EXECUTE DIRECT ACCESS PROGRAM (XDAP) MACRO-INSTRUCTION

Execute Direct Access Program (XDAP) is a macro-instruction of System/360 Operating System 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.

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 much less main storage space 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.

REQUIREMENTS FOR EXECUTION OF DIRECT-ACCESS PROGRAM

Before issuing the XDAP macro-instruction, you must issue a DCB macro-instruction, which produces a data control block (DCB) for the data set to be read or updated. You must also issue an OPEN macro-instruction, which initializes the data control block and produces a data extent block (DEB).

When the XDAP macro-instruction is issued, another control block, containing both control information and executable code, is 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 channel command words (CCWs). The type of channel program generated depends on specifications in the parameters of the XDAP macro-instruction.

After this XDAP control block is constructed, the direct-access channel program is executed. A block is located by either its actual address or its key, and is either read or updated.

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 event control block to determine whether 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.

XDAP PROGRAMMING SPECIFICATIONS

MACRO-INSTRUCTIONS

When you are using the XDAP macro-instruction, you must also issue DCB, OPEN, CLOSE, and, in some cases, the EOV macro-instruction. 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 the "EXCP Macro-Instruction" section of this publication for listings of their parameters.

DCB -- Define Data Control Block

The EXCP form of the DCB macro-instruction produces a data control block that can be used with the XDAP macro-instruction. You must issue a DCB macro-instruction for each data set to be read or updated by the direct-access channel program. The "EXCP Macro-Instruction" section of this publication contains a diagram of the data control block, as well as a listing of the parameters of the DCB macro-instruction.

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 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:

Operation	Operand
XDAP	ecb-symbol,type-{R|W|V}{I|K},dcb-addr,area-addr ,length-value,[(key-addr,keylength-value)],blkref-addr

ecb-symbol
: specifies the symbolic name to be assigned to the XDAP control block.

type-{R|W|V}{I|K}
: specifies 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. The codes and their meanings are as follows:

R - Read a block.
W - Write a block.
V - Verify 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 or written.)
K - Locate a block by key. (Only the data portion of the block is read or written.)

dcb-addr
specifies the address of the data control block of the data set.

area-addr
specifies the address of an input or output area for a block of the data set.

length-value
specifies 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 32767.

key-addr
specifies, when blocks are to be located by key, the address of a main storage field that contains the key of a block to be read or overwritten.

keylength-value
specifies, when blocks are to be located by key, the length of the key. The maximum length is 255 bytes.

blkref-addr
specifies the address of a main storage field containing the actual device address of the track containing the block to be located. When blocks are to be located by key, this field is seven bytes in length; when blocks are to be located by identification, an eighth byte indicating block identification must be included in this field. (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 indicates the bin number of direct-access volume; CC indicates the cylinder address; HH indicates the actual track address; and R indicates the block identification.)

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 only parameter of the CLOSE macro-instruction is the address of the data control block to be restored. (More than one data control block may be specified.)

THE XDAP CONTROL BLOCK

The three portions of the control block generated during execution of the XDAP macro-instruction are described here.

Event Control Block (ECB)

The event control block begins on a full word boundary and occupies the first 4 bytes of the XDAP control block. Each time the direct-access channel program terminates, the input/output supervisor places a completion code containing status information into the event control block (Figure 12). Before examining this information, you must test for the setting of the "Complete Bit" by issuing a WAIT macro-instruction specifying the event control block.

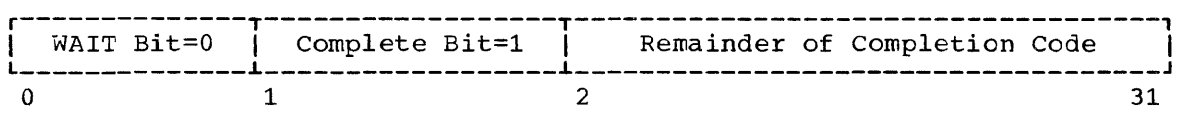


Figure 12. Event Control Block After Posting of Completion Code

WAIT Bit
A one bit in this position indicates that the WAIT macro-instruction has been issued, but 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.
44000000	Channel program has been intercepted because of permanent error associated with device end for previous request. You may reissue the intercepted request.

48000000	Request element for channel program has been made available after it has been purged.
4F000000	Error recovery routines have been entered because of direct-access error but are unable to read home address or record 0.

Input/Output Block (IOB)

The input/output block is 40 bytes in length and immediately follows the event control block. The section "EXCP Macro-Instruction" of this publication contains a diagram of the input/output block. The only fields with which the user of XDAP is concerned are the "First Two Sense Bytes" and "Channel Status Word" fields. You may wish to examine these fields when a unit check condition or an I/O interruption occurs.

Direct-Access Channel Program

The direct-access channel program is 24 bytes in length 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 13.

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
	1	Search ID Equal
Write by Identification	2	Transfer in Channel
	3	Write Key and Data
	1	Search Key Equal
Write by Key	2	Transfer in Channel
	3	Write Data
[1]For verifying operations, the third CCW is flagged to suppress the transfer of information to main storage.		

Figure 13. The XDAP Channel Programs

XDAP OPTIONS

CONVERSION OF RELATIVE TRACK ADDRESS TO ACTUAL 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 is in the communication vector table (CVT). The address of the CVT is in location 16. (The CVT macro-instruction defines the symbolic names of all fields in the CVT. The macro-definition and how to add it to the macro-library are in the Appendix of this chapter.)

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 the data set. (0 indicates the first or only data set in the concatenation, 1 indicates the second, etc.)
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 indicates the bin number of the direct-access volume; CC indicates the cylinder address; HH indicates the actual track address; and R indicates the block identification.
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.

APPENDAGES

For additional control over I/O operations, you may write appendages, which must be entered into the SVC library. Descriptions of these routines and their coding specifications are contained in the "EXCP Macro-Instruction" section of this publication.

L- AND E- FORMS OF XDAP MACRO-INSTRUCTION

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. The L- and E- forms are described in Appendix B of the IBM System/360 Operating System: Control Program Services, Form C28-6541.

Note: The BLKREF parameter is ignored by the "L" form of the XDAP macro-instruction. The field may be supplied in the E-form of the macro-instruction or moved into the IOB by you.

APPENDIX: CVT MACRO-INSTRUCTION

If you want to use the CVT macro-instruction, you must add the macro-definition to the macro-library (SYS1.MACLIB). This section contains the following:

- The format of the CVT macro-instruction.
- The Job Control and Utility statements needed to add the macro-definition to the library.
- The macro-definition to be added to the library.

Format of the CVT Macro-Instruction

This macro-instruction defines the symbolic names of all fields in the communication vector table (CVT). When coding this macro-instruction, you must precede it with a DSECT statement. The format of the macro-instruction is as follows:

Name	Operation	Operand
		VMS
	CVT	SYS= INT
		MIN

You specify, in the operand field, the control program that you are using.

VMS
designates multiprogramming with a variable number of tasks.

INT
designates multiprogramming with a fixed number of tasks.

MIN
designates the primary control program.

If the operand field is blank, it is assumed that you are using multiprogramming with a variable number of tasks.

Control Statements Required

```
//jobname       JOB     {parameters}
//stepname      EXEC    PGM=IEBUPDAT,PARM=NEW
//SYSPRINT      DD      SYSOUT=A
//SYSUT2        DD      DSNAME=SYS1.MACLIB,DISP=OLD
//SYSIN         DD      DATA
./        ADD   CVT,00,0,1
                .
                .
                .
                CVT Macro-Definition
                .
                .
                .
./        ENDUP
/*
```

CVT Macro-Definition

```
          MACRO
&NAME     CVT   &SYS=VMS,&FETCH=,&SCHED=PSS,&OPTIONS=(QTAM,INTERVAL)
```

```
*                                THIS IS A KEY WORD MACRO GENERATED SET OF
*                                CODE.  THE STANDARD FORMAT IS DEFINED AS-
                                      SPACE
*                                           CVT SYS=VMS
                                      SPACE
*                                THE THREE POSSIBLE VALUES OF THE KEYWORD
*                                SYS ARE-
*                                           VMS OPTION 4.MULTIPROGRAMMING
*                                                        WITH VARIABLE NO.
*                                                        OF TASKS
*                                           INT OPTION 2.MULTIPROGRAMMING
*                                                        WITH FIXED NO. OF
*                                                        TASKS
*                                           MIN PCP      PRIMARY CONTROL
*                                                        PROGRAM
         SPACE
CVTPTR   EQU   16              LOCATION OF THE CVT POINTER
         SPACE
&NAME    DS    0F
CVTTCBP  DC    V(IEATCBP)      ADDRESS OF A DOUBLE WORD, THE FIRST CON-
*                               TAINING THE NEXT-TO-BE DISPATCHED TCB
*                               ADDRESS, THE SECOND CONTAINING THE LAST
*                               (CURRENT) TCB ADDRESS.
CVT0EF00 DC    V(IEA0EF00)     ADDRESS OF ROUTINE TO SCHEDULE ASYNC.
*                               EXITS.
CVTLINK  DC    V(IEFLINK)      ADDRESS OF THE LINK LIBRARY DCB.
CVTJOB   DC    V(IEFJOB)       JOB SCHEDULER OPEN'ED DCB ADDRESS.
CVTBUF   DC    A(0)
CVTXAPG  DC    V(IECXAPG)      IOS APPENDAGE TABLE ADDRESS.
CVT0VL00 DC    V(IEA0VL00)     ENTRY POINT ADDRESS OF THE SUPERVISOR'S
*                               ADDRESS VALIDITY CHECKING ROUTINE.
CVTPCNVT DC    V(IECPCNVT)     ENTRY POINT ADDRESS OF THE ROUTINE WHICH
*                               CONVERTS RELATIVE TRACK ADDR. TO ABSOLUTE
CVTPRLTV DC    V(IECPRLTV)     ENTRY POINT ADDRESS OF THE ROUTINE WHICH
*                               CONVERTS ABSOLUTE TRACK ADDR. TO RELATIVE
CVTILK1  DC    V(IECILK1)      ADDRESS OF THE CHANNEL AND CONTROL UNIT
*                               PORTIONS OF THE TABLE LOOKUP.
CVTILK2  DC    V(IECILK2)      ADDRESS OF TWO BYTE UCB POINTERS
CVTXTLER DC    V(IECXTLER)     ENTRY POINT OF THE XCTL ROUTINE FOR THE
*                               ERROR TRANSIENT AREA.
CVTSYSAD DC    A(0)            SYSTEM RESIDENCE VOLUME ENTRY IN THE UCB
*                               TABLE.
CVTBTERM DC    V(IEA0AB00)     ENTRY POINT ADDRESS OF THE ABTERM ROUTINE
CVTDATE  DC    F'0'            CURRENT DATE IN PACKED DECIMAL.
CVTMSLT  DC    V(IEEMSLT)      ADDRESS OF MASTER SCHEDULER LINKAGE TABLE
CVTZDTAB DC    V(IECZDTAB)     ADDR. OF I/O DEVICE CHARACTERISTIC TABLE.
CVTXITP  DC    V(IECXITP)      ADDR. OF ERROR INTERPRETER ROUTINE.
CVTXWTO  DC    V(IECIWTST)     ADDR OF CONSOLE OUTPUT ROUTINE
CVT0FN00 DC    V(IEA0FN00)     ENTRY POINT ADDRESS OF THE FINCH ROUTINE.
CVTEXIT  SVC   3               AN SVC EXIT INSTRUCTION.
CVTBRET  BCR   15,14           A BCR 15,14 INSTRUCTION FOR DATA MGMNT.
CVTSVDCB DC    V(IEASVDCB)     ADDRESS OF THE SVC LIBRARY DCB.
         AIF   ('&OPTIONS(2)' EQ 'INTERVAL').NEXT1
CVTTPC   DC    A(0)
         AGO   .NEXT2
.NEXT1   ANOP
CVTTPC   DC    V(IEATPC)       ADDR OF TIMER PSEUDO CLOCKS (SHPC FIRST)
.NEXT2   ANOP
CVTPBLDL DC    V(IECPBLDL      BRANCH AND LINK ENTRY TO THE BLDL ROUTINE
CVTSJQ   DC    V(IEESJQ)       POINTER TO SELECTED JOB QUEUE.
CVTCUCB  DC    V(IEECUCB)      POINTER TO UCB FOR CURRENT CONSOLE.
         AIF   ('&OPTIONS(2)' EQ 'INTERVAL').NEXT3
CVTQTE00 DC    A(0)
CVTQTD00 DC    A(0)
         AGO   .NEXT4
```

```
.NEXT3    ANOP
CVTQTE00  DC    V(IEAQTE00)     ADDR OF TIMER ENQUEUE ROUTINE
CVTQTD00  DC    V(IEAQTD00)     ADDR OF TIMER DEQUEUE ROUTINE
.NEXT4    ANOP
CVTSTB    DC    V(IECSTB)       ADDRESS OF I/O DEVICE STATISTICS TABLE.
CVTDCB    DC    V(IFBDCB)       DCB ADDR.FOR LOGREC DATA SET USED BY SER.
CVTIOQET  DC    V(IECIOQET)     ADDRESS OF I/O QUEUE ELEMENT TABLE.
CVTIXAVL  DC    V(IECIXAVL)     POINTER TO NEXT AVAILABLE I/O QUEUE ELMNT
CVTNUCB   DC    A(0)            LOWEST ADDRESS NOT IN THE NUCLEUS--- AT
*                               A 2K BOUNDARY IF PROTECTION---AT A DOUBLE
*                               WORD BOUNDARY IF NO PROTECTION.
CVTFBOSV  DC    V(IEWFBOSV)     ADDRESS OF PROGRAM FETCH ROUTINE.
CVTODS    DC    V(IEA0DS)       ENTRY POINT ADDRESS OF THE DISPATCHER.
CVTILCH   DC    V(IECILCH)      ADDRESS OF THE LOGICAL CHANNEL TABLE.
CVTIERLC  DC    V(IECIERLC)     ADDRESS OF THE ERROR LOGICAL CHAN.QUEUE.
          AIF   ('&SCHED' EQ 'PPS' OR '&SCHED' EQ 'PSS').NEXT5
CVTMSER   DC    A(0)
          AGO   .NEXT6
.NEXT5    ANOP
CVTMSER   DC    V(IEEMSER)
.NEXT6    ANOP
CVT0PT01  DC    V(IEA0PT01)     BRANCH ENTRY POINT TO POST ROUTINE.
          AIF   ('&OPTIONS(1)' EQ 'QTAM').NEXT7
CVTTRMTB  DC    A(0)
          AGO   .NEXT8
.NEXT7    ANOP
CVTTRMTB  DC    V(IECTRMTB)     ADDR OF TERMINAL TABLE PRESENT IN SYSTEMS
*                               WITH QTAM
.NEXT8    ANOP
CVTHEAD   DC    V(IEAHEAD)      ADDRESS OF THE FIRST (HIGHEST PRIORITY)
*                               TCB IN THE READY QUEUE.
CVTMZ00   DC    F'0'            MACHINE SIZE (HIGHEST STORAGE ADDRESS FOR
*                               THIS MACHINE).
          AIF   ('&SYS' EQ 'MIN').SKPEND
*** THE FOLLOWING ENTRIES ARE FOR INTERMEDIATE SYSTEMS ***
          DC    A(0)            UNUSED, FORMERLY CIRBLINKAGE
CVTQOCR   DC    V(IEAQOCR)      ENTRY POINT TO OPERATOR COMMUNICATIONS
*                               QUEUE PURGING ROUTINE.
CVTQMWR   DC    V(IEFQMWR)      SYSOUT-CDA AREA ADDR USED BY THE NON-
*                               RESIDENT QUEUE MGR
          AIF   ('&SYS' EQ 'INT').SKPEND
*** THE FOLLOWING ENTRIES ARE FOR VMS SYSTEMS ***
CVTPKTBL  DC    V(IEAPKTBL)     PROTECT KEY TABLE ADDR
CVTQCDSR  DC    V(IEAQCDSR)     ADDR OF CONTENTS DIRECTORY SEARCH ROUTINE
CVTQLPAQ  DC    V(IEAQLPAQ)     ADDR. OF THE TOP C.D. ENTRY IN LPA QUEUE.
CVTQERA   DC    V(IEAQERA)      ENTRY POINT TO EOT ERASE PHASE.
CVTQPGTM  DC    V(IEAQPGTM)     ENTRY POINT TO EOT PURGE TIMER ELMNTS.RTN
CVTQPIE   DC    V(IEAQPIE)      ENTRY PT TO EOT PIERMVE RT.
CVTQABL   DC    V(IEAQABL)      ENTRY PT TO EOT CDABDEL RT.
CVTQSPET  DC    V(IEAQSPET)     ENTRY PT TO EOT SPEOT RT.
CVTQABST  SVC   13              SVC TO ABEND
          DC    H'0'            FILL OUT TO FULL WORD
.SKPEND   ANOP
          MEND
```

HOW TO USE THE TRACING ROUTINE

This chapter describes the function of the tracing routine, and provides a detailed description of the information made available by the tracing routine.

Before reading this chapter, you should be familiar with the information contained in the prerequisite publication.

PREREQUISITE PUBLICATION

The IBM System/360: Principles of Operation publication (Form A22-6821) contains information about the SIO instruction and the I/O and SVC interruptions.

HOW TO USE THE TRACING ROUTINE

The tracing routine is an Operating System/360 optional feature which you can use as a debugging and maintenance aid. The tracing routine stores, in a table, information pertaining to the following conditions:

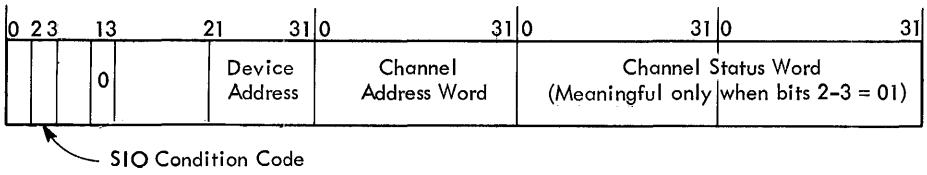
- SIO instruction execution.
- SVC interruption.
- I/O interruption.

You can include the tracing routine and its table in the control program during the system generation process. This is done using the TRACE option in the SUPRVSOR macro-instruction. The format of this option requires you to supply the number of entries in the table. Each table entry can contain information relating to one of the traced conditions. When the last entry in the table is filled, the next entry will overlay the first.

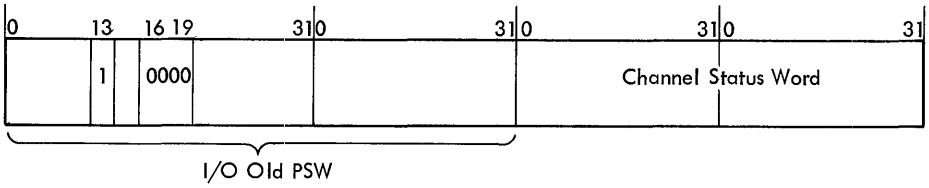
Table Entry Formats

Table entry formats are as follows:

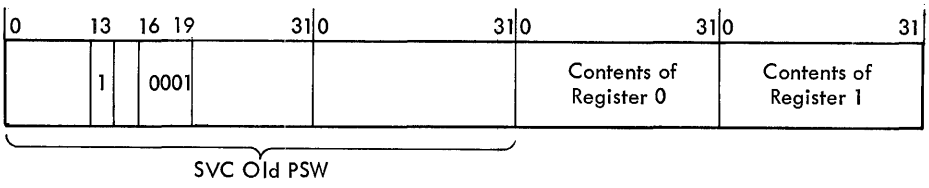
SIO Instruction



I/O Interruption

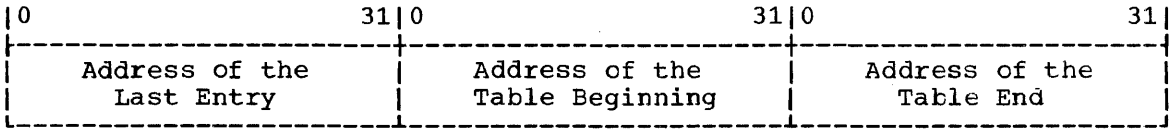


SVC Interruption



Location of the Table

The addresses of the last entry made in the table, the beginning of the table, and the end of the table are contained in a 12-byte field. The address of this field is contained in the full word starting at location 20. The format of the field is as follows:



The tracing routine is bypassed during abnormal termination procedures, except when incorporated in MFT or MVT configurations of the operating system.

IMPLEMENTING DATA SET PROTECTION

To use the data set protection feature of the operating system, you must create and maintain a data set, named PASSWORD, consisting of records that associate the names of protected data sets with the password designated for each data set. This chapter provides the information you need to create the PASSWORD data set, and describes operating characteristics of the data set protection feature.

Recommended Publications

The IBM System/360 Operating System: Data Management publication (Form C28-6537) contains a general description of the data set protection feature.

The IBM System/360 Operating System: Operator's Guide publication (Form C28-6540) contains a description of the operator messages and replies associated with the data set protection feature.

The IBM System/360 Operating System: Job Control Language publication (Form C28-6539) contains a description of the data definition (DD) statement parameter used to indicate that a data set is to be placed under protection.

Documentation of the operating system routines supporting data set protection can be obtained through your IBM Branch Office.

IMPLEMENTING DATA SET PROTECTION

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 one record for each data set placed under protection. In turn, each record contains a data set name, the 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."

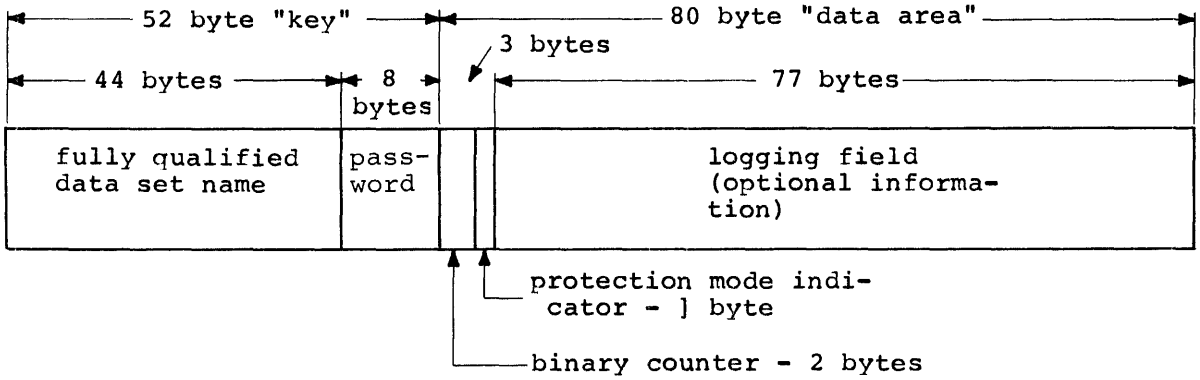
You must write routines to create and maintain the PASSWORD data set. These routines may be placed in your own library or the system's linkage editor library (SYS1.LINKLIB). You may use a data management access method or EXCP programming to handle the PASSWORD data set.

If a data set is to be placed under protection, it must have a protection indicator set in its label (DSCB or header 1 tape label). This is done by the operating system when the data set is created. The protection indicator is set in response to an entry in the LABEL= parameter of the DD statement associated with the data set being placed under protection. The Job Control Language publication describes the entry Note: Data sets on magnetic tape are protected only when standard labels are used.

The balance of this chapter discusses the PASSWORD data set characteristics and record format, the creation of protected data sets, and operating characteristics of the data set protection feature.

PASSWORD DATA SET CHARACTERISTICS AND RECORD FORMAT

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 is dependent on the number of data sets your installation desires to place under protection. The organization of the PASSWORD data set is physical sequential, and the content is unblocked format-F records, 132 bytes in length (key area plus data area). The following illustration shows the password records as you would build them in a 132 byte work area. Explanation of the fields follows the illustration.



The name of the protected data set being opened and the password entered by the operator are matched against the 52-byte "key area." The

data set name and the password must be left-justified in their areas and any unused bytes filled with blanks (X'40'). The password assigned may be from one to eight alphameric characters.

The operating system increments the binary counter by one each time the data set is successfully opened (except for performance of SCRATCH or RENAME functions on the data set). When you originate the password record, the value in the counter may be set at zero (X'0000') or any starting value your installation desires.

The protection mode indicator is set to indicate that the data set is to be read-only, or that it may be read or written. You set the indicator as follows:

- To zero (X'00') if the data set is to be read-only.
- To one (X'01') if the data set may be read or written.

You may use the 77-byte logging field to record any information about the data set under protection that your installation may desire, e.g., date of counter reset, previous password used with this data set, etc.

PROTECTING THE PASSWORD DATA SET

You 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.

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. You may 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 opening 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. Operating procedures at your installation must ensure that password records for all data sets currently under protection are entered in the PASSWORD data set.

PROTECTION FEATURE OPERATING CHARACTERISTICS

This section provides 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.

2. A password record does not exist in the PASSWORD data set for the protected data set being opened.

3. The protection mode indicator setting 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 setting.

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 section.

Volume Switching

The operating system end-of-volume routine does not request a password for a data set involved in a volume switch. Continuity of protection is handled in the following ways:

Input Data Sets - Tape and Direct-Access Devices

Processing continues if there is an equal comparison between the data set name in the tape label or DSCB on the volume switched to, and the name of the data set opened with the password. An unequal comparison terminates processing.

Output Data Sets - Tape Devices

The protection indicator in the tape label on the volume switched to is tested:

1. If the protection indicator is set ON, an equal comparison between the data set name in the label and the name of the data set opened with the password allows processing to continue. An unequal comparison results in a call for another volume.

2. If the protection indicator is OFF, processing continues, and a new label is written with the protection indicator set ON.

3. If only a volume label exists on the volume switched to, processing continues, and a new label is written with the protection indicator set on.

Output Data Sets - Direct-Access Devices

For existing data sets, an equal comparison between the data set name in a DSCB on the volume switched to, and the name of the data set opened with the password allows processing to continue. For new output data sets, the mechanism used to effect volume switching ensures continuity of protection and the DSCB created on the new volume will indicate protection.

Data Set Concatention

A password is requested for every protected data set that is involved in a concatentation of data sets, regardless of whether the other data sets involved are protected or not.

SCRATCH and RENAME Functions

An attempt to perform the SCRATCH or RENAME functions on a protected data set results in a request for the password. The protection feature issues an operator's message when a protected data set is the object of these functions. The Operator's Guide publication discusses the message.

Counter Maintenance

The operating system does not maintain the counter in the password record and no overflow indication will be given (overflow after 67,535 openings). You must provide a counter maintenance routine to check and, if necessary, reset this counter.

THE RESIDENT BLDL TABLE AND RESIDENT ACCESS METHOD OPTIONS

This chapter discusses the resident BLDL table and resident access method options and provides guidelines for their use.

Prerequisite Publications

The IBM System/360 Operating System: Data Management publication (Form C28-6537) contains a general discussion of the BLDL routine.

The IBM System/360 Operating System: System Generation publication (Form C28-6554) describes how you specify the resident BLDL table and resident access method option in the SUPRVSOR macro-instruction at system generation time.

The IBM System/360: Operating System: Utilities publication (Form C28-6586) contains a description of the IEBUPDAT utility which you use to construct lists of load module names in the procedure library (SYS1.PROCLIB).

The IBM System/360 Operating System: Storage Estimates publication (Form C28-6551) provides storage requirement information for the resident access method option and resident BLDL table option.

THE RESIDENT BLDL TABLE AND RESIDENT ACCESS METHOD OPTIONS

These options, when included in an Operating System/360 configuration, provide you with the capabilities of placing in the system nucleus:

1. All, or any selection of linkage library directory entries (40 bytes per entry).

2. A selected group of access method routines.

Placement occurs during the initial program load (IPL) process. You include either or both of these options when the system is generated. Parameters for specification of these options are provided in the system generation SUPRVSOR macro-instruction. Operator communication with these options may also be specified.

System issued ATTACH, LINK, LOAD, or XCTL macro-instructions requesting load modules in partitioned data sets require direct-access storage device accesses to search the data set directory for the location of the requested module (the BLDL table operation) and to fetch the module. The resident BLDL table option reduces the number of accesses required during execution of these macro-instructions when a load module (whose directory entry is resident) is requested from the linkage library. The resident access method option eliminates such accesses during execution of a system issued LOAD macro-instruction that requests any of the group of resident access method routines.

You specify the linkage library directory entries and the access method routines to be made resident through lists of linkage library or access method load module names placed in the procedure library (SYS1.PROCLIB). A standard list and alternative lists may exist for each option. IBM provides a standard list for the resident access method option. The standard lists are processed without operator intervention when the operator communication facility is not included with the options. When the operator communication facility is included, the operator must indicate the action to be taken. Selection of alternative lists may not be made unless the operator communication facility is included. The Operator's Guide publication describes the messages and replies associated with the two options.

The balance of this chapter discusses the function of each option, the creation of the procedure library lists, the use of the operator communication facility, and, in Appendix A, lists the content of the resident access method standard list.

THE RESIDENT BLDL TABLE OPTION

This option builds, in the system nucleus, a list of linkage library directory entries for use by ATTACH, LINK, LOAD, or XCTL macro-instructions requesting linkage library load modules. During execution of the BLDL operation in the macro-instruction routines, the linkage library directory is searched only when the directory entry for the requested load module is not present in the resident BLDL table.

You list, in a member of SYS1.PROCLIB, the load module name of linkage library load modules whose directory entries are to be made resident. The member name for the standard list is IEABLD00. Creation of procedure library lists is discussed later in this chapter. The next section provides guidelines for choosing the content of the list.

Note: Directory entries in the resident table are not updated as a result of updating the load module in the linkage library. The old

version of the load module is used until an IPL operation takes place and the new directory entry for the module is made resident.

SELECTING ENTRIES FOR THE RESIDENT BLDL TABLE

Any load module in the linkage library may have its directory entry placed in the resident BLDL table. Other items you should consider are:

1. Table size (each entry requires 40 bytes of storage).
2. Frequency of use of the load module.

Table Size

The resident BLDL table is incorporated in the system nucleus. The additional storage required is governed by the number of table entries and is acquired by reducing the amount of dynamic storage area available, i.e., the system nucleus expands. Each installation using the resident BLDL table option must determine the amount of storage it can afford for the resident BLDL table.

Frequency of Use

Short of placing the entire linkage library directory in the resident BLDL table, you make the option effective by selecting directory entries representing the load modules which are called most frequently. Your choice will depend on the system configuration and the operating practices of your installation. You should give load modules of the scheduling components of the system, linkage editor, and language processor(s) thorough consideration.

THE RESIDENT ACCESS METHOD OPTION

This option places access method load modules in the system nucleus and creates a resident list of the loaded modules. A LOAD macro-instruction requesting any access method module first scans the resident list. If the module is listed, no fetch operation is required.

You list, in a member of SYS1.PROCLIB, the load module names of access method load modules to be made resident. The member name for the standard list is IEAIGG00. A standard list of most frequently used access method modules is supplied by IBM, and is in SYS1.PROCLIB of the starter system under the standard member name.

The creation of procedure library lists and the content of the IBM supplied standard list is discussed later in this chapter. The next section discusses some considerations pertaining to the use of the access method option.

CONSIDERATIONS FOR USE

The storage space required for each access method module consists of the byte requirements of the module and its associated load request block (LRB). The Storage Estimates publication provides storage requirements for the resident access method option when used with the standard procedure library list provided by IBM.

All access method modules placed in the system nucleus are "only loadable". ATTACH, LINK, and XCTL macro-instructions cannot refer to the resident modules.

You may alter the standard access method list (or create alternative lists) to include access method modules supporting program controlled interrupt scheduling (PCI), exchange buffering, track overflow, and the UPDAT function of the OPEN macro-instruction.

To be eligible for use with the resident access method option, access method load modules must be reenterable. The module name must be of the form IGG019xx, where xx can be any two alphanumeric characters.

CREATING PROCEDURE LIBRARY LISTS

You use the IEBUPDAT utility program to construct the required lists of load module names in the procedure library. Standard member names for these lists are:

IEABLD00 for the BLDL table option
IEAIGG00 for the access method option

These are the member names that the nucleus initialization program will recognize at IPL time in the absence of any other specification, i.e., the operator communication facility was not incorporated.

Your input format (to IEBUPDAT) for the lists is the same for either option, consisting of library identification followed by the load module names. You use eighty character records with the initial or only record containing the library identification. Continuation is indicated by placing a comma after the last name in a record and a non-blank character in column 72. Subsequent records must start in column 16. The initial record format (with continuation) is:

```
1                                                    72
        SYS1.LINKLIB
[b...]  SYS1.SVCLIB    b...name1,name2,name3,...X
```

Subsequent records do not contain the library name.

SYS1.LINKLIB indicates that linkage library load module names follow.

SYS1.SVCLIB indicates that SVC library module names follow. You list linkage-library load module names in the same order that they appear in the linkage library directory.

You may construct alternative lists for either option and place them in the procedure library. Member names for these alternate lists are of the form:

IEABLDxx for the BLDL option
IEAIGGxx for the resident access method option
where xx can be any two alphanumeric characters.

Use of the alternative lists is indicated by the operator at IPL time and requires that the communication facility be present. When the operator communication facility is present, the operator must indicate for either or both options that the standard list is to be used; that alternative lists are to be used; or that, for this IPL, the option(s) will not be used. In the latter case, no resident BLDL table or access method routines will be placed in the nucleus.

EXAMPLE

The following coding illustrates the format and content of a BLDL option list that might be used to support the resident BLDL table option. The operator, at IPL time, would have to indicate the member

name, IEABLDAE, to the system. The load module names listed are from the Assembler (E), Linkage Editor, and scheduler components of the operating system.

```
//BLDLIST EXEC PGM=IEBUPDAT,PARM=NEW
//SYSPRINT DD SYSOUT=A
//SYSUT2 DD DSNAME=SYS1.PROCLIB,DISP=OLD
//SYSIN DD *
./       ADD   IEABLDAE,00,0,1
./       NUMBR 00000000,00000000,00000000,00000010
 SYS1.LINKLIB GO,IEEGESTO,IEEGK1GM,IEEICIPE,IEEIC2NQ,IEEIC3JF,          X
               IEEQOT00,IEFINTQS,IEFK1,IEFSD008,IEFW21SD,IEFXA,         X
               IETASM,IETDI,IETE1,IETE2,IETE2A,IETE3,IETE3A,IETE4M,     X
               IETE4P,IETE4S,IETE5,IETE5A,IETE5E,IETE5P,IETINP,IETMAC,  X
               IETPP,IETRTA,IETRTB,IET07,IET071,IET08,IET09,IET09I,     X
               IET10,IET10B,IET21A,IET21B,IET21C,IET21D,IEWL,IEWSZOVR
./       ENDUP
/*
```

Note: The operator reply "L" may be used in conjunction with a list specification and causes the content of the list to be printed. You should use this feature initially (especially with extensive lists) so that format errors, e.g., a 9 character name, and incorrect name specifications may be easily identified.

APPENDIX A: RESIDENT ACCESS METHOD OPTION - STANDARD LIST IEAIGG00

The content of the IBM supplied standard list for the resident access method option is shown below. The modules are listed in an ascending sequence by frequency of use, i.e., the least frequently used module is first in the list. This arrangement ensures efficient scanning of the resident list developed in storage.

Module Name	Access Method	Function
IGG019AV	QSAM (SB)	PUT Locate for Dummy Data Set
IGG019AN	QSAM (SB)	Backward Move - Format F, FB, U Records
IGG019AM	QSAM (SB)	Backward Locate - Format F, FB, U Records
IGG019AH	QSAM (SB)	GET Move with CNTL - Format V Records (Card Reader)
IGG019BE	BSAM	Magnetic Tape Forward Space or Backspace
IGG019AG	QSAM (SB)	GET Move with CNTL - Format V Records (Card Reader)
IGG019CB	SAM	Space or Skip Printer
IGG019CA	SAM	Stacker Select (Card Reader)
IGG019AK	QSAM (SB)	PUT Move, Format F, FB, U Records
IGG019AJ	QSAM (SB)	PUT Locate, Format V, VB Records
IGG019AI	QSAM (SB)	PUT Locate, Format F, FB, U Records
IGG019AC	QSAM (SB)	GET Move, Format F, FB, U Records
IGG019AB	QSAM (SB)	GET Locate, Format V, VB Records
IGG019AA	QSAM (SB)	GET Locate, Format F, FB, U Records
IGG019AR	QSAM (SB)	PUT Synchronization Routine
IGG019AQ	QSAM (SB)	GET Synchronization Routine
IGG019AL	QSAM (SB)	PUT Move, Format V, VB Records
IGG019AD	QSAM (SB)	GET Move, Format V, VB Records
IGG019BD	BSAM	NOTE/POINT Tape
IGG019BC	BSAM	NOTE/POINT Disk
IGG019BB	BSAM	CHECK (all devices)
IGG019BA	BSAM	READ/WRITE (all devices)
IGG019CK	SAM	SYSIN Delimiter Check (Appendage)
IGG019CJ	SAM	Read Length Check, Format V Records (Appendage)
IGG019CI	SAM	Length Check, Format FB Records (Appendage)
IGG019CH	SAM	End-of-Extent Check (Data Extent Block) (Appendage)
IGG019CL	SAM	Printer Test Channels 9,12 (Appendage)
IGG019CF	SAM	ASA Character to Command Code (Printer-Punch)
IGG019CE	SAM	End-of-Block (Printer-Punch)
IGG019CD	SAM	Schedules I/O for Direct-Access Output
IGG019CC	SAM	Schedules I/O for Tape, Direct-Access Input, Card Reader, Paper Tape Reader

SB=simple buffering
SAM=common sequential access method routines

CONSTRUCTING A DUMMY WAITR ROUTINE

This chapter discusses the preparation of a dummy WAITR routine for use with Option 2 (multiprogramming with a fixed number of tasks) of Operating System/360.

Recommended Publications

The IBM System/360 Operating System: Control Program Services publication (Form C28-6541) describes the WAITR macro-instruction.

The IBM System/360 Operating System: Assembler Language publication (Form C28-6514) describes the assembler language used to code the dummy WAITR routine.

CONSTRUCTING A DUMMY WAITR ROUTINE

The multiprogramming with a fixed number of tasks option (MFT) of Operating System/360 requires programs scheduled into any higher partitions to release the scheduler as soon after initiation as possible. The mechanism for release is the WAITR macro-instruction which causes the required scheduler shift to the next lower partition. You may desire to run programs not originally designed for the MFT environment, i.e., not containing the WAITR macro-instruction, in one of the higher partitions. In this circumstance you must provide a routine that will cause the required scheduler shift; invoke the program you desire to execute; and pass parameters to the invoked program. Your routine is executed as the first portion of any job with which it is associated.

The balance of this chapter discusses the functions of a dummy WAITR routine, provides a coding example, and discusses the job control language statements and programming considerations pertinent to use of a dummy WAITR routine.

FUNCTIONS OF THE DUMMY WAITR ROUTINE

When coding a dummy WAITR routine, your code must:

1. Issue a WAITR macro-instruction.
2. Dynamically invoke a specified program.
3. Restore the PARM= field of the EXEC statement that initiated execution of the WAITR routine, deleting only the name of the program to be invoked.

You use the WAITR instruction to initiate the desired scheduler shift. You dynamically invoke the program to be executed, i.e., transfer control via the XCTL macro-instruction, since once the WAITR macro-instruction is issued, the scheduler is released. Your WAITR routine identifies the program to be invoked by picking up its system name from the PARM= field of the EXEC statement that initiated execution of your WAITR routine.

Your WAITR routine must restore the PARM= field so that any parameter(s) present (other than the invoked program's name) may be picked up by the invoked program.

The next section, "A Coding Example" illustrates basic implementation of these functions.

A CODING EXAMPLE

The following source statement sequence illustrates the implementation of the dummy WAITR routine functions described in the preceding section. The statements are keyed to explanatory text by the circled numbers.

```
DUMWAIT  CSECT
         SAVE   (14,12)
         BALR   2,0
         USING  *,2
         ST     13,MYSAVE+4
         LA     13,MYSAVE
         L      3,0(1)        ADDRESS OF PARM AREA TO GR3
         LH     5,0(3)        PARM AREA COUNT FIELD TO GR5
```

```
[1]          WAITR  1,ECB=XECB    RELEASE SCHEDULER
             LA     3,0(3)
             LR     6,3
   SCAN      CLI    2(6),C','     SCAN FOR COMMA IN PARM FIELD
             BE     HIT           BRANCH IF FOUND
             LA     6,1(6)        POINT TO NEXT CHARACTER
             BCT    5,SCAN
             LA     5,1(5)
   HIT       SR     6,3           GR6 NOW CONTAINS NO. OF BYTES SCANNED
             BCTR   5,0           SUBTRACT 1 FROM COUNT FOR COMMA
             LA     7,3(3,6)      GR7 NOW POINTS AT REMAINING PARAMETERS
             BCTR   6,0           SETUP GR6 FOR USE IN EX INST.
[2]          EX     6,MODMOVE
             STH    5,0(3)        REMAINING COUNT TO PARM COUNT FIELD
             LTR    5,5           CHECK COUNT FOR ZERO
             BZ     XOUT          IF ZERO, SKIP PARMOVE
             BCTR   5,0           SETUP GR5 FOR USE IN EX INST.
[3]          EX     5,PARMOVE
   XOUT      L      13,MYSAVE+4
             L      14,12(13)
             L      1,24(13)
             XCTL   (2,12),EPLOC=MODNAME
   PARMOVE   MVC    2(1,3),0(7)
   MODMOVE   MVC    MODNAME(1),2(3)
   MYSAVE    DS     18F
             DS     0D
[4] MODNAME  DC     CL8' '
[5] XECB     DC     X'40000000'
             END
```

[1] The dummy WAITR macro-instruction must be coded as shown here. The event control block must be specified as shown in statement [5], i.e., the complete bit is ON.

[2] The subject instruction of this EX instruction places the name of the program to be invoked in the MODNAME field -- statement [4].

[3] The subject instruction of this EX instruction effectively deletes the invoked program name from the PARM= area by moving the remaining parameters in the area to the high order end of the area.

[4] This field must contain blanks.

[5] Coding the event control block as shown here sets the "complete bit" ON. The wait routine will then allow execution of the WAITR routine and the invoked program.

To the basic implementation shown in this example, you may wish to add diagnostic code to inform the operator that the PARM= field has been omitted from the EXEC statement. A count value of zero in the PARM= area count field indicates that no information has been placed in the PARM= area.

JOB CONTROL LANGUAGE STATEMENTS

You use the EXEC statement to initiate execution of the dummy WAITR routine; to specify the name of the program to be invoked; and to specify any parameters to be passed to the invoked program. A JOB statement and any DD statements defining data sets used by the invoked program must also be present in the input stream. A sample EXEC statement follows. The dummy WAITR routine has been cataloged as DUMWAIT.

```
//MFTJOB   JOB  (any valid parameters)
//         EXEC PGM=DUMWAIT,PARM='PROGX,COMPUTE,BINARY'

     (required DD statements for PROGX)
```

The name of the program to be invoked by DUMWAIT must be the first entry in the PARM= parameter list.

PROGRAMMING CONSIDERATIONS

A dummy WAITR routine itself does not require any special considerations for use with MFT. MFT conventions that apply to the invoked programs must be observed.

Note: Use of the dummy WAITR routine precludes the entering of input data via the job stream.

SYSTEM MACRO-INSTRUCTIONS

This chapter contains the description and formats of 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 contained in the prerequisite publications listed below.

Prerequisite Publications

The IBM System/360 Operating System: Control Program Services publication (Form C28-6541) contains the notation conventions used to describe the macro-instructions in this chapter.

The IBM System/360 Operating System: Assembler Language publication (Form C28-6514) contains the information necessary to code programs in the assembler language.

The IBM System/360 Operating System: System Control Block publication (Form C28-6628) contains format and field descriptions of the system control blocks referred to in this chapter.

LOCATE DEVICE CHARACTERISTICS (DEVTYPE) MACRO-INSTRUCTION

The DEVTYPE macro-instruction is used to request information relating to the characteristics of an I/O device, and to cause this information to be placed into a specified area.

Name	Operation	Operand
[symbol]	DEVTYPE	ddloc-addrx,area-addrx[,DEVTAB]

ddloc-addrx
: specifies the address of a double word that contains the symbolic name of the DD statement to which the device is assigned. The name must be left justified in the double word, and must be followed by blanks if the name is less than eight characters. The double word need not be on a double-word boundary.

area-addrx
: specifies the address of an area into which the device information is to be placed. The area can be either two full words or five full words, depending on whether or not the DEVTAB operand is specified. The area must be on a full word boundary.

DEVTAB
: If DEVTAB is specified, and the device is a direct-access device, five full words of information are placed into your area. If DEVTAB is specified, and the device is not a direct-access device, two full words of information are placed into your area. If DEVTAB is not specified, two full words of information are placed into your area.

  Note: Any reference to a dummy DD statement in the DEVTYPE macro-instruction will cause invalid information to be placed in the output area.

Device Characteristics Information

The following information is placed into your area:

Word 1 — Device Code from the UCB in which:

- Byte 1
  - bit 0 Unassigned
  - bit 1 Overrunable Device — 1 = yes
  - bit 2 Burst/Byte Mode — 1 = burst
  - bit 3 Data Chaining — 1 = yes
  - bit 4-7 Model Code
- Byte 2 — Optional Features
- Byte 3 — Device Classes
- Byte 4 — Unit Type

Note: Bit settings for Byte 2 -- Optional Features are noted in the UCB format and field description in the System Control Blocks publication.

Word 2 — Maximum block size. For direct-access devices, this value is the maximum size of an unkeyed block; for magnetic or paper tape, this value is the maximum block size allowed by the operating system. For all other devices, this value is the maximum block size accepted by the device.

If DEVTAB is specified, the next three full words contain the following information:

Word 3	Bytes 1-2	The number of physical cylinders on the device.
	Bytes 3-4	The number of tracks per cylinder.
Word 4	Bytes 1-2	Maximum track length.
	Byte 3	Block Overhead - the number of bytes required for gaps and check bits for each keyed block other than the last block on a track.
	Byte 4	Block Overhead - the number of bytes required for gaps and check bits for a keyed block that is the last block on a track.
Word 5	Byte 1	Block Overhead - the number of bytes to be subtracted if a block is not keyed.
	Byte 2	bits 0-6 Reserved bit 7 If 1, a tolerance factor must be applied to all blocks except the last block on the track.
	Bytes 3-4	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 2 of word 5. 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.

Output for Each Device Type

	UCB Type Field (Word 1 In Hexadecimal)	Maximum Record Size (Word 2 In Decimal)	DEVTAB (Words 3, 4, and 5 In Hexadecimal)
2540 Reader	10 00 08 01	80	Not Applicable
2540 Reader W/CI	10 01 08 01	80	Not Applicable
2540 Punch	10 00 08 02	80	Not Applicable
2540 Punch W/CI	10 01 08 02	80	Not Applicable
1442 Reader-Punch	50 00 08 03	80	Not Applicable
1442 Reader-Punch W/CI	50 01 08 03	80	Not Applicable
1442 Serial Punch	51 80 08 03	80	Not Applicable
1442 Serial Punch W/CI	51 01 08 03	80	Not Applicable
2501 Reader	50 00 08 04	80	Not Applicable
2501 Reader W/CI	50 01 08 04	80	Not Applicable
2520 Reader Punch	50 00 08 05	80	Not Applicable
2520 Reader Punch W/CI	50 01 08 05	80	Not Applicable
2520 B2-B3	11 00 08 05	80	Not Applicable
2520 B2-B3 W/CI	11 01 08 05	80	Not Applicable
1403	10 00 08 08	120*	Not Applicable
1403 W/UCS	10 80 08 08	120*	Not Applicable
1404	10 00 08 08	120*	Not Applicable
1443	10 00 08 0A	120*	Not Applicable
2671	10 00 08 10	32767	Not Applicable
1052	10 00 08 20	130	Not Applicable
2150	10 00 08 21	130	Not Applicable
2400 (9-track)	30 00 80 01	32767	Not Applicable
2400 (9-track phase encoding)	34 00 80 01	32767	Not Applicable
2400 (9-track dual-density)	34 20 80 01	32767	Not Applicable
2400 (7-track)	30 80 80 01	32767	Not Applicable
2400 (7-track and data conversion)	30 C0 80 01	32767	Not Applicable

2301	30 40 20 02	20483	000100C85003BA3535000200
2302	30 00 20 04	4984	00FA002E1378511414010219
2303	30 00 20 03	4892	0050000A131C922626000200
2311	30 00 20 01	3625	00CB000A0E29511414010219
2314	30 C0 20 08	7294	00C800141C7E922D2D010216

CI=Card Image Feature

UCS=Universal Character Set

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

Exceptional Returns

The following return codes are placed in register 15:

00 - request completed satisfactorily.

04 - ddname not found.

HOW TO READ 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).

Usually, the programmer is not concerned with the JFCB itself. In special applications, however, you may find it necessary to modify the contents of a JFCB before issuing an OPEN macro-instruction. To assist you, the system provides the RDJFCB macro-instruction. This macro-instruction causes a specified JFCB to be read into main storage from the job queue in which it has been stored. Format and field description of the JFCB is contained in the System Control Block publication.

When subsequently issuing the OPEN macro-instruction, you must indicate, by specifying the TYPE=J option, that you have supplied a modified JFCB to be used during the initialization process.

The JFCB is returned to the job queue by the OPEN routine or the OPENJ routine, if any of the modifications in the following list occur. These modifications can occur only if the information is not originally in the JFCB.

- Expiration date field and creation date field merged into the JFCB from the DSCB.
- Secondary quantity field merged into the JFCB from the DSCB.
- DCB fields merged into the JFCB from the DSCB.
- DCB fields merged into the JFCB from the DCB.
- Volume serial number fields added to the JFCB.
- Data set sequence number field added to the JFCB.
- Number of volumes field added to the JFCB.

If you make these, or any other modifications, and you want the JFCB returned to the job queue, you must set the high-order bit of field JFCBMASK+4 to one. This field is in the JFCB. Setting the high-order bit of field JFCBMASK+4 to zero does not necessarily suppress the return of the JFCB to the job queue. If the OPEN or OPENJ routines have made any of the above modifications, the JFCB is returned to the job queue.

OPEN -- Prepare the Data Control Block for Processing (S)

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 the publication IBM System/360 Operating System: Control Program Services. 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.

Name	Operation	Operand
[symbol]	OPEN	({dcb-addr,[(opt_1-code[,opt_2-code])],}...) [,TYPE=J]

TYPE=J

specifies that, for each data control block referred to, the programmer has supplied a job file control block (JFCB) to be used during initialization. A JFCB is an internal representation of information in a DD control 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 a job queue on direct-access storage. Its position, in relation to other JFCBs created for the same job step, is noted in a main storage table.

When this operand is specified, the user must also supply a DD control statement. However, the amount of information given in the DD statement is at the programmer's discretion, because he can ignore the system-created job file control block. (See the examples of the RDJFCB macro-instruction for a technique for modification of a system-created JFCB.)

Note: The DD statement must specify at least:

- Device allocation.
- A ddname corresponding to the associated data control block DCBDDNAM field.

RDJFCB -- Read a Job File Control Block (S)

The RDJFCB macro-instruction causes a job file control block (JFCB) to be read from the job queue into main storage for each data control block specified.

Name	Operation	Operand
[symbol]	RDJFCB	({dcb-addr,[(opt_1-code[,opt_2-code])],}...)

dcb,(opt_1,opt_2)

(same as dcb, opt_1, and opt_2 operands in OPEN macro-instruction)

Although the opt_1 and opt_2 operands are not meaningful during the execution of the RDJFCB macro-instruction, these operands can appear in the L-form of either the RDJFCB or OPEN macro-instruction to generate identical parameter lists, which can be referred to with the E-form of either macro-instruction.

Examples: The macro-instruction in 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; opt_2 for both blocks is assumed to be DISP. The macro-instruction in EX2 reads the system-created JFCBs for INVEN and MASTER from the job queue into main storage, thus making the JFCB's available to the problem program for modification. The macro-instruction in EX3 modifies the parameter list entry for the data control block named INVEN and indicates, through the TYPE=J operand, that the problem is supplying the JFCB's for system use.

```
EX1     RDJFCB (INVEN,,MASTER),MF=L
        .
        .
        .
EX2     RDJFCB MF=(E,EX1)
        .
        .
        .
EX3     OPEN   (,(RDBACK,LEAVE)),TYPE=J,MF=(E,EX1)
        .
        .
        .
```

Programming Notes: Any number of data control block addresses and associated options may be specified in the RDJFCB macro-instruction. This facility makes it possible to read 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 main 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 Appendix D of the IBM System/360 Operating System: Control Program Services publication. 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 (three 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 full word boundary.

The main 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.

Cautions: The following errors cause the results indicated:

Error	Result
A DD control statement has not been provided.	No action
A main storage address has not been provided.	Abnormal termination of task

L- and E-Form Use: The L and E forms of this macro-instruction are written as described in Appendix B of the IBM System/360 Operating System: Control Program Services publication.

Printed in U.S.A. C28-6550-2

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

READER'S COMMENTS

Title: IBM System/360 Operating System
System Programmer's Guide

Form: C28-6550-2

Is the material:	Yes	No
Easy to Read?	___	___
Well organized?	___	___
Complete?	___	___
Well illustrated?	___	___
Accurate?	___	___
Suitable for its intended audience?	___	___

How did you use this publication?
___ As an introduction to the subject ___ For additional knowledge
Other _____

fold

Please check the items that describe your position:
___ Customer personnel ___ Operator ___ Sales Representative
___ IBM personnel ___ Programmer ___ Systems Engineer
___ Manager ___ Customer Engineer ___ Trainee
___ Systems Analyst ___ Instructor Other_____

Please check specific criticism(s), give page number(s), and explain below:
___ Clarification on page(s)
___ Addition on page(s)
___ Deletion on page(s)
___ Error on page(s)

Explanation:

CUT ALONG LINE

fold

FOLD ON TWO LINES, STAPLE AND MAIL
No Postage Necessary if Mailed in U.S.A.

C28-6550-2

staple staple

fold fold

FIRST CLASS
PERMIT NO. 81

POUGHKEEPSIE, N.Y.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

IBM CORPORATION
P.O. BOX 390
POUGHKEEPSIE, N. Y. 12602

ATTN: PROGRAMMING SYSTEMS PUBLICATIONS
DEPT. D58

fold fold

Printed in U.S.A.

C28-6550-2

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

staple