File No. S360-20
Form C28-6550-5

OS

IBM Systems Reference Library

IBM System/360 Operating System

System Programmer's Guide

This publication consists of self-contained chapters, each of which provides information on how to modify, extend, or implement capabilities of the IBM System/360 Operating System control program. It is designed primarily for system programmers responsible for maintaining, updating, and extending the operating system features.

Topics:

Preface

This publication consists of self-contained chapters, each of which provides information on how to modify, extend, or implement 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. It is assumed that users of this publication are thoroughly familiar with the design of the operating system and its features. Each chapter contains its own introductory section and list of prerequisite publications. This organization has been used to reduce cross-referencing and to facilitate the addition of new chapters.

Sixth Edition (November, 1968)

This is a major revision of, and obsoletes, C28-6550-4. Descriptions of control volume pointer entries, the EXCP macro instruction, link pack area uses, the DEVTYPE macro instruction, standard list IEARSV00, and SYS1.ACCT specifications have been changed. Descriptions of checkpoint/restart, user-written I/O appendages, the blocked procedure library, automatic SYSIN batching, the STAE and PURGE macro instructions, and the Satellite Graphic Job Processor have been added. Changes to the text are indicated by a bar (|) to the left of the change; changed illustrations are indicated by the symbol • to the left of the caption.

This edition applies to release 17 of IBM System/360 Operating System, and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters. Changes are continually made to the specifications herein; before using this publication in connection with the operation of IBM systems, consult the latest SRL newsletter, Form N20-0360, for the editions that are current and applicable.

Requests for copies of IBM publications should be made to your IBM representative or to the branch office serving your locality.

A form is provided at the back of this publication for reader's comments. If the form has been removed, comments may be addressed to IBM Corporation, Programming Systems Publications, Department D58, PO Box 390, Poughkeepsie, N. Y. 12602

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: Supervisor and Data Management Services publication (Form C28-6646) 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: 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 255 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.

If a code of 4 is returned in register 15 indicating that the required control volume was not mounted, bytes 260-265 of the work area will contain the volume serial number of this required volume. If LOCATE finds an old CVOL pointer entry, and the CVOL is not mounted, binary zeros will be returned in bytes 253-256 of the work area. However, if a new CVOL pointer entry is found, the four-byte device code of the CVOL will be returned in those bytes.

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 from the beginning of the catalog, R is the record number of the desired block on the 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.

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          X'30002001'                       2311 DISK STORAGE
            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 10-byte area into which you have placed the 4-byte binary device code of the volume to be connected followed by the 6-byte area to contain the volume serial number of the volume to be connected.

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.

<u>Example</u>: 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.

aintain and Modify the Catalog and VTOC

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]
rs 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.
28	Relative track address (TTR) supplied to LOCATE is out of the SYSCTLG data set extents.

If the LOCATE macro instruction fails to perform 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.
16	Invalid workarea pointer
20	CCHH not within boundaries of VTOC extent (seek mode)

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.

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.

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.

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.

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.

A new control volume pointer entry can appear only in volume indexes. It is 22 bytes long and contains 5 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 fields 4 and 5.

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

Field 3: Halfword Count (1 byte) contains a binary 5 to indicate that five halfwords follow.

Field 4: Control Volume Device Code (4 bytes) contains the 4-byte binary device code of the control volume whose index contains an entry identifying the high level index name in field 1.

Field 5: Control Volume Serial Number (6 bytes) contains the serial number of the control volume whose 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
IBM 2321 Data Cell		30002005

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: Supervisor and Data Management Macro Instructions* publication (Form C28-6647) 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. If you wish to aid system facilities in recovering from machine malfunctions, your SVC routines should be refreshable.

- 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	SVC 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: • <u>Register 0</u> contains the address of the TCB of the task to be terminated. • <u>Register 1</u> 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. • <u>Register 14</u> 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.

Handling Accounting Information

You may add accounting facilities to PCP, MFT, and MVT configurations of the operating system. This chapter describes the input available to an accounting routine; the characteristics and requirements of an IBM-supplied data set writer that may be used to log accounting information generated by an accounting routine; and how to insert an accounting routine into the control program. Conventions to be followed in preparing an accounting routine are also noted.

REFERENCE PUBLICATIONS

The IBM System/360 Operating System: Operator's Guide publication (Form C28-6540) describes the procedure used to update system data sets (used when inserting your accounting routine into the control program in MFT and MVT configurations).

The IBM System/360 Operating System: Job Management program logic manuals, Forms Y28-6613 and Y28-6660[1] discuss the control program component in which your accounting routines are inserted.

---

[1]IBM documents with Y prefix form numbers are restricted in distribution and must be obtained with the approval of local IBM management.

Accounting Routines

Your installation may prepare accounting routines for insertion in PCP, MFT, or MVT configurations of the operating system. These routines are inserted in the control program during, or after, system generation. There are differences, between configurations, in the accounting routine attributes, the time(s) at which an accounting routine is entered, and the information and facilities available to an accounting routine. These differences are noted in the text.

Prerequisite Actions

At system generation you must specify that an accounting routine is to be supplied. This is done through the ACCTRTN=parameter of the system generation SCHEDULR macro instruction. The system generation specification must be made for PCP, MFT, and MVT configurations of the operating system.

This specification causes the linkage to your accounting routine to be installed in the scheduler component of the system being generated, and makes usable the accounting data set writer routine. If you are not going to install your accounting routine until after the system is generated, a dummy accounting routine (named IEFACTRT) is also placed in the system at this time. Insertion of accounting routines in the control program is discussed later in this chapter.

Accounting Routines Conventions

FORMAT

Your accounting routine may consist of one or more control sections.

ATTRIBUTES

An accounting routine written for insertion in PCP or MFT configurations of the operating system must be serially reusable.

An accounting routine written for insertion in an MVT configuration of the operating system must be reenterable.

CSECT NAME AND ENTRY POINT

The control section containing the entry point of your accounting routine, and the entry point, must be named IEFACTRT.

REGISTER SAVING AND RESTORING

The content of registers 0 through 14 must be saved upon entry to your accounting routine and restored prior to exiting.

ENTRANCES

Control is given to your accounting routine at the following times:

PCP, MFT, MVT Configurations
Step initiation
Step termination
Job termination

EXIT

You can use the RETURN macro instruction to restore the contents of the general registers and return control to the operating system.

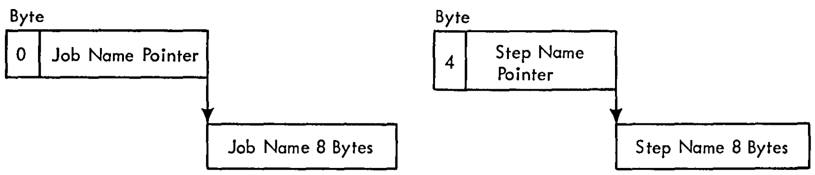
Input Available to Accounting Routines

The information available to an accounting routine varies slightly between PCP, MFT and MVT configurations of the operating system. These differences are noted in the following diagram.

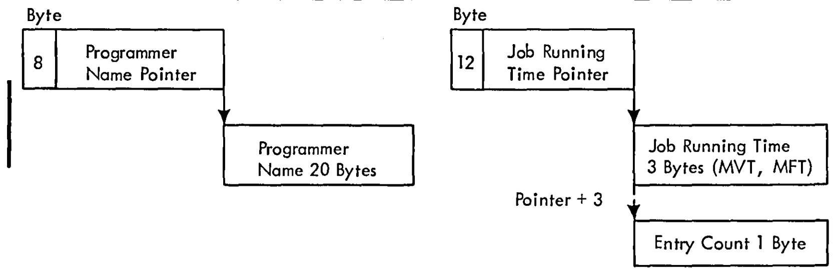
Register 0 contains an entrance code, indicating at what time the accounting routine is being given control.

```
          Register 0 =  8:  Step initiation
                     = 12:  Step termination
                     = 16:  Job termination
```

Register 1 contains the starting address of a list of pointers to items of accounting information. Each pointer is on a fullword boundary. The sequence of pointers in the list and the items of information provided are described in the following diagram.



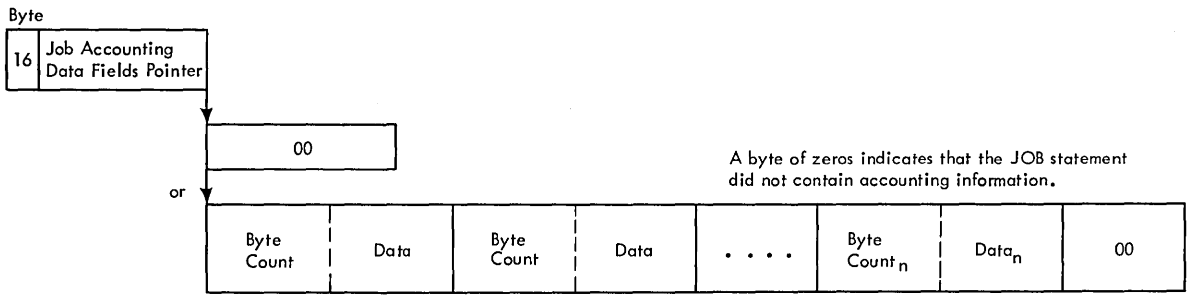
The step name pointer is zero at job termination.


Byte
8 Programmer Name Pointer
Programmer Name 20 Bytes
Byte
12 Job Running Time Pointer
Job Running Time 3 Bytes (MVT, MFT)
Pointer + 3
Entry Count 1 Byte

A right justified binary number represents job running time in hundredths (0.01) of a second.

If a programmer deferred restart occurs, the time used during the original execution is omitted from the job time passed to a user routine.

The entry count byte contains the number of job accounting entries picked up from the JOB statement. Commas used to denote omitted entries are counted.

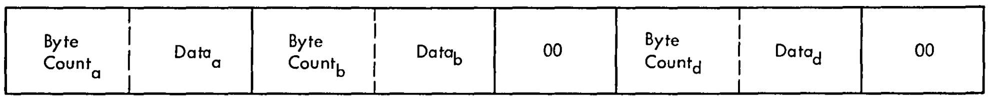


A byte of zeros indicates that the JOB statement did not contain accounting information.

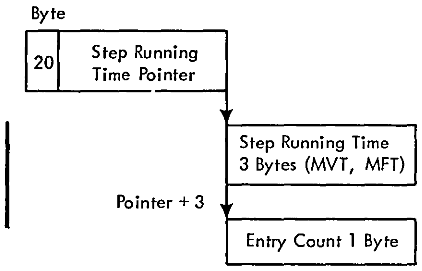
These data fields contain the accounting information that was specified in the JOB statement. The first byte of each field contains the number of bytes of data that follow. The last data field is followed by a byte of zeros.

A data field — consisting only of the first, or count byte, is developed for an omitted accounting entry. The byte contains zeros, indicating that no data is present for that field. In this case:

When (a, b,, d) appears in the JOB statement



Note: Use the entry-count byte (job running time pointer + 3) to determine if you have processed all the accounting data fields.

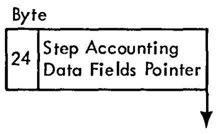

Byte
20 Step Running Time Pointer
Step Running Time 3 Bytes (MVT, MFT)
Pointer + 3
Entry Count 1 Byte

The step running time pointer is zero at job termination.

The step running time is not on a full word boundary. A binary numer, right justified, represents step running time in hundredths (0.01) of a second.

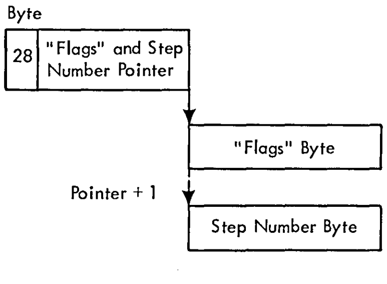
If an automatic restart occurs, the system gives control to a user routine prior to restarting; step time passed is the time used by the step. Upon successful completion of a step that was automatically restarted, the step time passed to a user routine does not include the time used by the step during its original execution. If a programmer deferred restart occurs, the time used during the original execution is not included in the step time passed to a user routine.

Number of step accounting entries picked up from the EXEC statement. Commas used to denote omitted entries are counted.



This pointer is zero at job termination

The step accounting data fields conform to the same specifications as the job accounting data fields.



Setting bit 7 of this byte to 1 effects job cancellation.

This byte contains the number of the job step currently being processed. The first step in the job is 1.

Note: You can use the flag byte to cancel the execution of a job whose accounting information does not conform to your installation's standards. You can equate step initiation for the first step in a job to job initiation, i.e., the step number byte contains 1.

Output From the Accounting Routine

You can write output in three ways: by issuing console messages; by using the standard system output; by using an IBM-supplied accounting data set writer.

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.

   When writing an accounting routine for inclusion in the job scheduler, you must be aware that register saving conventions within the control program are different from those for problem programs. In the job scheduler, registers are saved in the sequence 0-14 in a 15-word save area. There is no place provided to save register 13. You must provide some other means of saving register 13; you may either save it in another register or provide additional save area that is not known to the control program. This can be done by adding a word to the end of the save area that is provided and is addressed as SAVE + 60.

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)

3. Accounting Data Set Writer -- This writer places accounting records you have constructed in your accounting routine in a data set named SYS1.ACCT. The data set must reside on a permanently resident direct access device. You must provide, in your accounting routine, linkage to the writer, and pass the beginning address of the record to be written, to it.

   Appendix A of this chapter discusses the use of the data set writer.

Sample Accounting Routine

A sample accounting routine, showing use of the data set writer, output to system output, and issuance of console messages, is stored under the member name SAMACTRT in the SYS1.SAMPLIB data set furnished with the starter operating system.

Inserting an Accounting Routine Into the Control Program

Your accounting routine can be inserted in the control program in two ways; by placing the routine on the SYS1.CI505 data set used in system generation or by placing the routine in the appropriate load module of the control program after system generation. The effect of either action is to replace a dummy accounting routine with your accounting routine.

Insertion at System Generation

To insert your accounting routine into the control program during system generation, you must, prior to the start of the system generation process, place your routine in the SYS1.CI505 data set, using the linkage editor. The SYS1.CI505 data set (furnished with the starter operating system) contains load modules which are combined during the system generation process to form the load modules composing the control program. In response to the specification made in the system generation SCHEDULR macro instruction, your accounting routine is incorporated in the appropriate load modules for the system being generated.

You must place your accounting routine in the SYS1.CI505 data set under the name IEFACTRT. You will be replacing the dummy accounting routine -- also named IEFACTRT.

Insertion After System Generation

To insert your accounting routine into the control program after system generation you place the routine in load modules of the scheduler component of the generated control program, using the linkage editor. _The scheduler load modules are in the linkage library (SYS1.LINKLIB data set) of the generated system_. The affected load modules of the three PCP schedulers (18K, 44K, 100K), the MFT schedulers (30K, 44K), and the MVT scheduler are as follows:

PCP Configurations

18K Scheduler

load module IEFSELCT -- step initiation
load module IEFSTERM -- step termination
load module IEFJTRM1 -- job termination

44K Scheduler

load module IEFSTERM -- step initiation/termination
load module IEFJTERM -- job termination

100K Scheduler

load module GO -- step initiation/termination and job termination

MFT Configurations

30K Scheduler

load module IEFSD520 -- step initiation
load module IEFSD515 -- step/job termination

44K Scheduler

load module IEFW21SD -- step initiation
load module IEFSD510 -- step/job termination

MVT Configuration

MVT Scheduler

load module IEFSD061 -- step and job termination
load module IEFW21SD -- step initiation

An example of the input for a linkage editor run to insert your accounting routine into any of the job schedulers follows:

```
//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     *
               .                                   )
               .                                   |  This sequence must be
               .                                   |  repeated for each
          (object code)                            |  scheduler load module
               .                                   |  into which you wish
               .                                   |  to insert accounting
               .                                   |  routines.
             INCLUDE SYSLMOD(load module name)     )
               ALIAS alias names
               ENTRY entry point name
               NAME  load module name(R)
```

In this example "load module name" represents the appropriate scheduler load module as identified in the preceding text. To ensure accuracy in identifying the correct alias names and entry point names for the load modules, obtain these names from the system generation listing produced during generation of the system you are working with. These names are specified in the system generation Stage II linkage editor output for the linkage editor execution that produced the load module.

Appendix A: Accounting Data Set Writer (PCP, MFT, MVT)

The accounting data set writer (module IEFWAD) is inserted in the appropriate scheduler load modules during system generation when accounting routine inclusion is specified in the SCHEDULR macro instruction. These are the same modules in which your accounting routine is inserted. Scheduler storage requirements are increased by the amount of storage needed by your accounting routine plus 2600 bytes. The writer places accounting records developed by your routine in a data set named SYS1.ACCT.

Linkage

Your accounting routine links to the writer via the following mechanism:

```
              L      R15,VCON
            BALR      14,15
              .
              .
              .
    VCON     DC      V(IEFWAD)
```

Input

Your accounting routine passes in register 1 the address of the accounting record to be written.

The record format is:

```
    DS  3H     -- space used by the data set writer
    DC  H'__'  -- contains the number of bytes of data being passed.
                  This number cannot exceed the capacity of 1 track on
                  the direct access volume being written on.
    DC  __     -- the data to be written in SYS1.ACCT.
```

Registers 13, 14, and 15 are used as specified by operating system conventions (14 and 15 are used for linkage, as above; 13 must point to an 18-word save area).

Specifying the SYS1.ACCT Data Set

The SYS1.ACCT data set must be pre-allocated on a direct access volume that will be permanently resident. The data set must be named SYS1.ACCT, have no secondary extents, and be allocated contiguous space. Do not catalog the data set.

If your installation has two permanently resident volumes available for accounting routine use, you may create two SYS1.ACCT data sets and utilize the console messages and replies or the SET command (PCP only) to notify the system as to which data set is to be written to.

Output

When your accounting routine receives control from the writer:

Register 0	The number of empty tracks in SYS1.ACCT	
Register 15	Condition codes	
	0	Normal exit
	4	Record to be written is longer than a track
	8	No space left in SYS1.ACCT
	12	SYS1.ACCT not found
	16	Permanent I/O error
	20	SYS1.ACCT end of file not found
	24	Unit name not found

Use of ENQ/DEQ

IEFWAD enqueues on the major Q name SYSIEFAR and the minor Q name WD.

Specifying the Device on Which SYS1.ACCT Resides

The parameter [,ACCT=([unitname][,N])] has been added as an option to the SET command (PCP only). In this parameter:

unit name
    specifies the device on which SYS1.ACCT resides; if this parameter is omitted the system residence volume is assumed.

N
    specifies that the lowest extent of SYS1.ACCT may be used; if this parameter is omitted writing will be attempted from the last record written.

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 chapter contains the following:

- The format of the macro instructions.
- The job control and utility statements needed to add the macro instructions to the library.
- The macro-definition to be added to the library.

The information previously contained in this chapter on label handling routines may be found in the publication IBM System/360 Operating System: Tape Labels, Form C28-6680.

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=IEBUPDTE,PARM=NEW
//SYSPRINT      DD      SYSOUT=A
//SYSUT2        DD      DSNAME=SYS1.MACLIB,DISP=OLD
//SYSIN         DD      DATA
./        ADD   NAME=IECDSECT,LIST=ALL
                .
                .
                .
                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
         DS    CL1           RESERVED FOR FUTURE USE
FL2BLKA  DS    CL1           BLOCK ATTRIBUTE
FL2RES   DS    CL41          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
MVOLL     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'00000000'
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
DXDEBUCB  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
DXDCBDEB  DC    A (DXDEB)
          SPACE 1
*                       CHANNEL CONTROL WORDS
          SPACE 1
          CNOP  0,8
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=IEBUPDTE,PARM=NEW
//SYSPRINT     DD      SYSOUT=A
//SYSUT2       DD      DSNAME=SYS1.MACLIB,DISP=OLD
//SYSIN        DD      DATA
./       ADD   NAME=IEFUCBOB,LIST=ALL
               .
               .
               .
               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
UCBUCSID  EQU  UCBSNS+2    UCS CHARACTER SET-ID
SRTEVOLI  DS   CL6         VOLUME SERIAL
UCBUCSOP  EQU  UCBSNS+6    UCS OPTIONS
UCBUCSO1  EQU  128         DEFAULT CHARACTER SET
UCBUCSO2  EQU  64          BUFFER LOADED in FOLD MODE
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 QUIESCED
*                                     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

*THE FOLLOWING DESCRIBES ONE OF THE 10 SUB-UCBS  FOR THE 2321--

          ORG  SRTEUSER
DATACELL  DS   0CL16       10 OF THESE ARE PRESENT FOR 2321
DCELBBNR  DS   XL2         BIN NUMBER
DCELSTAB  DS   X           STATUS B
DCELSTAT  DS   X           STATUS A
DCELVOLI  DS   CL6         VOLUME SERIAL NUMBER
DCELJBNR  DS   X           INTERNAL JOB NUMBER
DCELDMCT  DS   X           DATA MANAGEMENT COUNT
DCELVTOC  DS   XL3         TTR OF VTOC START
DCELUSER  DS   X           CURRENT NUMBER OF USERS
          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=IEBUPDTE,PARM=NEW
//SYSPRINT      DD       SYSOUT=A
//SYSUT2        DD       DSNAME=SYS1.MACLIB,DISP=OLD
//SYSIN         DD       DATA
./        ADD   NAME=IEFJFCBN,LIST=ALL
                .
                .
                .
                IEFJFCBN macro-definition
                .
                .
                .
./        ENDUP
/*
```

IEFJFCBN Macro-Definition

```
         MACRO
         IEFJFCBN
INFMJFCB EQU   *
JFCBDSNM DS    CL44         DATA SET NAME
JFCBELNM DS    CL8          ELEMENT NAME OR VERSION
JFCBTSDM 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
*                          PSECIFIED 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
```

The Must Complete Function

This chapter provides information concerning system routine use of the must complete function. This function is available to system routines operating in MFT and MVT environments as an extension of the ENQ/DEQ facilities.

Reference Publications

The IBM System/360 Operating System: Supervisor and Data Management Services publication (Form C28-6646) describes ENQ and DEQ macro instruction use except for applications of the must complete function.

The IBM System/360: Supervisor and Data Management Macro Instructions publication (Form C28-6647) describes the ENQ and DEQ macro instructions except for the SMC and RMC operands.

The Must Complete Function

System routines (routines operating under a storage protection key of zero) often engage in updating and/or manipulation of system resources (system data sets, control blocks, queues, etc.) that contain information critical to continued operation of the system. These routines must complete their operations on the resource. Otherwise, the resource may be left in an imcomplete state or contain erroneous information -- either condition leads to unpredictable results.

The must complete function is provided in the ENQ service routine to ensure that a routine queued on a critical resource(s) can complete processing of the resource(s) without interruptions leading to termination. The effect of the must complete function is to place other routines (tasks) in a wait state until the requesting task -- the task (routine) issuing a ENQ macro instruction with the set-must-complete (SMC) operand -- has completed its operations on the resource. The requesting task releases the resource and terminates the must complete condition through issuance of a DEQ macro instruction with the reset-must-complete (RMC) operand.

Realize that, for the time it is in effect, the must complete function serializes operations to some extent in your computing system. Therefore, its use should be minimized -- use the function only in a routine that processes system data whose validity must be ensured.

As an example, in multi task environments, the integrity of the volume table of contents (VTOC) must be preserved during an updating process so that all future users may have access to the latest, correct, version of the VTOC. Thus, in this case, you should enqueue on the VTOC and use the must complete function (to suspend processing of other tasks) when updating a VTOC.

Just as the ENQ function serializes use of a resource requested by many different tasks, the must complete function serializes execution of tasks.

Scope

The must complete function can be applied at two levels:

THE SYSTEM LEVEL: Only the requesting task, and system tasks included during system generation, are allowed to execute. All other tasks in the system are placed in a wait state.

THE STEP LEVEL: In a partition or region, only the requesting task is allowed to execute. All other tasks in the partition or region, including the initiator task, are placed in a wait state.

CAUTION: Use of the must complete function at the system level should not be attempted until all alternatives have been exhausted. Except for extremely unusual conditions the system level of must complete should never be used.

REQUESTING THE MUST COMPLETE FUNCTION

You request the must complete function by coding the set-must-complete (SMC) operand in an ENQ macro instruction. The format is:

```
name      ENQ        ...,SMC={SYSTEM}
                             {STEP  }
```

You may specify SYSTEM or STEP. The parameters SYSTEM and STEP indicate the level to which the must complete function is to apply. The other operands of ENQ are described in the <u>Supervisor and Data Management Macro Instructions</u> publication.

Because of the properties of the TEST and USE parameters of the RET operand of the ENQ macro instruction, the SMC operand should be used only if the RET operand is to use the parameters HAVE, or NONE (in the E-form of ENQ), or if the RET operand is not used at all.

You may request the must complete function only in routines operating under a protection key of zero. If the protect key is not zero, the task using the routine requesting "must complete" is abnormally ended.

OPERATING CHARACTERISTICS

When the must complete function is requested the requesting task is marked as being in the must complete mode and all asynchronous exits from the requesting task are deferred. Other tasks in the system (except the allowed tasks at the system level) or associated with the requesting task in a job step (step level) are placed in a wait state. Thus tasks external to the requesting task are prevented from initiating procedures that will cause termination of the requesting task. Other external events, such as a CANCEL command issued by an operator, or a job step timer expiration are also prevented from terminating the requesting task.

The must complete mode of operation is not entered until the resource(s) queued upon are available.

At the system or step level, the requesting task can cause its own abnormal termination. If the requesting task does come to an abnormal termination before a reset condition has been effected, the operating system is stopped at the point of error to permit investigation of the trouble. It is then necessary to restart the system with the initial-program-load (IPL) procedure.

PROGRAMMING NOTES

1. All data used by a routine that is to operate in the must complete mode should be checked for validity to ensure against a program-check interruption.

2. A routine that is already in the must complete mode should avoid calling another routine which also operates in the must complete mode. However, one level of nesting is permitted, when necessary, with the following cautions:

   a. A task may set the must complete mode for both the system and the step. If multiple settings are made for either the system or the step, only the first setting of each is effective -- the others are treated as no operation.

   b. The same is true for reset-must-complete. The first RMC for the system will reset the status of the system, the first RMC for the step will reset the status of the step, and all others will be treated as no operation.

3. Interlock conditions that can arise with the use of the ENQ function are discussed in the <u>Supervisor and Data Management Services</u> publication.

Additionally, an interlock may occur if your routine issues an ENQ macro instruction while in the must complete mode. The resource you want to queue on may already be queued on by a task placed in the wait state due to the must complete request you have made. Since the resource cannot be released, all tasks wait.

4. The macro instructions ATTACH, LINK, LOAD, and XCTL should not be used, <u>unless extreme care is taken</u>, by a routine operating in the must complete mode. An interlock condition will result if a serially-reusable routine requested by one of these macro instructions has been requested by one of the tasks made non-dispatchable by the use of the SMC operand or was requested by another task and has been only partially fetched.

   For example, suppose routine "b" in task B has requested and is using subroutine "c". Subsequently routine "a" in task A (of a higher priority than task B) receives control of the processing before routine "b" finishes with subroutine "c". If routine "a" issues an ENQ macro instruction with the SMC operand and puts task B (and, thus, routine "b") in a non-dispatchable condition, subroutine "c" remains assigned to routine "b". Now, if routine "a" issues a request (via a LINK, LOAD, etc. macro instruction) for subroutine "c", an interlock will occur between tasks A and B: task A cannot continue since subroutine "c" is still assigned to task B, and task B cannot continue (and thus release subroutine "c") because task A in the must complete mode has made task B nondispatchable.

5. The time your routine is in the must complete mode should be kept as short as possible -- enter at the last moment and leave as soon as possible. One suggested way is to:

   a. ENQ (on desired resource(s))
   b. ENQ (on same resource(s)),RET=HAVE,SMC= $\left\{\begin{array}{l}\text{SYSTEM}\\ \underline{\text{STEP}}\end{array}\right\}$

   Item a gets the resource(s) without putting the routine into the must complete mode.

   Later, when appropriate, issue the ENQ with the must complete request (Item b). Issue a DEQ macro instruction to terminate the must complete mode as soon as processing is finished.

TERMINATING THE MUST COMPLETE FUNCTION

You terminate the must complete function and release the resource queued upon by coding the reset-must-complete (RMC) operand in a DEQ macro instruction. The format is:

name DEQ ...,RMC= $\left\{\begin{array}{l}\text{SYSTEM}\\ \underline{\text{STEP}}\end{array}\right\}$

The parameter (SYSTEM or STEP) <u>must agree</u> with the parameter specified in the SMC operand of the corresponding ENQ macro instruction.

Tasks placed in the wait state by the corresponding ENQ macro instruction are made dispatchable and asynchronous exits from the requesting task are enabled.

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: Supervisor and Data Management Services* publication (Form C28-6646) 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: Supervisor and Data Management Macro Instructions* publication (Form C28-6647) 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.

When planning EXCP operations and appendages for use on central processing units with parallel processing, special precautions must be observed. An example of such central processing units is the IBM System/360 Model 91 which can execute instructions in a sequence other than the physical sequence in which they appear in a listing. Such a central processing unit maintains logical consistency in its own operations, including the beginning and ending of I/O operations. However, it is impossible for such a central processing unit to maintain consistency with operations performed by asynchronous units. This type of central processing unit recognizes a special "no operation" to force sequential operations in the environments where it might be required. The appropriate hardware manual should be carefully studied before coding EXCP and appendage routines for this type of central processing unit.

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.

----

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

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 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 or by link-editing it into the SYS1.SVCLIB. 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 when one of the following occurs:

- Start I/O is issued.
- Program controlled interruption.
- End of extent.
- Channel end.
- Abnormal end.

Appendages are executed in supervisor state. 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, SVC, or any privileged instruction). 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 queue element (RQE) for the channel program.

  The request queue element contains the following information:

  Bytes 1 and 2
    are the link field when the RQE is an I/O 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, if the priority option has been selected for the system.

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

The request queue element is normally 12 bytes in length; for a multitasking environment, it includes 4 more bytes that contain 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).
- Register 15 contains the address of the entry point to the appendage.

Register 14 contains 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; this is reserved in case an abnormal condition occurs while the appendage is in control. 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 following table summarizes register conventions.

Appendages	Entry Point	Returns		Available Work Reg.
EQE	Reg 15		Extent Error	Reg. 10, 11, 12 & 13
		Reg 14 + 0	Return	
		Reg 14 + 4	Skip	
		Reg 14 + 8	Try Again	
SIO	Reg 15	Reg 14 + 0	Normal	Reg. 10, 11 & 13
		Reg 14 + 4	Skip	
PCI	Reg 15	Reg 14 + 0	Normal	Reg. 10, 11, 12 & 13
CE	Reg 15	Reg 14 + 0	Normal	Reg. 10, 11, 12 & 13
		Reg 14 + 4	Skip	
		Reg 14 + 8	Re-EXCP	
		Reg 14 + 12	By-Pass	
XCE	Reg 15	Reg 14 + 0	Normal	Reg. 10, 11, 12 & 13
		Reg 14 + 4	Skip	
		Reg 14 + 8	Re-EXCP	
		Reg 14 + 12	By-Pass	

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, unless an error recovery procedure is in control. If SIO is not initiated because of a busy condition, the appendage will be reentered before SIO is reissued.

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 do the posting by taking the following steps:

1. Save necessary registers.
2. Place pointer to post entry address from the CVT in Reg 15.
3. Place current TCB address from the CVT in Reg 12.
4. Place ECB address from the IOB in Reg 11.
5. Set the completion code in the high order byte in Reg 10.
6. Go to Post using BALR 14, 15.

You may use registers 10, 11, and 13 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. This appendage may be reentered for the same channel program if the error recovery procedure is in the process of retrying a CCW with the program controlled bit set on. The IOBERR flag is set when the error recovery procedure is in control (IOBFL1 = X'20').

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 (X'42') 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 returned to the available queue.

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

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

Note: If an end-of-cylinder or file-protect condition occurs, the 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 you wish to try the request in the next extent, you must move the new seek address into the UCB at UCB+48.

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

Channel End Appendage

This appendage is entered when a channel end, unit exception with or without channel end, or channel end with wrong length record occurs 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, and the IOBEX flag (X'04') in IOBFL1 is set on. The condition code may be directly tested by using a BC instruction. A CC=0 means no UEX or WLR accompanied this interruption. The CSW status may be obtained from the IOBCSW.

If the appendage takes care of the wrong length record and/or unit exception it may turn off the IOBERR flag and return normally. The event will then be posted complete (completion code 7F under normal conditions, taken from the high-order byte of the IOBECB field). If the appendage returns normally without resetting the IOBEX flag to zero, the request will be routed to the associated device error routine, and then the abnormal end appendage will be immediately entered. This abnormal end appendage will be entered with IOBECB completion code = '41'.

You may use the following optional return addresses:

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

- Contents of register 14 plus 8 - The channel program is not posted complete, and its request element is placed back on the request queue so that the I/O operation can be retried. For correct re-execution of the channel program, you must re-initialize the "Flags 1", "Flags 2", and "Flags 3" fields of the input/output block and set the "Error Counts" field to zero. As an added precaution, the IOBSNS and IOBCSW fields should be cleared.

- Contents of register 14 plus 12 - The channel program is not posted complete, and its request element is not made available. (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 may be entered on abnormal conditions, such as: unit check, unit exception, wrong length indication, program check, protection check, channel data check, channel control check, interface control check, chaining check, out-of-extent error, and intercept condition (i.e., device end error). It may also be entered when an EXCP is issued for a DCB that has already been purged.

1. When this appendage is entered due to a unit exception and/or wrong length record indication, the IOBECB code is set to X'41'. For further information on these conditions see "Channel End Appendage."

2. When the appendage is entered due to an out-of-extent error, the IOBECB code is set to X'42'.

3. When the appendage is entered due to an intercept condition the IOBECB code is set to X'44'. The intercept condition signals that an error was detected at device end after channel end on the previous request.

4. When the appendage is entered due to an EXCP being issued to an already purged DCB, this request will enter the abnormal end appendage with the IOBECB code set to X'48'. This applies only to related requests.

5. When the appendage is entered with the IOBECB code set to 7F, it may be due to a unit check, program check, protection check, channel data check, channel control check, interface control check or chaining check. When the IOBECB code is 7F, it may be the first detection of an error in the associated channel program, or it could occur after an error routine has attempted to correct the error but was unsuccessful in its retry. Under these two conditions, the IOBERR flag is set; it indicates that the error routine is in control but has not yet declared the error to be 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 in the IOB. However, when the ECB code is X'42' or X'48', these fields are not applicable. For X'44' the CSW is applicable, but the sense is valid only if the unit check bit is set. If you use the return address in register 14 to return control to the system, the channel program is posted complete, and its request element is made available. (The SYNADAF macro instruction described in the Supervisor and Data Management Macro Instructions publication may be used in an error analysis routine to analyze permanent I/O errors.) 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", "Flags 2", and "Flags 3" fields of the input/output block and set the "Error Counts" field to zero. As an added precaution, the IOBSNS and IOBCSW fields should be cleared.

- Contents of register 14 plus 12 - The channel program is not posted complete, and its request element is not made available. (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 Supervisor and Data Management Macro Instructions publication. 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 1 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.

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

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.

DCB Address			
+0	The device dependent portion of the data control block varies in length and format according to specifications in the DSORG and DEVD parameters. Illustrations of this portion for each device type are included in the description of the DEVD parameter.		
+ 4			
+ 8			Device Dependent
+12			
+16			
+20	BUFNO	BUFCB	
+24	BUFL	DSORG	Common Interface
+28	IOBAD		
+32	BFTEK, BFALN HIARCHY	EODAD	
			Foundation Block Extension
+36	RECFM	EXLST	
+40	(TIOT)	MACRF	
+44	(IFLGS)	(DEB Address)	Foundation Block
+48	(OFLGS)	Reserved	
+52	OPTCD		
+56	Reserved		
+60	EOEA	PCIA	EXCP Interface
+64	SIOA	CENDA	
+68	XENDA	Reserved	

•Figure 1. Data Control Block Format for 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 the Supervisor and Data Management Services publication.

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 Supervisor and Data Management Macro Instructions publication.

BFTEK={S|E}
specifies the buffer technique as either simple or exchange. BFTEK bits 0 and 5 specify whether hierarchy 0 or hierarchy 1 is used to form the buffer pool. If HIARCHY={0|1} is omitted from the DCB, the buffer pool is formed in hierarchy 0.

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:

```
DCB          ┌────────┬──────────────────────┐
Address + 4  │Reservd │                      │
             ├────────┘                      │
             │                               │
        + 8  │     (FDAD - MBBCCHHR)         │
             │                               │
             │        ┌────────┬─────────────┤
        +12  │        │ DVTBL  │  Reserved   │
             ├────────┼────────┼─────────────┤
        +16  │KEYLEN  │ DEVT   │  (TRBAL)    │
             └────────┴────────┴─────────────┘
```

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. OPEN will initialize DVTBL and DEVT.

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

```
DCB          ┌───────┬──────────────────────┐
Address +16  │KEYLEN │       Reserved       │
             └───────┴──────────────────────┘
```

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

```
DCB          ┌──────────────────────────────┐
Address +12  │         (BLKCT)              │
             ├───────┬───────┬───────┬──────┤
        +16  │TRTCH  │Reservd│DEN    │Resrvd│
             └───────┴───────┴───────┴──────┘
```

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. When using EXCP to process a tape data set open at a checkpoint, you must be careful to maintain the correct count; otherwise the system may position the data set incorrectly when restart occurs.

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

```
DCB          ┌───────┬──────────────────────┐
Address +16  │CODE   │       Reserved       │
             └───────┴──────────────────────┘
```

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

```
DCB          ┌───────┬──────────────────────┐
Address +16  │PRTSP  │       Reserved       │
             └───────┴──────────────────────┘
```

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

```
DCB          ┌───────────┬──────────────────┐
Address +16  │MODE,STACK │     Reserved     │
             └───────────┴──────────────────┘
```

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. (A dummy data set may not be opened for EXCP.) 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 EOV macro instruction description.

CONTROL BLOCK FIELDS

The fields of the input/output block, event control block, and data extent block are illustrated and explained here; the data control block fields have been described with the parameters of the DCB macro instruction in the section "EXCP Programming Specifications."

Input/Output Block Fields

The input/output block 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 2, 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.

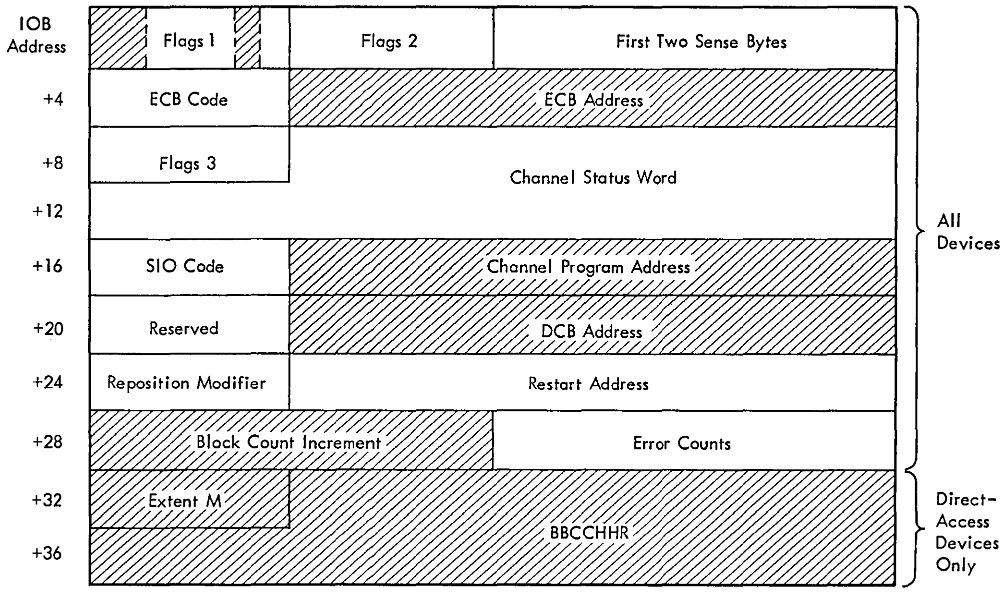


Figure 2. Input/Output Block Format

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 3). 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.

WAIT Bit=0	Complete Bit=1	Remainder of Completion Code
0	1	2 — 31

Figure 3. 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 and Purge Macro Instruction

If you want to use the RESTORE or PURGE macro instruction, you must either add the 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 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 the address of a pointer to the first IOB address in a previously purged IOB list. It could be the DEBUSRPG field in the data extent block (see "SVC Purge Routine").

Control Statements Required

```
//jobname       JOB     {parameters}
//stepname      EXEC    PGM=IEBUPDTE,PARM=NEW
//SYSPRINT      DD      SYSOUT=A
//SYSUT2        DD      DSNAME=SYS1.MACLIB,DISP=OLD
//SYSIN         DD      DATA
./        ADD   NAME=RESTORE,LIST=ALL
                .
                .
                .
                RESTORE Macro Definition
                .
                .
                .
./        ENDUP
/*
```

RESTORE Macro-Definition

```
         MACRO
&NAME    RESTORE      &LIST
         AIF          ('&LIST' EQ '').E1
&NAME    IHBINNRA     &LIST                   LOAD REG 1
         SVC          17                      ISSUE SVC FOR RESTORE
         MEXIT
.E1      IHBERMAC     01,150                  LIST ADDR MISSING
         MEND
```

PURGE Macro Instruction

The PURGE macro instruction is used to return request elements to the I/O supervisor inactive queue (next available).

PURGE Macro Definition

```
            MACRO
 &NAME      PURGE          &LIST
            AIF            ('&LIST'EQ'').E1
 &NAME      IHBINNRA       &LIST                   LOAD REG 1
            SVC            16
            MEXIT
 .E1        IHBERMAC       01,147                  LIST ADDR MISSING
            MEND
```

Control Statements Required

```
//jobname        JOB       {parameter}
//stepname       EXEC      PGM=IEBUPDTE,PARM=NEW
//SYSPRINT       DD        SYSOUT=A
//SYSUT2         DD        DSNAME=SYS1.MACLIB,DISP=OLD
//SYSIN          DD        *
./          ADD  NAME=PURGE,LIST=ALL
                 .
                 .
                 PURGE Macro Definition
                 .
                 .
./          ENDUP
/*
```

Name	Operation	Operand
symbol	PURGE	User Purge Parameter List

The purge parameter list is constructed in the user's program area. Depending on the options specified in the PURGE parameter list, elements can be purged from

1. The asynchronous exit queue of the task supervisor.
2. The request blocks chained to the TCB.
3. The I/O supervisor logical channel queues.

You can bypass the purge of the RBs chained to the TCB by setting bit 5 of the option byte. The parameter list is a three-word list constructed prior to issuing the PURGE macro instruction; this list must fall on a fullword boundary. It is constructed as follows:

Word 1

Byte 1
(options byte)

Bit 0 -		Specified DEB or DEB chain
	=0 -	Purge request elements associated with complete DEB chain starting at the DEB specified in bytes 2, 3, and 4 of word 1.
	=1 -	Purge only the request elements associated with the DEB specified by bytes 2, 3, and 4 of word 1.
Bit 1 -		POST request purged or ignore posting.
	=0 -	Do not POST the purged requests.
	=1 -	POST the purge requests, code = X'48'.
Bit 2 -		HALT I/O or quiesce active requests.
	=0 -	Allow the active requests to quiesce.
	=1 -	HALT the I/O operations. (The HALT I/O is simulated if the operation is a SEEK.
Bit 3 -		Purge all or only related requests.
	=0 -	Purge all requests.
	=1 -	Purge only related requests.
Bit 4 -		(Spare)
Bit 5 -		Purge all queues or bypass RB purge.
	=0 -	Purge AEQ, RB, and I/O Supervisor logical channel queues.
	=1 -	Purge only the I/O Supervisor logical channel queue(s) and AEQ.
Bit 6 -		Purge by TCB or DEB
	=0 -	Purge by DEB
	=1 -	Purge by TCB
		Note: This bit must be zero in order to honor bit 0. If this bit is one, all requests associated with the TCB are purged, and bit 0 is ignored.
Bit 7 -		(Spare)

Bytes 2, 3, and 4
DEB address - not required if purging by TCB.

Word 2

Byte 1
completion code

Bytes 2, 3, and 4
TCB address - if none, the current TCB is used.

Word 3

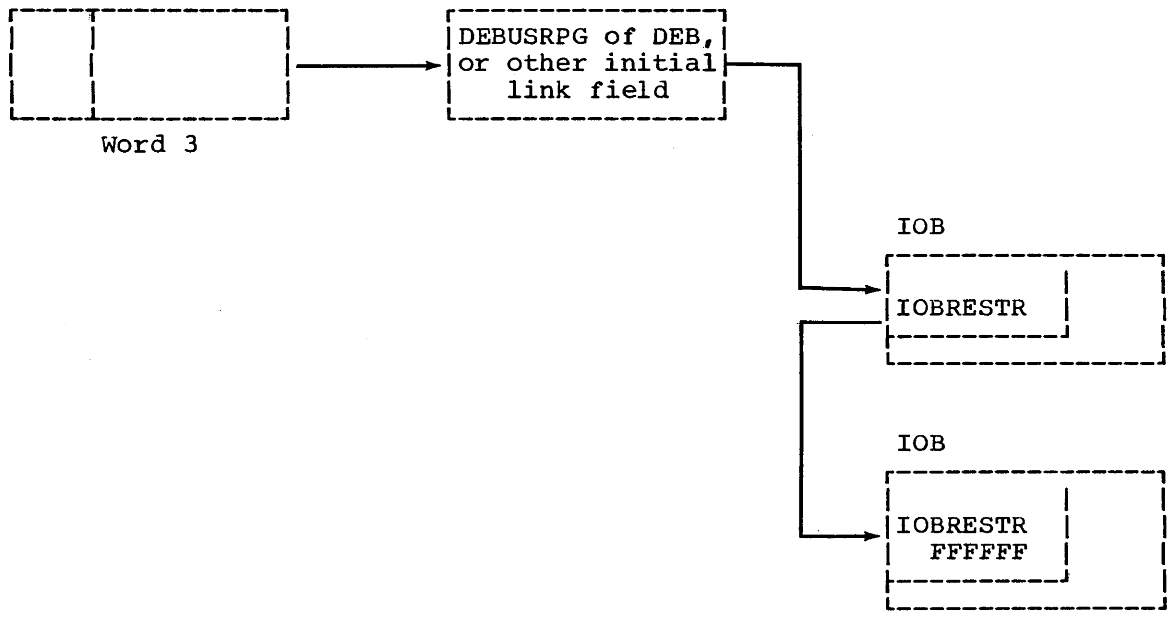
Byte 1
: Quiesce indicator field. It will indicate X'01' if one or more requests are quiescing.

Bytes 2, 3, and 4
: Address of the initial link field for chaining IOBs that are purged. The initial link field can be the user purge field in the DEB (DEBUSRPG) or any area you select. The initial link field points to the first IOB in the chain. At the completion of purge, the contents of word 3 are unpredictable. No chaining is done when TCB with HALT I/O option is specified.

  If the IOB restart field (IOBRESTR) is used as a link field, the last one will contain X'FFFFFF' in its three low-order bytes.

The following figure below shows the IOB chain.

Chaining IOBs



IOB Chain for PURGE

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: Supervisor and Data Management Services publication (Form C28-6646) 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 4). 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.

WAIT Bit=0	Complete Bit=1	Remainder of Completion Code	
0	1	2	31

Figure 4. 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 5.

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 5. 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 the IBM System/360 Operating System: Supervisor and Data Management Services publication, Form C28-6646 and the IBM System/360 Operating System: Supervisor and Data Management Macro Instructions publication, Form C28-6647.

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.

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
	CVT	

Control Statements Required

```
//jobname       JOB     {parameters}
//stepname      EXEC    PGM=IEBUPDTE,PARM=NEW
//SYSPRINT      DD      SYSOUT=A
//SYSUT2        DD      DSNAME=SYS1.MACLIB,DISP=OLD
//SYSIN         DD      *
./        ADD   NAME=CVT,LIST=ALL
                .
                .
                .
                CVT Macro-Definition
                .
                .
                .
./        ENDUP
/*
```

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:

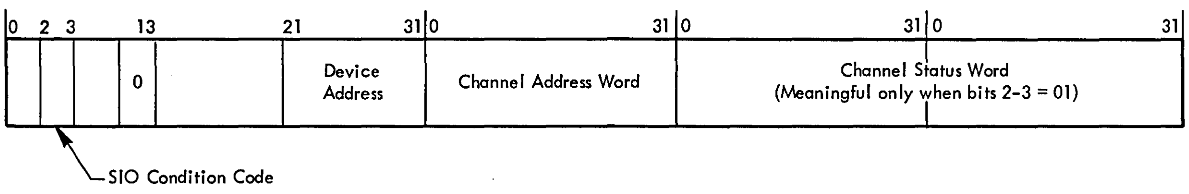
- SIO instruction execution.
- SVC interruption.
- I/O interruption.

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.

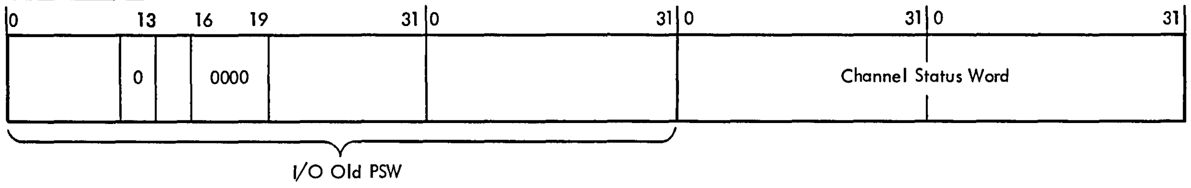
Table Entry Formats

Table entry formats are as follows:

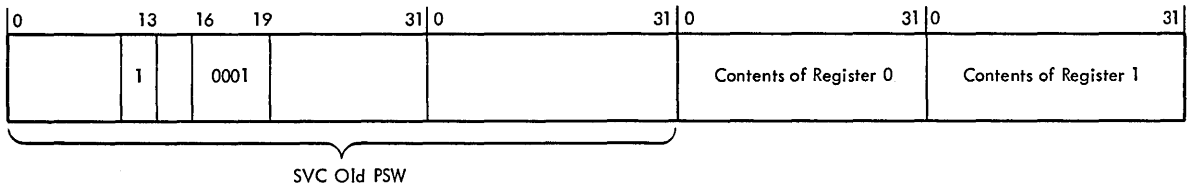
SIO Instruction



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:

```
|0                     31|0                     31|0                     31|
├────────────────────────┼────────────────────────┼────────────────────────┤
|     Address of the     |     Address of the     |     Address of the     |
|       Last Entry       |     Table Beginning    |       Table End        |
└────────────────────────┴────────────────────────┴────────────────────────┘
```

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

The abnormal termination dump lists the SIO, SVC, and I/O interruptions table entries, starting with the oldest. A number is assigned to each entry and the oldest entry is 0001.

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: Supervisor and Data Management Services publication (Form C28-6646) contains a general description of the data set protection feature.

The IBM System/360 Operating System: Messages and Codes publication (Form C28-6631) 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 (containing SYS1.NUCLEUS and SYS1.SVCLIB). 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.

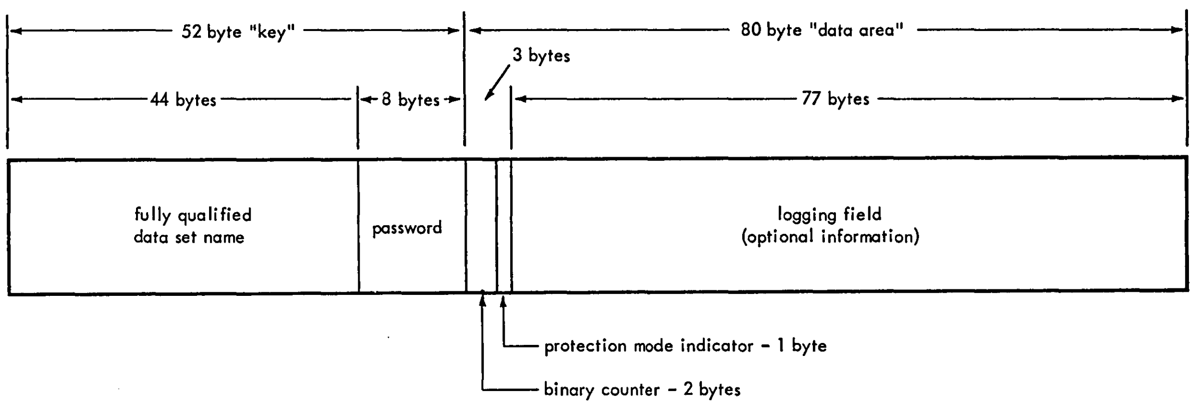
If a data set is to be placed under protection, it must have a protection indicator set in its label (DSCB or header 1 tape label). This is done by the operating system when the data set is created. The protection indicator is set in response to an entry in the LABEL= parameter of the DD statement associated 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.

Users who wish to have the password supplied by some method other than operator key-in may replace the password reading module with their own routine. The READPSWD source module may be used as a base for writing a new module. In this case, the new object module replaces module READPSWD on the SVCLIB.

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.


52 byte "key"
80 byte "data area"
3 bytes
44 bytes
8 bytes
77 bytes
fully qualified data set name
password
logging field (optional information)
protection mode indicator - 1 byte
binary counter - 2 bytes

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. Read only and read/write protection for a data set can be attained by including the same data set name in the password data set twice and giving it different passwords. 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 Concatenation

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

SCRATCH and RENAME Functions

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 Messages and Codes 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 PRESRES Volume Characteristics List

This chapter describes the creation and use of a direct access volume characteristics list that is placed in the system parameter library under the member name PRESRES.

Prerequisite Publications

The *IBM System/360 Operating System: Job Control Language* publication (Form C28-6539) discusses volume characteristics and states.

The *IBM System/360 Operating System: Messages and Codes* publication (Form C28-6631) describes the operator messages and responses associated with system use of the volume characteristics list.

The PRESRES Volume Characteristics List

You may use the PRESRES volume characteristics list to define the mount and allocation characteristics of direct access device volumes used by your installation. Use of the list enables you to predefine the mount characteristics (permanently resident, reserved) and allocation characteristics (storage, public, private) for any, or all, direct access device volumes used by your installation. The Job Control Language publication provides a full discussion of the volume characteristics and the operating system's response to the various designations. The information presented here describes the creation of the characteristics list, the format and content of entries in the list, and how the operating system uses the list.

Creating the List

You use the IEBUPDTE utility program to place the list (under the member name PRESRES) in the system parameter library, SYS1.PARMLIB. This utility is also used to maintain the list.

PRESRES Entry Format

Each PRESRES entry is an 80-byte record, consisting of a 6-byte volume serial number field, a 1-byte mount characteristic field, a 1-byte allocation characteristic field, a 4-byte device type field, a 1-byte mount-priority field, and an optional information field. Commas are used to delimit the fields, except the optional information field is *always* preceded by a blank. All character representation is EBCDIC. This format is shown below.

```
┌──────────────────┬─┬─┬─┬─┬─┬─────────────┬─┬─┬─┬────────────┐
│ Volume Serial    │,│ │,│ │,│Device Type  │,│ │ │Optional    │
│ Number 6 Bytes   │ │ │ │ │ │ 4 Bytes     │ │ │ │Information │
└──────────────────┴─┴─┴─┴─┴─┴─────────────┴─┴─┴─┴────────────┘
                       ↑   ↑                      ↑
                       │   │                      └──Mount Priority--1 Byte
                       │   └─────Allocation Characteristic--1 Byte
                       └────Mount Characteristic--1 Byte
```

The volume serial number consists of up to six characters, left justified.

Mount characteristics are defined by:

0 to denote permanently resident
1 to denote reserved

The default characteristic is "permanently resident" and is assigned if any character other than 0 or 1 is present in the field.

Allocation characteristics are defined by:

0 to denote storage
1 to denote public
2 to denote private

The default characteristic is "public" and is assigned if any character other than 0, 1, or 2 is present in the field.

The device type is defined by:

A four digit number designating the type of direct access device on which the volume resides, e.g., the IBM 2311 Disk Storage Drive is indicated by the notation 2311. Note that this field only indicates the basic device type for the associated volume. <u>You must advise the operator if the device requires special features (such as track overflow) to process the data on the designated volume</u>.

The mount priority field is used to suppress mount messages at IPL time for a volume; the alphabetic character N should be inserted in this field to suppress the mount message. This field allows the user to list seldom used volumes in the PRESRES list without having a mount message issued at each IPL. When these volumes are required, they may be mounted and attributes will be set from the PRESRES list entry. If the user does not wish to have the mount message suppressed, he may omit the mount priority field and the preceding comma.

The optional information field contains:

Any descriptive information about the volume that you may wish to enter. This information is not used by the system, but will be available to you on a print-out of the list. If necessary, comments may start in the second byte after the mount priority field or if the mount priority field is omitted, in the second byte following the comma after the device type field.

Embedded blanks are not permitted in the volume serial, mount, allocation, or device type fields.

Operational Characteristics

Upon receiving control from the nucleus initialization program (NIP), the scheduler compares the volume serial numbers in the PRESRES characteristics list with those of currently mounted direct access volumes. Each equal comparison results in the assignment to the mounted volume of the characteristics noted in the PRESRES entry. (Fields in the unit control block for the device on which the volume is mounted are set to reflect the desired characteristics.) If the volume is: the IPL volume; the volume containing the data sets SYS1.LINKLIB, SYS1.PROCLIB, SYS1.SYSJOBQE; or a physically nondemountable volume (such as a 2301 drum storage unit), the mount characteristic (permanently resident) has already been assigned and only the allocation characteristic is set.

A mounting list is issued for the volumes in the PRESRES characteristics list that are not currently mounted (except those for which mounting messages have been suppressed) and the operator is given the option of mounting none, some, or all of the volumes listed. The mount and allocation characteristics for the volumes mounted by the operator are set according to the PRESRES list entry for the volume. The operator selects the unit on which the volume is to be mounted.

The Messages and Codes publication describes the operator messages and responses associated with the use of the PRESRES volume characteristics list.

After the scheduler has finished PRESRES processing reading of the job input stream begins, and the PRESRES list is not referred to again until the next IPL.

Volume characteristics assigned by a PRESRES list entry are inviolate. They cannot be altered by subsequent references to the volume in the input stream.

Note:

1. A PRESRES entry identifying a physically nondemountable volume will appear in the mount list issued to the operator if the volume (device) is OFFLINE or is not present in the system.

2. Use of the PRESRES list can only be suppressed by deleting the member from the parameter library (SYS1.PARMLIB).

Programming Considerations

The only way to assign an allocation characteristic other than "public" to volumes whose mount characteristic is "permanently resident" is through a PRESRES characteristic list entry.

Selection of the volumes for which PRESRES entries are to be created should be done so that critical volumes are protected. Since the combination of mount and allocation characteristics assigned to a specific volume determine the types of data sets that can be placed on the volume and its usage, you can exercise effective control over the volume through a PRESRES list entry.

Using the Resident BLDL Table, Access Method, SVC Routine, and Job Queue Options (PCP and MFT) and the Link Pack Area (MVT)

Section 1 of this chapter discusses the BLDL Table, Access Method, SVC Routine and Job Queue options and provides guidelines for their use. The purpose of these options is to improve performance by reducing or eliminating the access time required to obtain the records or routines with which these options are concerned. You may incorporate any or all of these options in the PCP (Primary Control Program) or MFT (Multiprogramming with a Fixed Number of Tasks) configurations of the operating system.

Section 2 of this chapter discusses the inclusion of SVC routines, reenterable load modules, and linkage library directory entries in the Link Pack Area of Multiprogramming with a Variable Number of Tasks (MVT) configurations of the operating system.

Section 3 of this chapter discusses the link library list and provides guidelines for its use. The purpose of this facility is to allow concatenation of data sets for SYS1.LINKLIB. The link library list must be included in the system.

Prerequisite Publications

The IBM System/360 Operating System: System Generation publication (Form C28-6554) describes how to specify the options and content of the link pack area at system generation time.

The IBM System/360 Operating System: Supervisor and Data Management Services publication (Form C28-6646) contains a general discussion of the BLDL function.

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

The IBM System/360 Operating System: Storage Estimates publication (Form C28-6551) provides storage requirement information for the options and link pack area.

The IBM System/360 Operating System: Messages and Codes publication (Form C28-6631) contains the operator message and replies associated with the options and link pack area.

Section 1: The Resident BLDL Table, Resident Access Method, Resident SVC Routine, and Resident Job Queue Options (PCP and MFT)

These options, when included in a PCP or MFT configuration of the operating system, enable you to place in main storage:

1. All, or a selection of, linkage library (SYS1.LINKLIB) directory entries.
2. A selected group of access method routines.
3. A selected group of type 3 and 4 SVC routines.
4. A specified number of job queue records. (PCP only)

Placement occurs during the initial program load (IPL) process, except for the job queue option: a main storage area large enough to contain the specified number of job queue records is reserved at IPL time. The main storage area that these resident routines and records occupy becomes part of the "fixed storage" area of the system. In effect, the nucleus is expanded.

These options are included in the system when it is generated. The System Generation publication describes the procedure. The resident SVC routine option requires that the Transient SVC Table option also be included in the system.

If you wish to exercise control over these options (except the Transient SVC Table option) at IPL time, you must also specify the operator communication facility for these options when the system is generated.

You specify the linkage library (SYS1.LINKLIB) directory entries, the access method routines, and the type 3 and 4 SVC routines to be made resident through lists of linkage library, access method, and SVC routine load module names placed in the parameter library (SYS1.PARMLIB). The resident job queue option does not use a parameter library list; you specify, at system generation time (and, if desired, at IPL time) a value indicating the number of job queue records to be made resident.

A standard list and alternative lists of load module names may exist for the BLDL table, access method, and SVC routine options. The standard list (so called because its member name in the parameter library is predefined) is automatically referred to during the IPL process when the operator communication facility is not included in the system with the options. When the operator communication facility is included, the operator must designate which list is to be used. IBM provides suggested standard lists for the resident access method and SVC routine options. These lists are in the starter system parameter library.

Inclusion of the operator communication facility enables full control over all the options at IPL time, i.e., selection of alternative or standard lists, specification of the number of resident job queue records, and suppression of the options until the next IPL. Otherwise, the options are in effect at every IPL, using the standard lists and the job queue record value specified during system generation.

The Messages and Codes publication describes the message (message number IEA101A) and replies associated with the options.

The balance of this chapter discusses the function of each option, the creation of the parameter library lists, and, in Appendix A, lists the content of the resident access method and resident type 3 and 4 SVC routine standard lists.

The Resident BLDL Table Option

System issued ATTACH, LINK, LOAD, or XCTL macro instructions requesting load modules from partitioned data sets cause a search of the data set directory for the location of the requested module (the BLDL table operation) and a fetch of the module. The resident BLDL table option eliminates the directory search required during execution of these macro instructions when a load module (whose directory entry is resident) is requested from the linkage library.

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.PARMLIB, the names of those linkage library load modules whose directory entries are to be made resident. The member name for the standard list is IEABLD00. The load module names must be listed in the same order as they appear in the directory; that is, they must be in ascending collating sequence. Creation of parameter 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 with PCP and MFT; MVT requires 56 bytes for each entry.)
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 these 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.PARMLIB, 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 (see Appendix A), and is in SYS1.PARMLIB of the starter system under the standard member name. The modules are listed by frequency of use; the least used module is first in the list.

The creation of parameter library lists 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 the byte requirements for access method modules eligible to be made resident. The byte requirement of the code supporting the option is also provided.

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.

The Resident SVC Routine Option

This option places type 3 and 4 SVC routine load modules in main storage. Some, or all, of the modules associated with a SVC service routine may be made resident. Placing the most frequently used SVC load modules of a system service routine, such as OPEN, in main storage improves system performance. All load modules of type 3 and 4 SVC routines are eligible for residence. For type 3 SVC load modules and initial type 4 SVC load modules, the SVC table entries associated with these modules are adjusted to reflect an entry point address rather than a relative track address. A resident SVC load list is used by the XCTL macro instruction for transfer of control between resident type 4 SVC load modules.

You list, in a member of SYS1.PARMLIB the type 3 and 4 SVC load modules to be made resident. The member name for the standard list is IEARSV00. The modules are listed by frequency of use; the least used module is first in the list. A standard list of type 3 and 4 SVC load modules is provided by IBM (see Appendix A) and is in SYS1.PARMLIB of the starter system under the standard member name. The creation of parameter library lists is discussed later in this chapter.

Storage Requirements

The Storage Estimates publication provides the byte requirements of type 3 and 4 SVC routines eligible to be made resident. The byte requirement of the code supporting the option is also provided.

The Resident Job Queue Option (PCP only)

This option places a specified number of system job queue records in main storage rather than in external storage (the SYS1.SYSJOBQE data set). The records are taken sequentially from the beginning of the queue. There is one break in the sequence which is noted in the next section "Operational Characteristics."

Operational Characteristics

The job queue is formatted as a series of 176 byte records. The first 42 records form a "fixed group" of job queue records used by the scheduler. These 42 records are always present in the job queue. Of this group, 26 records are used by the interpreter routines of the scheduler as a work area. These 26 records are never made resident by the option. The remaining 16 records in the "fixed group" may be made resident. After the "fixed group" of records, a series of records forming a "variable group" of job queue records is developed. The number of records in the "variable group" fluctuates from job to job reflecting the make-up of the input job stream for the job being read in. All records in the "variable group" may be made resident.

Starting with the first (in sequence) of the 16 eligible "fixed group" records, the option places the specified number of records in main storage. For example, a specification of 5 resident records will place the first 5 of the 16 "fixed group" records in main storage; a specification of 20 resident records will place all 16 of the "fixed group" records in main storage plus the first 4 records from the "variable group."

Reference to a specific job queue record causes a test to be made -- in resident queue or in external storage -- and the record is referred to accordingly.

In an MFT configuration of the operating system only the "variable group" job queue records developed from the input job stream for the lowest priority partition may be made resident.

Determining Resident Job Queue Size

The storage occupied by the resident job queue cannot be allocated to any other use, therefore you must determine the amount of storage your installation can afford to devote to a resident job queue. Since the size of the queue can be varied from IPL to IPL you may want to estimate several sizes -- each estimate reflecting a feasible job queue size in view of the work to be performed after the IPL.

The following formula can be used to estimate the number of resident job queue records developed for a given job. The constant (16) represents the 16 "fixed group" records that are always developed and are eligible for inclusion in the resident job queue.

$$\text{Number of Records} = 16 + \frac{B}{3} + 2C + \frac{E}{28} + \frac{F}{176} + 3G + \frac{H-5}{15} + \frac{J}{22}$$

Where:

B = the number of data sets passed between job steps.

C = the number of steps in the job.

E = the number of volume serial numbers specified in the DD statements for each job step. (Evaluate each job step separately and sum the results to obtain the total value.)

F = the number of characters in data set names, including qualifiers, appearing in DD statements in the parameter VOL=REF=dsname. (Evaluate each job step separately and sum the results to obtain the total value.)

G = the number of DD statements in the job.

H = the number of volume serial numbers specified in each DD statement (if $H \leq 5$, $H-5=0$). (Evaluate each DD statement separately and sum the results to obtain the total value.)

J = the total number of job control language statements used to describe the job, when all messages are to be written on the system output device, otherwise J=1.

Multiplying the number of records by 176 provides the resident job queue size in terms of bytes.

If possible, the entire set of eligible job queue records should be made resident. It is recommended that at least the 16 eligible records from the "fixed group" of job queue records be made resident.

Creating Parameter Library Lists

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

IEABLD00	for the BLDL table option
IEAIGG00	for the access method option
IEARSV00	for the SVC routine option
LNKLST00	for the link library list option

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

Note: The nucleus initialization program (NIP) will search the system catalog to locate the SYS1.PARMLIB data set. If it is not found in the catalog, SYS1.PARMLIB is assumed to reside on the IPL volume. If no VTOC entry can be found, the operator will receive message IEA211I "OBTAIN FAILED FOR SYS1.PARMLIB DATA SET". Message IEA208I "RAM, BLDL, RSVC FUNCTIONS INOPERATIVE" will follow either of these messages. Processing will continue; however, any resident functions dependent on parameter lists contained in the parameter library will be omitted from the system nucleus.

Your input format (to IEBUPDTE) for the lists is the same for all three options, 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 may construct alternative lists for all three options and place them in the parameter library. Member names for these alternative lists are of the form:

```
IEABLDxx for the BLDL option
IEAIGGxx for the resident access method option
IEARSVxx for the resident SVC routine option
         where xx can be any two alphameric characters.
LNKLST00 for the link library list option
```

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 all three 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, access method routines, or SVC routines are 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. Note that the module names are listed in ascending collating sequence as required for the resident BLDL option. Resident access method or SVC modules should be listed in order of anticipated frequency of use.

```
//BLDLIST EXEC PGM=IEBUPDTE,PARM=NEW
//SYSPRINT DD SYSOUT=A
//SYSUT2 DD DSNAME=SYS1.PARMLIB,DISP=OLD
//SYSIN DD *
./      ADD    NAME=IEABLDAE,LIST=ALL
./      NUMBER NEW1=01,INCR=02
 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: During IPL 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) to easily identify format errors, e.g., a 9 character name, or incorrect name specifications.

Section 2: Using the Link Pack Area (MVT)

In MVT configurations of the operating system the link pack area is always present in main storage, and, as a minimum, always contains a group of system-specified load modules concerned with job management processing. You may extend the link pack area to contain:

- Load modules of nonresident SVC routines.
- Other reenterable load modules from the system linkage library (SYS1.LINKLIB) and SVC library (SYS1.SVCLIB).
- A table (the BLDL table) containing directory entries of load modules in the linkage library (SYS1.LINKLIB).

Essentially, the link pack area in MVT configurations is the counterpart of the PCP and MFT configuration residency options (except job queue) discussed in Section 1 of this chapter.

You select the load modules to be made resident and the linkage library load modules whose directory entries are to appear in the BLDL table. You indicate your choices to the system through lists of the load module names placed in the system parameter library (SYS1.PARMLIB). Standard (default) and alternative lists may be made up for each category.

During the initial program loading (IPL) process the nucleus initialization program places the specified load modules in the link pack area and constructs the BLDL table. The load modules and BLDL table remain, unchanged, in the link pack area until the next IPL procedure is performed, and can be used by all tasks.

PROCEDURE FOR USING THE LINK PACK AREA

The following material, under the headings "Initialization," "Creating Parameter Library Lists," "List Specification," and "Operational Characteristics," provides guidelines for use of the link pack area.

Initialization

When your MVT operating system is generated you must indicate whether you wish to extend the link pack area to include nonresident SVC routines, other reenterable load modules, the BLDL table, or any combination of these. The *System Generation* publication describes the procedure (the SUPRVSOR macro instruction).

To exercise full control over the content of the link pack area (except for the mandatory modules which are always loaded) you must specify, at system generation, that the operator communication facility be included. The *System Generation* publication describes the procedure (the SUPRVSOR macro instruction). The operator communication facility enables you to respecify the content of the link pack area at each IPL.

Creating Parameter Library Lists

As discussed under the same heading in Section 1 of this chapter you use the IEBUPDTE utility program to place your load module name lists in the parameter library (see Section 1). The format of your input to the utility program is the same.

Note: In an MVT configuration of the operating system, updating of the system data set SYS1.PARMLIB should not be attempted while other jobs are operative. The recommended procedure is described in the *Operator's Guide* publication.

List Specification

The names and content of the parameter library lists are:

List Name	List Content
IEARSV00 -- standard list IEARSVxx[1] -- alternative list(s)	Names of type 3 and 4 SVC routine load modules.
IEAIGG00 -- standard list IEAIGGxx[1] -- alternative list(s)	Names of reenterable load modules in the SVC and linkage libraries.
IEABLD00 -- standard list IEABLDxx[1] -- alternative list(s)	Names of linkage library load modules whose directory entries are to be entered in the BLDL table.
LNKLST00 -standard list	SYS1.LINKLIB -- Additional data sets may be concatenated after system generation via IEBUPDTE.

[1]xx can be any two alphameric characters.

SVC LOAD MODULE LISTS: Only one standard SVC load module list -- IEARSV00 -- may be present in the parameter library. You may create as many alternative lists as your needs require. To use alternative lists, you must have specified the operator communication facility at system generation. The standard list is the only list referred to by the nucleus initialization program at IPL time if the operator communication facility is not installed in the system. A suggested standard list, supplied by IBM, is shown in Appendix A of this chapter. The Storage Estimates publication provides a list (with storage requirements) of IBM originated type 3 and 4 SVC load modules that are eligible for inclusion in the link pack area.

REENTERABLE LOAD MODULE LISTS: Only one standard list of reenterable load modules -- IEAIGG00 -- may be present in the parameter library. You may create as many alternative lists as your needs require. You cannot incorporate load modules from the SVC library and the linkage library in one list. Use of the standard and/or alternative lists is as discussed under SVC LOAD MODULE LISTS. A suggested standard list, supplied by IBM, is shown in Appendix A of this chapter. The Storage Estimates publication provides a list (with storage requirements) of IBM originated reenterable load modules (other than SVC modules) that are eligible for inclusion in the link pack area.

BLDL TABLE LISTS: Only one standard list of linkage library load modules -- IEABLD00 -- may be present in the parameter library. You may create as many alternative lists as your needs require. Use of the standard and/or alternative lists is as discussed under SVC LOAD MODULE LISTS. A suggested standard list, supplied by IBM, is shown in Appendix A of this chapter. Each load module name in the list originates a 56-byte entry in the BLDL table. You compute the amount of storage required for the BLDL table constructed from any given list by multiplying the number of names in the list by 56.

Note: You must arrange the linkage library load module names in your list(s) in the same order as they appear in the linkage library directory. All load modules in the linkage library are eligible to have their directory entries placed in the BLDL table.

Operational Characteristics

Your specifications at system generation time determine the types of load modules that are placed in the link pack area and whether a BLDL table is constructed in the link pack area. In response to your specifications, the nucleus initialization program (at IPL time) refers to the parameter library lists to determine the specific load modules to be placed in the link pack area and/or the specific linkage library directory entries to be placed in the BLDL table. In the absence of the operator communication facility only the standard lists are referred to. If the operator communication facility is present the operator must specify the list or lists to be used. The operator may:

- Specify use of the standard list for each category, i.e., SVC load modules, other reenterable load modules, the BLDL table content.
- Specify alternative lists for each category, or a combination of the standard list and alternative lists. Up to four lists may be specified for each load module category.

  Only one list may be specified for the BLDL table.
- Specify that (for the current IPL) the loading of modules and/or construction of a BLDL table be suppressed. Each category is treated independently.

With operator communication you can specify, at each IPL, the content of the link pack area extension. The number and type of load modules selected for inclusion in the link pack area, and the content of the BLDL table, can thus be altered to reflect the type of workload to be presented to the system after the IPL.

The Messages and Codes publication describes the operator message and responses associated with use of the link pack area.

PROGRAMMING NOTES

A list of the load modules always placed in the link pack area by the system is contained in the Storage Estimates publication. The main storage space requirements of these modules determines the basic (minimum) size of the link pack area. The area is extended by the number of storage bytes needed to accommodate the load modules and BLDL table content specified at IPL time.

Placing the initiator/terminator load module IEFSD061 in the link pack area enables the system to make more efficient use of the dynamic area of storage. The operating system allocates to each job a part of a region not less than the size required to accommodate the initiator-terminator. This allocation is from processor storage (hierarchy 0) and occurs even when the REGION parameter requests less than the required space or no space. After initiation, the part of the region in hierarchy 0 is reduced by as much as 40,000 bytes when the job terminator is resident in the link pack area.

EXAMPLE OF LINK PACK AREA SPECIFICATION

The following example illustrates the extension of the link pack area to contain SVC load modules, other reenterable load modules, and a BLDL table. The RESIDNT field of your system generation SUPRVSOR macro instruction would look like:

```
                  .
                  .
                  .
          SUPRVSOR  RESIDNT=TRSVC,RENTCODE,BLDLTAB...
                  .
                  .
                  .
```

If you intend to alter the content of your link pack area, you would also specify: OPTIONS=COMM,... in the SUPRVSOR macro instruction.

Assume that you wish to place five lists on SYS1.PARMLIB. These lists are:

1. IEARSV00, which contains names of modules of the open SVC routine used for direct access devices.
2. IEARSV20, which contains names of modules of the close SVC routine.
3. IEAIGG01, which contains names of modules of the basic sequential access method (BSAM).
4. IEABLD00, which contains names of modules of the initiator portion of the job scheduler.
5. IEABLDF0, which contains names of modules of both the FORTRAN compiler and the initiator.

Note that there is no standard list for reenterable modules from the linkage or SVC library (IEAIGG00). This implies that you don't want modules of this type loaded unless a list is explicitly specified.

To place these lists in SYS1.PARMLIB, you could use the IEBUPDTE utility program as shown:

```
//ADDLISTS    JOB       61938,R.L.WILSON
//STEP        EXEC      PGM=IEBUPDTE,PARM=NEW
//SYSPRINT    DD        SYSOUT=A
//SYSUT2      DD        DSNAME=SYS1.PARMLIB,DISP=OLD
//SYSIN       DD        DATA
./            ADD       NAME=IEARSV00,LIST=ALL
./            NUMBER    NEW1=01,INCR=02
SYS1.SVCLIB   IGC00019,IGG0190Z,IGG0190I,IGG0190L,                C
                        IGG0190M,IGG0190S
./            ADD       NAME=IEARSV20,LIST=ALL
./            NUMBER    NEW1=01,INCR=02
SYS1.SVCLIB   IGC00020,IGG0200A,IGG0200B,IGG0200C,IGG0200F,       C
                        IGG0200G,IGG0200Y
./            ADD       NAME=IEAIGG01,LIST=ALL
./            NUMBER    NEW1=01,INCR=02
SYS1.SVCLIB   IGG019BA,IGG019BB,IGG019BC,IGG019BD,                C
                        IGG019BE,IGG019BF,IGG019BG,               C
                        IGG019BH,IGG019BI,IGG019BK,IGG019BL
./            ADD       NAME=IEABLD00,LIST=ALL
./            NUMBER    NEW1=01,INCR=02
SYS1.LINKLIB IEFSD061,IEFSD062,IEFW21SD,IEFWC000,                    C
                        IEFSD065,IEFW42SD,IEFSD104,IEFUM1,           C
                        IEFXJ000,IEFWD000,IEFW41SD
./            ADD       NAME=IEABLDF0,LIST=ALL
./            NUMBER    NEW1=01,INCR=02
SYS1.LINKLIB IEFSD061,IEFSD062,IEFW21SD,IEFWC000,                    C
                        IEJAAA0,IEJEAA0,IEJFAA0,IEJGAA0,             C
                        IEJJAA0,IEJLAA0,IEJNAA0,IEJPAA0,             C
                        IEJRAA0,IEJVAA0,IEJXAA0,IEFSD065,            C
                        IEFW42SD,IEFSD104,IEFUM1,IEFXJ000,           C
                        IEFWD000,IEFW41SD
./            ENDUP
/*
```

Without operator communication only the standard lists IEARSV00 and IEABLD00 would be referred to at IPL time. With operator communication use of all the lists or any combination could be specified at IPL time.

If after a given IPL you intend to extensively use the FORTRAN compiler, and BSAM with direct access devices, you would probably want to use all of these lists -- except IEABLD00 -- to specify the content of your extended link pack area. To do this your operator would specify the following in response to the SPECIFY SYSTEM PARAMETERS operator's message:

```
REPLY id, "RSVC=00,20,RAM=01,BLDL=F0"
```

If, after an IPL you intended to perform general processing without extensive use of any particular compiler or access method, you might want to put just the linkage library directory entries of initiator modules in a BLDL table. In this case, your operator's reply at IPL would be:

```
REPLY id, "RSVC=,RAM=,"
```

Since the list of initiator modules is the standard list, it need not be specified. "RSVC=," must be specified to prevent the use of the standard list of SVC modules. Although you have no standard list of reenterable modules "RAM=," should be specified to prevent NIP from performing unnecessary processing.

Section 3: The Link Library List

The link library list (LNKLST00) enables you to concatenate up to 16 data sets, on multiple volumes, to form SYS1.LINKLIB. LNKLST00 is included in the system when it is generated as a required member of SYS1.PARMLIB. If SYS1.PARMLIB does not include the member LNKLST00, SYS1.LINKLIB will be used as the system link library and a warning message will be provided.

Note: The amount of space required for SYS1.PARMLIB is discussed in IBM System/360 Operating System: Storage Estimates, Form C28-6551.

LNKLST00 contains one member, SYS1.LINKLIB. After system generation you will have the option of adding members via the IEBUPDTE utility program. Each member may have up to 16 extents. After making additions to SYS1.SVCLIB, SYS1.LINKLIB, or data sets concatenated to LINKLIB via LNKLST00, and before using the additions, IPL should be performed to update the description of the link and/or SVC library in main storage.

Your input format (to IEBUPDTE) consists of eighty character records. 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 format is:

```
[b...] SYS1.LINKLIB
```

To add member names to LNKLST00, replace the initial record with:

```
[b...]   SYS1.LINKLIB,name1,name2,name3,...
```

IBM System/360 Operating System: Messages and Codes, Form C28-6631, describes the NIP messages associated with LNKLST00.

Appendix A: Standard Lists IEAIGG00, IEARV00, IEABLD00

The content of the IBM supplied standard list IEAIGG00 is shown below.

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
IGG019FJ	QSAM (SB)	PUT Locate, Format VS, VBS 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
IGG019FB	QSAM (SB)	GET Locate, Format VS, VBS 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
IGG019FL	QSAM (SB)	PUT Move, Format VS, VBS Records
IGG019FG	QSAM (SB)	PUT Data, Format VS, VBS Records
IGG019AD	QSAM (SB)	GET Move, Format V, VB Records
IGG019FD	QSAM (SB)	GET Move, Format VS, VBS Records
IGG019FF	QSAM (SB)	GET Data, Format VS, VBS 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

Note: The BSAM and SAM Modules are necessary if the checkpoint/restart facility is to be used.

The content of the IBM supplied standard list IEARSV00 is shown below.

Module Name	Function
IGC0001F	Purge Routine
IGC0001I	Open - Initial Load
IGC0005E	EOV - Initial Load
IGC0002_ (ƀ,?)*	Close - Initial Load
IGG0190L	Open - Merge and Access Method Determination
IGG0190M	Open - Merge and DCB Exit Routine
IGG0190N	Open - Final Load
IGG0190S	Open - Rewrite JFCB
IGG0191A	Open - DEB Construction
IGG0191B	Open - Main Executor
IGG0191D	Open - Direct Access Executor
IGG0191G	Open - Tape and Unit Record Executor
IGG01910	Open - Load Executor
IGG01911	Open - IOB and Buffer Construction
IGG0199M	Open - JFCB Merge
IGG0200A	Close - Read JFCB and DSCB
IGG0200F	Close - Direct Access Routine
IGG0200G	Close - Delete Routine
IGG0200Y	Close - Direct Access Processing
IGG0200Z	Close - Second Load
IGG0201Z	Close - SAM Executor (SAM - Common sequential access methods modules)

*The last (eighth) character is a 12 and 0 punch. In EBCDIC this is ƀ (the blank character), in BCD ? (the question mark).

The IBM supplied standard list IEABLD00 is shown below.

```
SYS1.LINKLIB  IEBCOMPR,IEBGENER,IEBPTPCH,IEBUPDTE,IEHLIST,IEHMOVE,     X
              IEHPROGM,LINKEDIT,SORT
```

MVT and MFT Job Queue Formatting

In MVT and MFT configurations of the operating system, the job queue format is specified when the system is generated and may be altered during subsequent system start procedures. Formatting consists of specifying the number of queue records in a job queue logical track, reserving queue records for initiators, and reader/interpreters, and reserving queue records for job cancellation.

This chapter provides guidelines for estimating:

- The number of queue records in a job queue logical track.
- The number of queue records to be reserved for use by an initiator and reader/interpreter.
- The number of queue records to be reserved for cancellation of job initiation and running when the number of queue records reserved for initiator use is insufficient.

Reference Publications

The *IBM System/360 Operating System: System Generation* publication (Form C28-6554) describes the SCHEDULR macro instruction parameters used to initially specify job queue format.

The *IBM System/360 Operating System: Operator's Guide* publication (Form C28-6540) describes the procedure used to alter job queue format.

MVT Job Queue Formatting

In MVT and MFT operating system configurations, the basic element of the system job queue (the data set SYS1.SYSJOBQE) is a 176-byte record -- the queue record. The total number of queue records available is fixed by the space allocated to the SYS1.SYSJOBQE data set. Queue records contain the tables, control blocks, and system messages developed by the reader/interpreter and initiator control program routines -- the information used to run a job.

Lack of queue records to work with is not critical for a reader/interpreter routine. In MVT processing of the input job stream assigned to a reader/interpreter is suspended until queue records become available, at which time processing is resumed. In MFT the operator will receive a message if there is insufficient space for a reader/interpreter. He may wait for space or cancel the reader. An initiator, however, must have sufficient queue records available to complete the initiation and running of a job or the job is canceled. Because, in an MVT configuration, one or more reader/interpreters and one or more initiators may be concurrently active, steps must be taken to ensure that queue records are available to each initiator started, so that it may complete its operations. In addition queue records must be reserved for use by initiators in the event job cancellation does take place. The main function of job queue formatting is to reserve queue records for initiator use.

To format the job queue you must:

1. Designate the number of queue records to be contained in a job queue logical track. A logical track consists of a header record (20 bytes) plus the designated number of queue records. Reader/interpreters and initiators are assigned queue records in terms of logical tracks.

2. Designate the number of queue records to be reserved for use by an initiator. Each initiator is allocated this number of records. If the allocation is insufficient for the job currently being processed by the initiator, the job is canceled in MVT.

3. Designate the number of queue records to be reserved for use in case of job cancellation. All initiators that cancel use these queue records. If the allocation is insufficient, the initiator is placed in a WAIT state and a message issued.

The balance of the queue (total queue records less the reservations in items 2 and 3) is available for use by the reader/interpreters.

You specify initial values for logical track size, queue record reservation for initiators, and queue record reservation for job cancellation, in the SCHEDULR macro instruction parameters JOBQFMT, JOBQLMT, and JOBQTMT respectively. The System Generation publication describes the procedure.

There are no comprehensive, foolproof formulas for calculating values of JOBQFMT, JOBQLMT, and JOBQTMT. The values to be estimated are dependent upon the requirements and structure of the jobs to be presented to the system, the number of job steps, the number of I/O devices required, the number and type of data sets, the number of volumes, and most unpredictable, the number of system messages issued during the initiation and running of a job. The rest of this chapter provides some basic guidelines for your use in determining these values.

LOGICAL TRACK SIZE -- JOBQFMT

Logical track size -- the number of queue records in a logical track -- affects the efficient use of queue records. Reader/Interpreters and initiators are allocated queue records in terms of logical tracks. Unused queue records in a logical track are not available for use by other reader/interpreters or initiators. Therefore, an over generous logical track size specification results in wasted queue records and reduction of job queue capacity, i.e., the unused queue records, if available, could contain the required information for another job.

Logical track size affects performance to some extent. Specification of a logical track size of 10 queue records or less can result in excessive execution of the track assignment routines, etc., i.e., the "overhead" required to use very small logical track sizes impairs performance.

You may, as a starting point, wish to use the default value for JOBQFMT (12 queue records).

You may make your logical track size (or multiples of it) correspond to the physical track capacity of the device on which the job queue is resident. For example, if the IBM 2301 Drum Storage unit is to be used, 66 queue records may be contained in one physical track. You might specify, in this case, a logical track size of 22 queue records, thereby allocating 3 logical tracks to one physical track (3 x 22 = 66 queue records). The 3 logical track header records (20 bytes each) use up the remaining record.

You may wish to make your logical tracks contain the same number of queue records as are reserved for initiator use.

RESERVING INITIATOR QUEUE RECORDS -- JOBQLMT

The value you specify for JOBQLMT must be large enough for the queue entries of any job that enters the system. The following list shows the factors that affect the value of JOBQLMT:

- Number of entire generation data groups in a job.
- Number of passed data sets in a job.
- Number of devices required for passed data sets.
- Number of volumes containing the data sets in a step.
- Number of system messages issued during initiation of a step.

The sum of the queue records required for each of these items provides you with a JOBQLMT value.

When a start initiator command is issued, a check is made to see if enough free logical tracks are available to provide the required number of queue records for the initiator. If not, the system rejects the command.

Number of Generation Data Groups

Each entire generation data group (GDG) used during a job increases the number of queue records needed by an initiator. Two queue records should be reserved for every generation in excess of the first in a GDG. One queue record should be reserved for every four GDGs used in a job.

Thus, if a job uses two entire GDGs, one having 5 data sets (generations), and the other having 24 data sets, 55 queue records must be reserved -- (4+23)x2+1.

Number of Passed Data Sets

Two queue records are needed by an initiator for every three data sets passed during a job. If the number of data sets passed is not a multiple of three, queue records must be allocated as if the number of data sets passed was a multiple of three. Thus if one, two, or three data sets are passed, two queue records are allocated; if four, five, or six data sets are passed, 4 queue records are allocated, and so on.

Number of I/O Devices For Passed Data Sets

When a data set being passed requires more than ten I/O devices, one queue record is required by an initiator. This queue record accommodates 43 devices. If the number of required devices exceeds 53, a second queue record is needed. Separate calculations must be made for each data set.

Number of Volumes

An initiator requires queue records for each data set that occupies more than five volumes, and is located by a search of the catalog. (If a data set's location is specified in a DD statement, the reader routines acquire the necessary records.) One queue record is needed if the data set occupies between 6 and 20 volumes; two queue records if 21 to 35 volumes; three if 36 to 50 volumes; and so on. Separate calculations must be made for each data set.

Number of System Messages

An initiator requires queue records for system messages it issues. If you assume that each message is 80-characters in length, each queue record holds two messages. Messages from initiators are primarily device allocation, allocation recovery, and data set disposition messages.

To cover most device allocation and data set disposition situations, you should allocate two queue records for every three DD statements in a job step.

Allocation recovery messages apply to devices that are offline. You will cover most situations if you allocate queue records according to the following algorithm:

- Determine the largest number of devices of a given class that will be offline at any given time.
- Divide by two.
- Add one.

Since you will probably make this calculation for a job step, you should multiply your result by the number of steps in a large job.

System messages are the least predictable of all the variables used in calculating initiator queue record needs. The number of messages depends on the number of devices offline, the number not available, and the number required at any given time.

RESERVING QUEUE RECORDS FOR CANCELLATION -- JOBQTMT

If an initiator's queue record requirements exceed the number of queue records reserved for it, the job associated with that initiator is canceled. Queue records must be reserved for this purpose. Enough queue records must be reserved to accommodate two (or more) initiators that may be cancelling concurrently. The JOBQTMT value (like the value JOBQLMT) is unpredictable because of factors such as the installation's configuration, the size of the job being canceled, and the number of jobs that can be multiprogrammed.

The following guidelines should be used in calculating JOBQTMT:

- Number of devices used during a job.
- Number of jobs that might be concurrently canceled because of insufficient initiator queue records.
- For any system task to be started, combined JCL from its associated catalogued procedure and the START command must first be interpreted. This requires queue records, and the system allows assignment of records for this purpose whenever any logical tracks are available. During normal use of the queues, this space is always available. However, in order to insure availability of queue records for system tasks when the reserves approach the critical state, the value of JOBQTMT should be increased over the above amount by the number of records necessary to get tasks started. (This is especially true for writer and initiator tasks, since they return queue records to the system.) This amount may be estimated in a manner similar to calculating JOBQLMT, taking into consideration that each valid START command generates one input and one output queue entry. Formulas for estimating queue entry sizes are given in the _Storage Estimates_ publication.

Number of Devices

The devices currently assigned to a job are released when the job is canceled. Since messages are issued when devices are released, you should reserve a number of queue records equal to the largest number of devices assigned at any one time to a job, multiplied by two. Thus if your largest job (in terms of devices) has three steps requiring 4, 11, and 8 devices respectively, 22 queue records should be reserved.

Number of Jobs

The number of queue records reserved for cancellation must be large enough to fill the requirements of all jobs being canceled at any one time because of insufficient initiator queue records. If your estimate of initiator queue records was accurate, it is unlikely that you will have more than one job (if any) cancelling at any one time.

An initiator that runs out of queue records for cancellation is placed in the wait state and an operator message -- IEF4261 QUEUE CRITICAL -- is issued. This can result in the interlocking of all reader/interpreters, initiators, and sysout writers functioning at the moment.

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: 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. (The results of a DEVTYPE macro instruction executed before a checkpoint is taken should not be considered valid after a checkpoint restart occurs.)

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 zeroes 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 (except for the 2321 on which a 1 in bit 6 indicates the device has byte addressing). 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

	UCB Type Field (Word 1 In Hexadecimal)	Maximum Record Size (Word 2 In Decimal)	DEVTAB (Words 3, 4, and 5 In Hexadecimal)
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	00CB00141C7E922D2D010216
2321	30 00 20 05	2000	140A051407D0641010030219

Graphics Devices

1053	14 0 0 10 04	Not Applicable	
2250 (Mod 1)	31 x x 10 02	Not Applicable	
2250 (Mod 2)	32 x x 10 02	Not Applicable	
2250 (Mod 3)	33 x x 10 02	Not Applicable	
2280	30 0 0 10 05	Not Applicable	
2282	30 0 0 10 06	Not Applicable	
2260 (Mod 1)	11 x x 10 03	Not Applicable	
2260 (Mod 2)	12 x x 10 03	Not Applicable	

CI=Card Image Feature

UCS=Universal Character Set

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

xx = Special Feature (byte 2) configurations may be obtained from the System Control Blocks publication.

Communication Equipment	UCB Type Field	Record Size
1030,1050,83B3, TWX,2250, S360	51xx40YZ	Not Applicable
1060,115A,1130	52xx40YZ	Not Applicable
2780	53xx40YZ	Not Applicable
2740	54xx40YZ	Not Applicable

Y=Adapter Type (Bits 0-3)

Hex Value	Meaning
1	IBM Terminal Adapter, Type I
2	IBM Terminal Adapter, Type II
3	IBM Telegraph Adapter
4	Telegraph Adapter, Type I
5	Telegraph Adapter, Type II
6	World Trade Telegraph Adapter
7	Synchronous Adapter, Type I
8	IBM Terminal Adapter, Type III
9	Synchronous Adapter, Type II

Z=Control Unit (Bits 4-7)

Hex Value	Meaning
1	2702
2	2701
3	2703

Exceptional Returns

The following return codes are placed in register 15:

00 - request completed satisfactorily.

04 - ddname not found.

08 - invalid area address. The address of the output area either violates protection, or it is out of the range of main storage.

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 Blocks 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. To inhibit writing the JFCB back to the job queue during an OPENJ, the field JFCBTSDM should be set to X'08' prior to issuing the OPEN macro.

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 Supervisor and Data Management Macro Instructions publication. 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.)

Caution: In MVT configurations of the operating system, data set integrity provided by the job scheduler functions is lost if you change, or do not use, the DSNAME=parameter in the DD statement.

Note: The DD statement must specify at least:

- Device allocation (refer to the Job Control Language publication for methods of preventing share status).
- 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 instructionto 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 JFCBs 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 program is supplying the JFCBs for system use.

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

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 the Supervisor and Data Management 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 the Supervisor and Data Management Services and Supervisor and Data Management Macro Instructions publications.

CIRB—Create IRB for Asynchronous Exit Processing

The CIRB macro instruction is included in SYS1.MACLIB and must be included in your system at system generation time if you intend to use it. The issuing of this macro instruction causes a supervisor routine (called the exit effector routine) to create an interruption request block (IRB) if one is not already in existence for the task in question. In addition, other operands of this macro instruction may specify the building of a register save area and/or a work area to contain interruption queue elements, which are used by supervisor routines in the scheduling of the execution of user exit routines.

Name	Operation	Operand
[symbol]	CIRB	{EP=addrx}, KEY= $\left\{\begin{matrix}\underline{\text{PP}}\\ \text{SUPR}\end{matrix}\right\}$, MODE= $\left\{\begin{matrix}\underline{\text{PP}}\\ \text{SUPR}\end{matrix}\right\}$, [STAB=code,] SVAREA= $\left\{\begin{matrix}\underline{\text{NO}}\\ \text{YES}\end{matrix}\right\}$, [WKAREA=value]

EP
: specifies the entry point address of the user's asynchronous exit routine.

KEY
: specifies whether the user's asynchronous routine will operate with a CPU protection key established by the supervisory program (SUPR) or with a protection key obtained from the task control block of the task for which the macro instruction is issued (PP).

MODE
: specifies whether the user asynchronous routine will be executed in the problem program (PP) state or in a supervisory (SUPR) state.

STAB
: indicates the status condition of the interruption request block. The 'code' parameter may be either of the following:

(RE) to indicate that the IRB is reusable in its current form.

(DYN) to indicate that the storage area assigned to the IRB is to be made available (i.e., freed) for other uses when the asynchronous exit routine is completed.

SVAREA
: specifies whether a register save area (of 72 bytes) is to be obtained from the main storage assigned to the problem program. If it is, the address of this save area is placed in the IRB. The asynchronous exit routine then follows the system register saving convention of using the SAVE and RETURN macro instructions. In this manner, a generalized subroutine can be used as an asynchronous exit routine.

WKAREA
: specifies the number of double words (given as a decimal value) required for an area in which the routine issuing the macro instruction can construct interruption queue elements.

SYNCH—Synchronous Exits to Processing Program

The SYNCH macro instruction is a system macro instruction that permits control program supervisor call (SVC) routines to make synchronous exits to a processing program.

Name	Operation	Operand
[symbol]	SYNCH	{entry-point / (15)}

entry-point
: specifies the address of the entry point for the processing program that is to be given control.

  If (15) is specified, the entry-point address of the processing program must have been pre-loaded into parameter register 15 before execution of this macro instruction.

SYNCH Macro Definition

```
         MACRO
&NAME    SYNCH     &EP
         AIF       ('&EP' EQ '').E1
         AIF       ('&EP'(1,1) EQ '(').REG
&NAME    LA        15,&EP                   LOAD ENTRY POINT ADDRESS.
         AGO       .SVC
.REG     AIF       ('&EP' EQ '(15)').NAMEIT
&NAME    LR        15,&EP(1)                LOAD ENTRY POINT ADDRESS.
.SVC     SVC       12                       ISSUE SYNCH SVC
         MEXIT
.NAMEIT  ANOP
&NAME    SVC       12                       ISSUE SYNCH SVC
         MEXIT
.E1      IHBERMAC  27,405
         MEND
```

Programming Notes: In general, you use the SYNCH macro instruction when a control program in the supervisor state is to give temporary control to a processing program routine, and you expect the processing program to return control to the supervisor state. The program to which control is given must be in main storage when the macro instruction is issued. The use of this macro instruction is similar to that of the BALR instruction in that register 15 is used for the entry point address. When the processing program returns control, the supervisor state bit, the storage protection key bits, the system mask bits and the program mask bits of the program status word are restored to the settings they had before execution of the SYNCH macro instruction.

Example: As a result of an OPEN macro instruction, label processing may be carried out to a point at which a user's processing program indicates that private processing is desired (or necessary). The control program's open routine then will issue a SYNCH macro instruction giving the entry point of the subroutine required for the user's private label processing.

STAE—Specify Task Asynchronous Exit

The STAE macro instruction permits control to be returned to the user when a task is scheduled for ABEND. When issued, STAE causes a supervisor routine (called the STAE service routine) to create a STAE control block (SCB) which points to the user's STAE exit routine address.

Name	Oper-ation	Operand
[symbol]	STAE	$\begin{Bmatrix}0\\ \text{exit address}\end{Bmatrix},\begin{Bmatrix}\text{OV}\\ \underline{\text{CT}}\end{Bmatrix}\left[\text{,PARAM=list address}\right]\left[\text{,XCTL=}\begin{Bmatrix}\text{YES}\\ \underline{\text{NO}}\end{Bmatrix}\right]$

exit address
: specifies the address of a STAE exit routine to be entered if the task issuing this macro instruction terminates abnormally. If 0 is specified, the last SCB created is cancelled and the previously created SCB becomes current. The address may be loaded into one of the general registers (r_1) 2 through 12.

OV
: indicates that the parameters passed in this STAE macro instruction are to overlay the data currently in the SCB.

CT
: indicates the creation of a new STAE environment; i.e., creates and initializes an SCB with the parameters specified.

PARAM=
: specifies the address of a parameter list containing data to be used by the STAE exit routine when it is scheduled for execution. The address may be loaded into one of the general registers (r_2) 2 through 12.

XCTL=YES
: indicates that the STAE macro instruction will not be cancelled if an XCTL macro instruction is issued.

XCTL=NO
: indicates that the STAE macro instruction will be cancelled if an XCTL is issued.

Programming Notes: When control is returned to the user after the STAE macro instruction has been issued, register 15 contains one of the following return codes:

Code	Meaning
00	An SCB is successfully created, overlaid, or cancelled.
04	Storage for an SCB is not available.
08	The user is attempting to cancel or overlay a non-existent SCB, or is issuing a STAE in his STAE exit routine.
0C	The exit routine or parameter list address is invalid.
10	The user is attempting to cancel or overlay an SCB not associated with his level of control.

task issuing this macro instruction terminates abnormally. If 0 is specified, the most recent STAE request is canceled. The address may be loaded into one of the general registers 2 through 12.

OV
: indicates that the parameters passed in this STAE macro instruction are to overlay the data contained in the previous STAE request.

CT
: indicates the creation of a new STAE request. If neither OV or CT is specified, CT is assumed.

PARAM=
: specifies the address of a parameter list containing data to be used by the STAE exit routine when it is scheduled for execution. The address may be loaded into one of the general registers 2 through 12.

XCTL=YES
: indicates that the STAE macro instruction will not be canceled if an XCTL macro instruction is issued.

XCTL=NO
: indicates that the STAE macro instruction will be canceled if an XCTL is issued by this program. If neither XCTL=YES or XCTL=NO is coded, XCTL=NO is assumed.

PURGE=

QUIESCE
: indicates that all outstanding requests for input/output (I/O) operations will be saved when the STAE exit is taken. At the end of the STAE exit routine, the user can code a retry routine to handle the outstanding I/O requests. (See the description of the STAE macro instruction in _IBM System/360 Operating System: System Programmer's Guide_, for a description of the STAE retry routine.) If the PURGE operand is not specified, QUIESCE is assumed. If I/O cannot be quiesced, then I/O is halted (see PURGE=HALT).

HALT
: indicates that all outstanding requests for input/output operations will not be saved when the STAE exit is taken.

NONE
: indicates that input/output processing is allowed to continue normally when the STAE exit is taken.

Note: If PURGE=NONE is specified and the ABEND was originally scheduled because of an error in input/output processing, an ABEND recursion will develop when an input/output interruption occurs, even if the exit routine is in progress. Thus, it will appear that the exit routine failed when in reality input/output processing was the cause of the failure.

ASYNCH=

YES
: indicates that asynchronous interrupt processing is allowed to interrupt the processing done by the STAE exit routine. ASYNCH=YES must be coded if:

- Any supervisor services that require asynchronous interruptions to complete their normal processing are going to be requested by the STAE exit routine.

STAE -- Specify Task Abnormal Exit

The STAE macro instruction enables the user to intercept a scheduled ABEND and to have control returned to him at a specified exit routine address. The STAE macro instruction operates in both problem program and supervisor modes.

The STAE macro instruction creates a STAE control block (SCB) which represents a STAE environment that remains in effect during the execution of the program that issued the STAE. When a RETURN, XCTL, or SVC3 is issued, the system automatically cancels the STAE environment for that program, unless XCTL=YES is coded in the STAE macro instruction. If XCTL=YES is coded and an XCTL macro instruction is issued, the STAE environment remains in effect for the program that receives control as a result of the XCTL macro instruction.

Note that issuing a LINK macro instruction does not cancel the STAE environment and that the user is responsible for canceling the STAE environment if his program does not exit via a RETURN, XCTL, or SVC3. The user cannot cancel or overlay a STAE control block not created by his own program. For more information on the STAE macro instruction, see the publication IBM System/360 Operating System: Supervisor Services.

Within the STAE exit routine, the user may perform pre-termination functions or diagnose an error. Upon completion of STAE exit routine processing, the user can either allow abnormal termination processing to continue for the task or request that a STAE retry routine be scheduled which would circumvent the scheduled ABEND. For further explanation of the facility for scheduling a STAE retry routine, see the publication IBM System/360 Operating System: System Programmer's Guide.

The STAE exit routine cannot contain a STAE or an ATTACH macro instruction. When a STAE retry routine is not to be scheduled, the STAE exit routine should return with a code of zero in register 15.

The STAE macro instruction is written as follows (see Section III for the list and execute forms):

[symbol]	STAE	$\left\{\begin{matrix}0\\ \text{exit address}\end{matrix}\right\}\left[\begin{matrix}\text{,OV}\\ \text{,CT}\end{matrix}\right]$ [,PARAM=list address] $\left[\text{,XCTL=}\left\{\begin{matrix}\text{YES}\\ \text{NO}\end{matrix}\right\}\right]\left[\text{,PURGE=}\left\{\begin{matrix}\text{QUIESCE}\\ \text{HALT}\\ \text{NONE}\end{matrix}\right\}\right]$ $\left[\text{,ASYNCH=}\left\{\begin{matrix}\text{YES}\\ \text{NO}\end{matrix}\right\}\right]$

exit address
    specifies the address of a STAE exit routine to be entered if the

----

[] indicates optional name or operand; select one from vertical stack within {}; select one or none from vertical stack within [].

- PURGE=QUIESCE is specified for any access method that requires asynchronous interruptions to complete norma input/output processing.

- PURGE=NONE is specified and the CHECK macro instructi is issued in the STAE exit routine for any access method that requires asynchronous interruptions to complete normal input/output processing.

Note: If ASYNCH=YES is specified and the ABEND was original scheduled because of an error in asynchronous exit handling, an ABEND recursion will develop when an asynchronous interruption occurs. Thus, it will appear that the exit routine failed when in reality asynchronous exit handling wa the cause of the failure.

NO

indicates that asynchronous interrupt processing is not allowed to interrupt the processing done by the STAE exit routine. If the ASYNCH operand is not specified, NO is assumed.

ISAM Notes: If ISAM is being used and PURGE=HALT is specified or PURGE=QUIESCE is specified but I/O is not restored:

- Only the input/output event on which the purge is done will be posted. Subsequent event control blocks (ECBs) will not be posted.

- The ISAM Check routine will treat purged I/O as normal I/O.

- Part of the data set may be destroyed if the data set is being updated or added to when the failure occurred.

Control is returned to the instruction following the STAE macro instruction. When control is returned, register 15 contains one of th following return codes:

Hexadecimal Code	Meaning
00	Indicates successful completion of creating, overlaying, or canceling a STAE request.
04	Indicates that STAE was unable to obtain storage fc the STAE request.
08	Indicates that the user was attempting to cancel o overlay a non-existent STAE request, or that the user issued an STAE in his STAE exit routine.
0C	Indicates that the exit routine or parameter list address was invalid.
10	Indicates that the user was attempting to cancel o overlay a STAE request of another user.

When a program with an active STAE environment encounters an ABEND situation, control is returned to the user through the ABEND/STAE interface routine at the STAE exit routine address. The register contents are as follows:

* Register 0:

Code	Indication
0	Active I/O at the time of the ABEND was quiesced and is restorable.
4	Active I/O at the time of the ABEND was halted and is not restorable.
8	No I/O was active at the time of the ABEND.

* Register 1: Address of a 104-byte work area:

0	STAE exit routine parameter list addr or 0	ABEND completion code
8	PSW at time of ABEND	
16	Last P/P PSW before ABEND	
24	Registers 0-15 at time of ABEND (64 bytes)	

If problem program issued STAE:

88	Name of ABENDing program or 0	
96	Entry point addr of ABENDing program	0

If supervisor program issued STAE:

88	Request Block addr of ABENDing program	0
96	0	

* <u>Registers 2-12</u>: Unpredictable.
* <u>Register 13</u>: Address of a supervisor save area.
* <u>Register 14</u>: Address of an SVC 3 instruction.
* <u>Register 15</u>: Address of the STAE exit routine.

Registers 13 and 14, if used by the STAE exit routine, must be saved and restored prior to returning to the calling program. Standard subroutine linkage conventions are employed.

If storage was not available for the work area, the register contents upon entry to the STAE exit routine are as follows:

* Register 0: 12.
* Register 1: ABEND completion code as it appears in the TCBCMP field.
* Register 2: Address of STAE exit parameter list.

The STAE exit routine may contain an ABEND, but must not contain either a STAE or an ATTACH macro instruction. At the time the ABEND is scheduled, the STAE exit routine must be resident as part of the program issuing STAE, or brought into storage via the LOAD macro instruction.

The STAE exit routine may perform pre-termination functions, attempt to diagnose the error, or attempt to correct the error by specifying that a retry routine is to be scheduled. If a retry routine is not specified, normal ABEND processing continues by returning control to the ABEND/STAE interface routine with a 0 in register 15. To schedule a retry routine with a purge of the RB chain (not including the STAE user's RB), the STAE exit routine must return to the ABEND/STAE interface routine with a 4 in register 15. Register 0 must contain the address of the retry routine, and register 1, the address of the work area. (The work area is the same as that passed to the exit routine except that the first word may now contain user data.)

In supervisor mode, you may want the failing task to remain in its present status and not be reestablished. A retry routine may be scheduled without a purge of the RB chain by returning to the ABEND/STAE interface routine with an 8 in register 15 and registers 0 and 1 initialized as described above.

Like the STAE exit routine, the STAE retry routine must be in storage when the exit routine determines that retry is to be attempted. If not already resident within your program, the retry routine may be brought into storage via the LOAD macro instruction by either the user's program or exit routine.

Upon entry to the STAE retry routine, register contents are as follows:

• Register 0:	0.
• Register 1:	Address of the work area, as previously described, except that word 2 now contains the address of the first I/O Block and word 26 now contains the address of the I/O restore chain.
• Registers 2-13:	Unpredictable.
• Register 14:	Address of an SVC 3 instruction.
• Register 15:	Address of the STAE retry routine.

The retry routine should free the 104 bytes of storage occupied by the work area when it is no longer needed.

Again, if the ABEND/STAE interface routine was not able to obtain storage for the work area, register 0 contains a 12; register 1, the ABEND completion code upon entry to the STAE retry routine; and register 2, the address of the first I/O Block on the restore chain, or 0 if I/O is not restorable.

If continued protection against ABEND is desired within the STAE retry routine, a STAE macro instruction may be issued.

Note: If the program using the STAE macro instruction terminates via the EXIT macro instruction, the EXIT routine cancels all SCBs related to the terminating program. If the program terminates via the XCTL macro instruction, the EXIT routine cancels all SCBs related to the terminating program except those SCBs that were created with the XCTL=YES option. If the program terminates by any other means, the terminating program must reinstate the previous STAE environment by canceling all SCBs related to the terminating program.

Writing System Output Writer Routines

This chapter provides guidelines for writing your own output writer routines for use in an MVT or MFT configuration of the operating system.

Reference Publications

IBM System/360 Operating System: Supervisor and Data Management Macro Instructions, Form C28-6647

IBM System/360 Operating System: Supervisor and Data Management Services, Form C28-6646

Writing System Output Writer Routines

When a job is executing, system messages and data sets specifying the SYSOUT parameter (e.g., in the DD statement) are recorded on direct access devices. When the job completes, entries are made in system output class queues that represent the data sets and messages directed to the output classes. Later system output writers remove these entries from the queues and process the data they represent. Processing consists of transcribing system messages to the output device and calling a data set writer routine for each data set encountered. The data set writer routine used for a data set may be specified by name in a DD statement, otherwise, a standard IBM-supplied writer routine is used. The standard routine transcribes the data set to the specified output device, making only those data format and control character transformations required to conform to the attributes specified for the output data set.

The following material describes how you may write a non-standard data set writer routine.

Output Writer Functions

Before writing or modifying an output writer routine, you should be familiar with the functions performed by the standard data set writer for Operating System/360. (For the remainder of this chapter, the Operating System/360 data set writer is referred to as the standard writer.) In general, these functions include opening the data set (referred to as an input data set) that contains the processed information, obtaining the records of the data set, making any necessary transformations in record format or control character attributes, and placing these (possibly transformed) records in the output data set, which appears on a specified output device. The standard writer also must close the input data set and restore system conditions to the state they were in before the writer routine was invoked.

Conventions to be Followed

To use your own output writer routine, you must specify the name of your routine as a parameter in the SYSOUT operand of a DD statement (see the Job Control Language publication). Your routine must be in the system library (SYS1.LINKLIB). A writer routine is not limited in size except that size may influence the region requirements of the system output writer (see the Storage Estimates publication).

In MVT your routine is attached (via the ATTACH macro instruction) when a data set requiring the routine is to be processed. The standard linkage conventions for attaching are used. Any storage required for work areas and tables should be obtained by the GETMAIN macro instruction and released by the FREEMAIN macro instruction. Your output writer routines must be reentrable.

When your routine is finished, it must return control to the standard writer by using the RETURN macro instruction.

After job management routines perform initialization requirements and open the output data set into which your writer routine will put records, control is given to your routine via the ATTACH macro instruction. At this time, general registers 1 and 13 contain information that your program must use. Register 1 contains the storage address of a 12-byte list. Table 2 describes the information in this parameter list.

Table 2. Parameter List Referred to by Register 1.

Byte 0	Output Device Indicator. Bit 0 (High-order bit): If this bit is on (set to 1), the output unit is a 1442 punch. Bit 1 If this bit is on, the output unit is either a punch or a tape with a punch as the ultimate destination. Bit 2 If this bit is on, the output unit is either a printer or a punch. Bits 3 - 7 No significant information.
Bytes 1-3	Not used (i.e., do not contain information significant to data set writers, but must be left intact.)
Byte 4-7	This word contains the address of the data control block (DCB) for the opened output data set to be referred to by the writer.
Bytes 8-11	This word contains the DCB address for the input data set from which your writer will obtain logical records. (At the time this 12-byte parameter list is given to your writer, the input data set is not open.)

The switches indicated by the three high-order bit settings in byte 0 should be used to translate control character information from the input data set records to the form required by the output data set records. Based on the indications given in Table 2, the high-order three bits of byte 0 signify the type of output device as follows:

111.....	1442 punch unit
011.....	2520 punch unit or 2540 punch unit
001.....	1403 printer, 1404 printer, or 1443 printer unit
010.....	tape unit with ultimate punch destination
000.....	tape unit with ultimate printer destination

When your writer gets control, it must preserve the contents of register 0 through 12, and 14. Register 13 contains the address of a standard register save area where you are to save the contents of these registers. You can save the contents of register 13 by using the SAVE macro instruction.

An output writer routine must issue an OPEN macro instruction to open the desired input data set residing on a direct access device as a result of the previous execution of a processing program. (Note: The output data set used by a writer is opened by a job management routine before control is given to the writer. This output data set must be given records by a PUT macro instruction operating in the 'locate' mode. The Supervisor and Data Management Macro Instructions publication describes this macro instruction.)

If the processing program that produces a given data set (to be used as an input data set by a writer) did not open the data set, the data set contains no records, and the DCBBLKSI and DCBBUFL fields of the input DCB contains zero. The DCBBLKSI field may also be zero even if the data set does contain records -- if the processing program did not put the block size value for the input data set in the DCB. If both these DCB fields are zero, a value (the standard writer uses the decimal value 18) is inserted in the DCBBLKSI field to permit the open routine to continue. The standard writer does this via a routine pointed to by an entry in the EXLIST parameter of the DCB. Since there is no data set, nothing is put on the output device. Your data set writer must provide a SYNAD routine to process errors associated with the output as well as the input data set.

Before the OPEN macro instruction is issued, the DCBD macro instruction can be used to symbolically define the fields of the DCB, and the EXLIST and/or SYNAD routine addresses can be inserted. Other than SYNAD, no modifications can be made to the output DCB.

After your routine finishes writing the output data set, it must close the input data set and return using the RETURN macro instruction. A return code must be placed in register 15. This code should indicate that an unrecoverable output error either has occurred (code of 8) or has not occurred (code of 0).

General Processing Performed by Standard Output Writer

This section provides a general description of the procedures followed by the standard writer. (See Figure 6.) If you write your own writer routine, you may wish to delete, modify, or add to some of these procedures, depending on the characteristics of your data set(s). However, your procedures must be consistent with operating system conventions.

SAVING REGISTER CONTENTS: Upon entering the writer program, your program must save the contents of the general registers, as previously discussed.

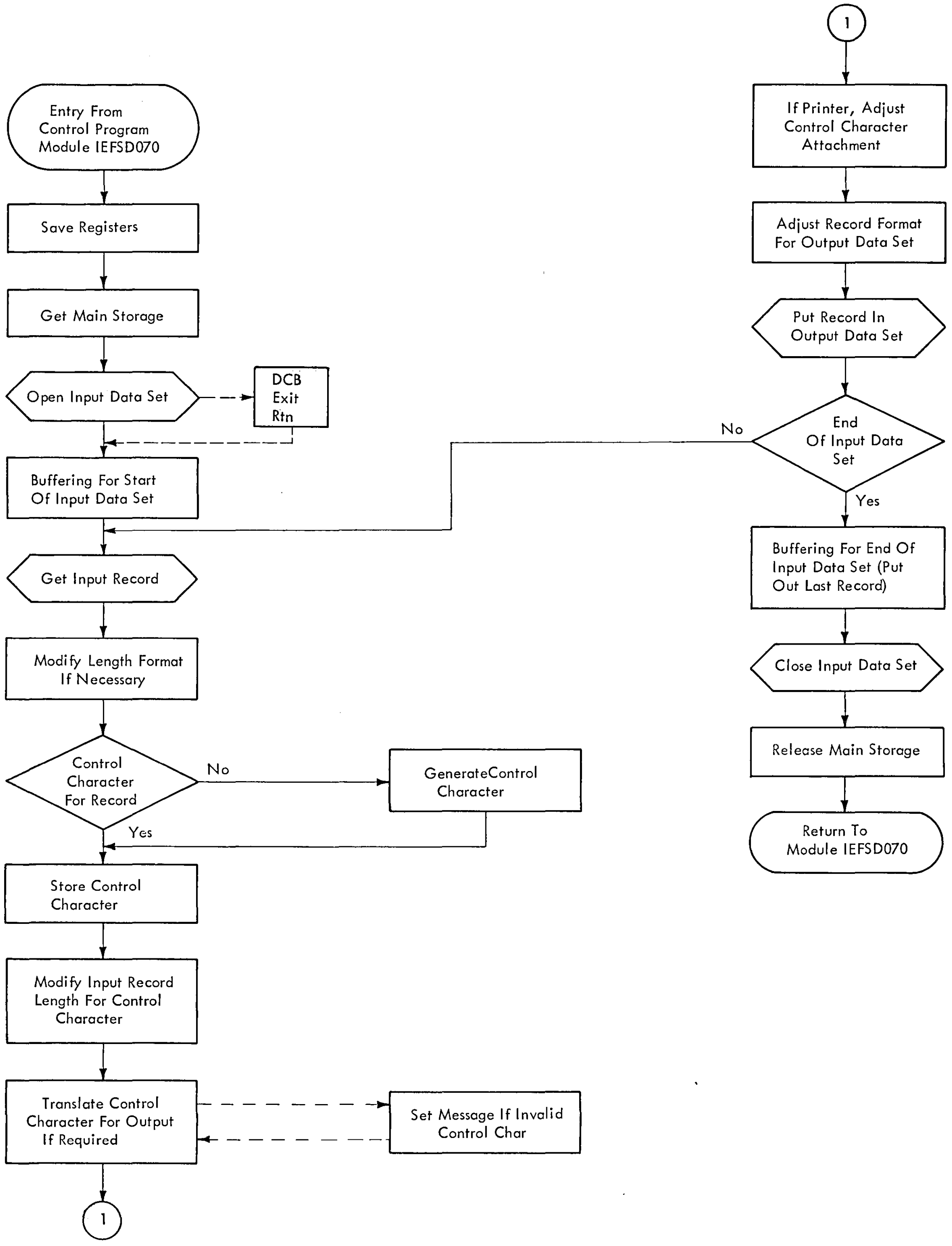
OBTAINING MAIN STORAGE FOR WORK AREAS: In this work area, switches are established, record lengths and control characters are saved, and space is reserved for other uses. You should obtain storage by a GETMAIN macro instruction.

PROCESSING INPUT DATA SET(S): To process a data set, the writer must get each record individually from the input data set, transform (if necessary) the record format and the control characters associated with the record in accordance with the output data set requirements, and put the record in the output data set. Data set processing by the standard writer can be considered in three aspects.

1. The first consideration is what must be done before actually obtaining records from an input data set. If the output device is a printer, provision must be made to handle the two forms of record control character that may accompany a record in an output data set. The printer is designed so that if the output data set records contain machine control characters, a record (line) is printed before the effect of its control character is considered. However, if USASI control characters are used in the output data set records, the control character effect is considered before the printer prints a record. See Appendix A.

   Thus, if all the input data sets do not have the same type of control characters, it may be desirable to avoid overprinting of the last line of one data set with the first line of the following data set. If the records of the input data set have machine control characters (mcc) and the output data set records are to have USASI control characters (acc), the standard writer produces a control character that indicates one line should be skipped before printing the first line of output data.

   If the input data set records have acc and the output data set records are to be written with mcc, the standard writer prints a line of blanks before printing the first actual output data set record. Following this line of blanks, a one-line space is generated before the first output record is printed.
   The preceding 'printer initialization' procedure (or a similar one based on the characteristics of your data sets) is recommended.



1
Entry From Control Program Module IEFSD070
Save Registers
Get Main Storage
Open Input Data Set
DCB Exit Rtn
Buffering For Start Of Input Data Set
Get Input Record
Modify Length Format If Necessary
Control Character For Record
No
Yes
GenerateControl Character
Store Control Character
Modify Input Record Length For Control Character
Translate Control Character For Output If Required
Set Message If Invalid Control Char
1
If Printer, Adjust Control Character Attachment
Adjust Record Format For Output Data Set
Put Record In Output Data Set
End Of Input Data Set
No
Yes
Buffering For End Of Input Data Set (Put Out Last Record)
Close Input Data Set
Release Main Storage
Return To Module IEFSD070

Figure 6. General Logic of Standard Output Writer

2. After an input data set is properly opened and any necessary printer initialization completed, the writer obtains records from the input data set. The locate mode of the GET macro instruction is used. As each record is obtained, its format and control character must be adjusted, if necessary, to agree with that required for output.

Note: Check the MACREF field of the input data set DCB to see if GET in locate mode can be used. If not the MACREF field must be overridden.

Since the output data set is previously opened by another routine (job management), a writer routine must adhere to the established conventions. The data set is opened to receive records from the PUT macro instruction operating in the locate mode. For fixed-length record output, the length of the records in the output data set is obtained from the DCBLRECL field of the DCB. If an input record length is greater than the length specified for the records of the output data set, the standard writer truncates the necessary right-hand bytes of the input record. If the input record length is smaller than the output record length, the standard writer left-justifies the input record and adds blanks on the right end to give the correct length.

When the output record length is variable and the input record length is fixed, the standard writer constructs each output record by adding control character information (if necessary) and variable record control information to the output record. The record control information is four bytes long and the control character information is one byte long. Both additions are made to the left end of the record. If the output record is not at least 18 bytes long, it is further modified by padding bytes (blanks) added to the right end of the record. If the output record length does not agree with the length of the output buffer, the standard writer makes the proper adjustment.

3. The third aspect to consider is an end-of-input data set routine. The standard writer handles output to either a card punch unit or a printer unit, as required. Output to an intermediate device such as a tape unit is considered in light of the ultimate destination (e.g., punch or printer). If proper consideration is not given, all records from a given data set may not be available on the output device until the output of records from the next data set is started or until the output data set is closed. When the output data set is closed, the standard writer automatically puts out the last record of its last input data set.

Punch Output: Normally, when the standard writer is using a card punch as the output device, the last three output records are not in the collection pockets of the punch when the input data set is closed. To put out these three records with the rest of the data set and with no intervening pauses, the writer provides for three blank records following the actual data set records.

Printer Output: When the standard writer uses a printer as an output device, the last record of the input data set is not normally put in the output data set when the input data set is closed. To force out this last record, the writer generates a blank record that follows the last record of the actual data set.

The problem of overprinting the last line of one data set by the first line of the following data set must also be considered. Depending on the combination of input record control character and required output record control character, a line of blanks and a spacing control character may be used either individually or in combination to preclude overprinting. (Note: If overprinting is desired for some reason, control characters in the data set records themselves may be used to override the effect (but not the action) of the previously described solutions to overprinting.)

CLOSING INPUT DATA SET(s): After the standard writer finishes putting out the records of an input data set, it closes the data set before returning control to the system output writer. You must close all input data sets.

RELEASING MAIN STORAGE: The storage and buffer areas obtained for the writer must be released to the system before the writer relinquishes control. The FREEMAIN macro instruction should be used for this.

RESTORING REGISTER CONTENTS: The original contents of general registers 0 through 12, and 14 must be restored. The RETURN macro instruction is used for this. To inform the operating system of the results of the processing done by the writer, a return code is placed in general register 15 before control is returned. If the writer routine terminates because of an unrecoverable error on the output data set, the return code is 8; otherwise, the return code is 0. Unrecoverable input errors must be handled by the data set writer.

Appendix A: Control Character Transformations

To help determine what you can do with a writer routine, the control character transformation features of the standard writer are described.

Effectively there are nine control character combinations that can occur between input data set records and output data set records. Both data sets may have records whose control characters are either USASI type (acc) or machine type (mcc), or the records may not contain any control characters. However, within any given data set, the records all must contain the same form of control character. The standard writer has procedures to handle control character transformations for both an output to a card punch unit and an output to a printer unit.

Card Punch Unit

If an input data set record does not have a control character, the standard writer produces one that indicates output into pocket 1 of the punch. If the output unit is a tape unit and the ultimate destination is a punch unit, the standard writer assumes that the punch unit is either a 2540 or a 2520 unit and sets a control character accordingly. The standard writer translation of punch-type control characters is given in Table 3. In this table, the first three columns of figures are machine control character codes, and the right hand column of figures represent USASI control character codes. Each record that requires a control character has one of these 8-bit codes attached to it. Input records whose control characters are mcc and are shown in horizontal rows 1, 2, 5, and 6 are given the acc code of 'V' if they are placed in an output data set that has acc. An mcc given in rows 3 or 4 is changed to an acc code of 'W'. However, if translation is from an acc input to an mcc output, the standard writer translates the control character into the appropriate mcc on the same horizontal row.

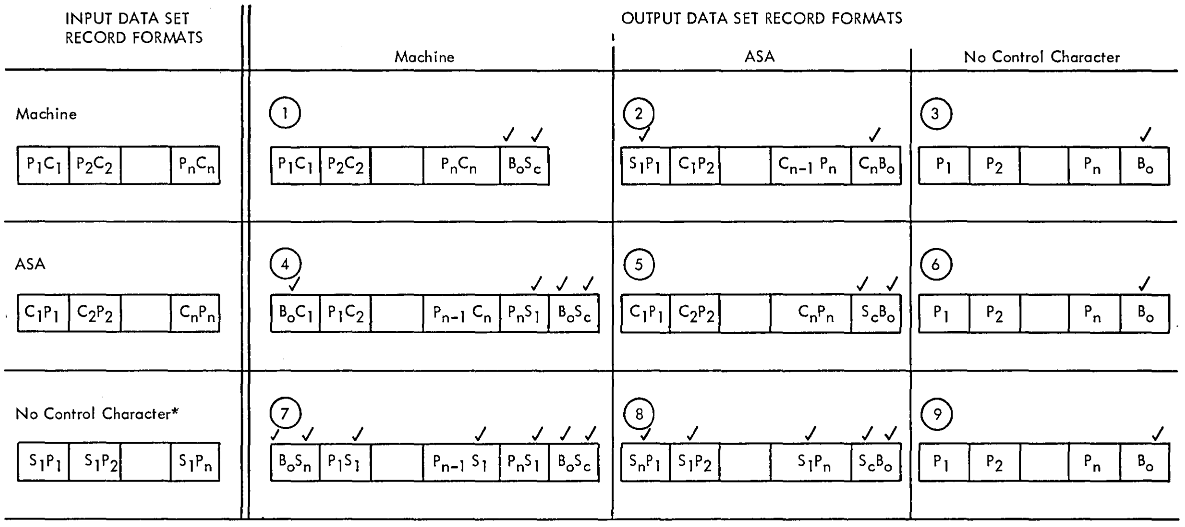
Table 3. Control Character Translation for Punch Unit Output

Stacker Unit	Machine Control Characters			USASI Control Characters
	2540 Punch	2520 Punch	1442 Punch	
1. P1	00000001	00000001	10000001	11100101 (V)
2. P1 Column Binary	00100001	00100001	10100001	
3. P2	01000001	01000001	11000001	11100110 (W)
4. P2 Column Binary	01100001	01100001	11100001	
5. RP3	10000001			
6. RP3 Column Binary	10100001			

Printer Unit

When the output unit is a printer, the standard writer prevents overprinting between data sets. If the successive data sets contain records with the same type of control character, there is no overprinting problem. If the control characters vary from one data set to the next, the standard writer solutions are applications of the technique illustrated by Figure 7. In this figure, the possible forms of the input record control characters are given in the left hand column. The three right hand columns (containing cases 1-9) represents the possible forms of the output record control characters. Within each of the 12 main sections of the figure is shown a symbolic representation of a data set whose records possess the indicated form of control character. Each record consists of a print line representation and a control character representation (where appropriate). For records with acc, the control character is shown preceding the print line, since the effect of the control character occurs before the line is printed. For records with mcc, the converse is shown. An input record with no control character is treated as if it had an acc. Because of this variance in the printer's mechanical action, whenever there is a control character transformation, the standard writer places a transformed control character with an output data set record other than the record to which the character was attached in the input data set.

In Figure 7, case 1 and 5 represent situations in which there is the same type of control character in the output as there is in the input. Thus, for records 1 through n, there is no change in the record format. However, there is a provision to allow for the possibility that two consecutive input data sets may have different control characters. In this case, a minimum separation between the data sets as they appear in the output data set is provided as indicated by the printing of blanks and suppressing the spacing of the printer to allow another control character to take effect. The 'extra' record (S_cB_o or B_oS_c) provides the more important function of forcing out the last record of the current data set before the writer's processing of that data set is done.



INPUT DATA SET RECORD FORMATS
OUTPUT DATA SET RECORD FORMATS
Machine
ASA
No Control Character
Machine
P_1C_1 P_2C_2 P_nC_n
1
P_1C_1 P_2C_2 P_nC_n B_oS_c
2
S_1P_1 C_1P_2 $C_{n-1}P_n$ C_nB_o
3
P_1 P_2 P_n B_o
ASA
C_1P_1 C_2P_2 C_nP_n
4
B_oC_1 P_1C_2 $P_{n-1}C_n$ P_nS_1 B_oS_c
5
C_1P_1 C_2P_2 C_nP_n S_cB_o
6
P_1 P_2 P_n B_o
No Control Character*
S_1P_1 S_1P_2 S_1P_n
7
B_oS_n P_1S_1 $P_{n-1}S_1$ P_nS_1 B_oS_c
8
S_nP_1 S_1P_2 S_1P_n S_cB_o
9
P_1 P_2 P_n B_o

✓ = Writer generated.
* = No control character on input causes the standard writer to generate an ASA control character as indicated.
B_o = A print line of blanks.
C_1-C_n = Control characters of records 1-N of a given data set.
P_1-P_n = Print lines of a given data set.
S_1 = A control character causing a 1-line space.
S_c = A control character causing spacing to be suppressed.
S_n = A control character causing a skip to channel 1.

Figure 7. Symbolic Representation of Record Formats

In cases 2 and 4 of Figure 7, the output data set records have different control characters than the input data set records. Case 2 shows that the standard writer generates a 1-line space control character to precede the first print line of the output. When the output is written, each control character of an input record is then attached to the next record. The last input record control character (C_n) is attached to a line of blanks (B_o). In case 4, the first input record control character is attached to a line of blanks, and each of the other control characters is attached to a preceding record, as indicated. The last input record (P_n) has a writer-generated space 1-line control character attached to it before the buffering and forcing record (B_oS_c) generated by the writer is put out.

Cases 7 and 8 show that the standard writer first generates a 'skip to channel 1' control character and then generates '1 line space' and then generates '1 line space' control characters for all but the last control character. The last control character is the space suppression character shown as part of the buffering or forcing record generated.

Cases 3, 6, and 9 show that if no control characters are required in the output data set, the records are printed consecutively and a line of blanks generated by the writer is printed after the final record in a data set. Any control character appearing in the input data set are dropped in the output data set.

Notice that in all cases involving control characters in the output data set, the standard writer allows for (1) an output record to force the printing of the last record of an input data set and (2) a means of minimum buffering between data sets by using generated control characters and print lines in conjunction with the actual data set control characters.

The standard writer translation of printer-type control characters is given in Table 4. In this table, the type of action indicated is given in the left-hand column. The middle column and the right-hand column show, respectively, the bit settings of the control character byte for machine type and USASI type control characters corresponding to the entries in the left-hand column. A control character transformation is effected by changing the bit-configuration of the control character byte as indicated in the table.

Table 4. Control Character Translation for Printer Unit Output.

Action Desired	Machine Type Control (1403, 1404, 1443 Printers)	USASI Type Control
Write space 0	00000001	01001110
Write space 1	00001001	01000000
Write space 2	00010001	11110000
Write space 3	00011001	01100000
Write skip to channel 1	10001001	11110001
Write skip to channel 2	10010001	11110010
Write skip to channel 3	10011001	11110011
Write skip to channel 4	10100001	11110100
Write skip to channel 5	10101001	11110101
Write skip to channel 6	10110001	11110110
Write skip to channel 7	10111001	11110111
Write skip to channel 8	11000001	11111000
Write skip to channel 9	11001001	11111001
Write skip to channel 10	11010001	11000001
Write skip to channel 11	11011001	11000010
Write skip to channel 12	11100001	11000011

When machine control characters are used which indicate spacing or skipping without writing (bit 6 set to 1, e.g., write and space 0-00000011) the standard writer generates the indicated USASI control character and also generates a blank record of the proper type and length.

The Output Separator Function of the System Output Writer

In the MVT, PCP, and MFT operating system configurations, the system output writer can use the output separator facility to write separation records prior to writing the output of each job. These separation records make it easy to identify and separate the various job outputs that are written contiguously on the same printer or card punch device.

This chapter describes the output separator that is supplied by IBM, and tells how to write your own. A separate section describes the differences between separators for MVT configurations and PCP, MFT configurations. 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: Assembler Language publication (Form C28-6514) contains the information necessary to code programs in the assembler language.

The IBM System/360 Operating System: Supervisor and Data Management Services publication (Form C28-6646) describes the queued sequential access method (QSAM) used by the system output writer, and discusses program linkage conventions.

The IBM System/360 Operating System: Supervisor and Data Management Macro Instructions publication (Form C28-6647) describes the system macro instructions that can be used in programs coded in the assembler language.

The Output Separator Function of the System Output Writer

In the MVT and MFT operating system configurations, problem programs write their output on a direct access volume. Later, a system output writer transcribes the data from the direct access volume to the output unit (printer, punch, or magnetic tape device). As a result, the output of more than one job may be written contiguously on the same output unit.

The output separator facility of the operating system provides a means of identifying and separating the output of various jobs processed by the same output unit. To do this, the separator writes separation records to the system output data set prior to the writing of each job's output.

You can use the output separator that is supplied by IBM, or you can create and use your own output separator programs.

Using an Output Separator

The output separator function operates under control of the system output writer. To use the function, the separator program must reside in the link library (SYS1.LINKLIB), and its name must be included as a parameter in the output writer's cataloged procedure (the second part of the PARM field in the EXEC statement). Cataloged procedures for the output writer are fully described in another chapter of this publication.

The name of a separator program is <u>not</u> specified in the IBM-supplied cataloged procedure for the output writer, and <u>cannot</u> be added by the START command at execution time. In other words, if you wish to use the separator function, you must add the parameter to the output writer cataloged procedure. Also, you must have a different output writer cataloged procedure for each separator program to be used.

Functions of the IBM Output Separator

The IBM-supplied output separator resides in the link library (SYS1.LINKLIB). When its name, IEFSD094, is specified as a parameter in the output writer cataloged procedure, the output writer uses it to separate job output. The type of separation provided by the separator depends on whether the output is punch-destined or printer-destined.

<u>Punch-Destined Output</u>

For punch-destined output, the IBM-supplied separator provides three specially punched cards (deposited in stacker 1) prior to the punched card output of each job. Each of these separator cards is punched in the following format:

```
Columns 1 to 35   --  blanks
Columns 36 to 43  --  jobname
Columns 44 to 45  --  blanks
Column  46        --  output classname
Columns 47 to 80  --  blanks
```

Printer-Destined Output

For printer-destined output, the IBM-supplied separator provides three specially printed pages prior to printing the output of each job. Each of these three separator pages is printed in the following format:

- Beginning at the channel 1 location (normally near the top of the page), the jobname is printed in block character format over 12 consecutive lines. The first block character of the 8-character jobname begins in column 11. Each block character is separated by 2 blank columns.
- The next 2 lines are blank.
- The output classname is printed in block character format covering the next 12 lines. This is a 1-character name, and the block character begins in column 55.
- The remaining lines to the bottom of the page are blank.

In addition to the above, a full line of asterisks(*) is printed twice (overprinted) across the folds of the paper. These lines are printed on the fold preceding each of the three separator pages, and on the fold following the third page. This feature provides easy separation of job output in a stack of printed pages.

For printer-destined output with the IBM-supplied separator, you must include a channel 9 punch on the carriage control tape for the printer, in addition to the channel 1 punch. The channel 9 punch controls the location of the line of asterisks and should correspond to the bottom of the page. To print the line of asterisks on the fold of the pages, you must also offset the printer registration.

Creating an Output Separator Program

You can write your own output separator program by using the information provided by the system output writer and by conforming to the requirements explained below. Your separator program, when added to the link library (SYS1.LINKLIB), is invoked by specifying its name as a parameter in the EXEC statement of the output writer cataloged procedure.

Parameter List

The output writer provides your separator program with a 4-word parameter list of needed information. When your program receives control, register 1 contains the address of a 4-word parameter list, and the parameter list contains the following:

Bytes 0-3 --	In this word, byte 0 contains switches that indicate the type of output unit, and bytes 1-3 are reserved for future use.
Bytes 4-7 --	This word is the address of the output DCB (data control block).
Bytes 8-11 --	This word is the address of an 8-character field containing the jobname.
Bytes 12-15 --	This word is the address of a 1-character field containing the output classname.

In the parameter list, the three high-order bits of byte 0 are switches that your separator program uses to determine the type of output unit. The first bit to the left is set to 1 if the output unit is a 1442 punch device. The second bit is set to 1 if the output unit is a punch device or a tape device with punch-destined output. The third bit is set to 1 if the output unit is a printer or punch device. The resulting bit combinations indicate the following:

111.	1442 punch device
011.	2520 or 2540 punch device
001.	1403, 1404, or 1443 printer device
010.	tape device with punch-destined output
000.	tape device with printer-destined output

The parameter list also points to the DCB for the output data set. This DCB is established for the queued sequential access method (QSAM), and is already open when your separator program receives control.

The address of the jobname and the address of the output classname are provided in the parameter list so that this information may be used in the separation records written by your separator program.

Programming Considerations

Your separator program, if specified in the output writer cataloged procedure, is brought in by a LINK macro instruction issued from module IEFSD078 of the output writer. Your separator program can be any size, but a program over 8K may affect the region requirement of the output writer.

CAUTION: Since the separator program operates with the supervisor protection key, but in the program mode, your separator program must insure data protection during its execution.

When writing a separator program, you must observe the following programming requirements:

- Your program must conform to the standard linkage conventions. This includes saving and restoring the contents of registers 0 through 12, and 14. These registers can be preserved with the SAVE and RETURN macro instructions. When your program receives control, the address of a standard save area is in register 13.
- Your program must use the PUT macro instruction in the locate mode to write separation records on the output data set. (This method is required by the QSAM DCB that is open for the output data set.)
- Your program must establish its own synchronous error exit routine, and the address of this routine must be placed into the DCBSYNAD field of the output DCB. This gives control to your error exit routine in case an uncorrectable I/O error occurs while writing your program's output.
- Your program should use the RETURN macro instruction to return control to the output writer. Before returning, your program must free any main storage it obtained during its operation, and your program must place a return code (binary) in register 15. The return codes signify:

  0 -- Successful operation.

  8 -- Unrecoverable output error (should be set if your error exit routine is entered).

Output From the Separator Program

Your separator program can write any kind of separation identification. The jobname and the output classname for each job are available through the parameter list for inclusion in your output, if desired. You can use an IBM-supplied routine that constructs block characters (explained later). You can punch as many separator cards or print as many separator pages as you deem necessary.

The output from your separator program must conform to the attributes of the output data set. These attributes, which can be determined from the open output DCB pointed to by the parameter list, are:

- Record format (fixed, variable, or undefined length).
- Record length.
- Type of carriage control characters (machine, USASI, or none).

For printer-destined output, you can begin your separation records on the same page as the previous job output, or skip to any subsequent page. However, your separator program should skip at least one line before writing any records, because in some cases the printer is still positioned on the line last printed.

After completing the output of your separation records, your separator program should write sufficient blank records to force out the last separation record. This also allows your error exit routine to obtain control if an uncorrectable output error occurs while writing the last record. The requirements are:

- One blank record for printer-destined output.
- Three blank records for punch-destined output.

Using the Block Character Routine

For printer-destined output, your separator program can use an IBM-supplied routine to construct separation records in a block character format. This routine is a reenterable module named IEFSD095, and resides in the module library (SYS1.CI505).

The block character routine constructs block letters (A to Z), block numbers (0 to 9), and a blank. Your program furnishes the desired character string and the construction area. The block characters are constructed one line position at a time. Each complete character is contained in 12 lines and 12 columns; therefore, a block character area consists of 144 print positions. For each position, the routine provides either a space or the character itself.

The routine spaces 2 columns between each block character in the string. However, the routine does not enter blanks between or within the block characters. Your program must prepare the construction area with blanks or other desired background before entering the block character routine.

To use the IBM-supplied block character routine, your separator program executes the CALL macro instruction with the entry point name of IEFSD095. Since the block characters are constructed one line position at a time, complete construction of a block character string requires 12 entries to the routine. Each time you enter the routine, you must provide the address of a 4-word parameter list in register 1. The parameter list must contain the following:

Bytes 0 - 3 --	This word is the address of a field containing the desired character string in EBCDIC format.
Bytes 4 - 7 --	This word is the address of a full word field containing the line count as a binary integer from 1 to 12. This represents the line position to be constructed on this call.
Bytes 8 -11 --	This word is the address of a construction area in main storage where the routine will construct a line of the block character string. The required length in bytes of this construction area is 14n-2, where n represents the number of characters in the string.
Bytes 12-15 --	This word is the address of a full word field containing, in binary, the number of characters in the string.

Output Separators—PCP

For PCP configurations of the operating system IBM will supply output separators for output classes A and B. Final destination for class A output is the printer, and final destination for class B is the punch. Separators for either or both may be chosen at system generation, or both may be omitted. If you choose separator routines for classes A and/or B at system generation time, these routines will be entered automatically when one of these output classes is requested. The separation provided is the same as that for MVT configurations. The parameters passed to these routines are the same also; the switches in byte 0 indicate the type of output device. These switches are the three high-order bits only:

Class A	001	Device is a printer
	000	Device is a tape
Class B	111	Device is a 1442 punch
	011	Device is a punch
	010	Device is a tape
Others	011	Device is a punch
	111	Device is a 1442 punch
	001	Device is a printer
	000*	Device is a tape

*If the output device is a tape, this bit setting indicates to the IBM separator routines that final destination is the printer. To use IBM routines for punched output you must set bit 1 on to signify eventual destination as the punch. Therefore, the bit setting for punch separators at entrance to IBM routines would be 010. The classname is available to the user for determination of final destination.

MODIFYING OR ADDING OUTPUT SEPARATORS

If you choose the IBM separators for classes A and/or B, you may wish to employ a separator for classes other than A or B; in this case you must replace the module IEFSEPAR with your own routine or use the IBM supplied routines by setting the correct switch in the parameters for printer or punch final destination and branching to IEFSD094 or IEFSD095. These routines are entered by branch and link register 14. If you write your own routine, the procedure is the same as that described for MVT.

If you do not choose IBM separator routines, control is passed to IEFSEPAR for all output requests, and if you wish output separation, you must replace this with your own routine.

System Reader, Initiator, and Writer Cataloged Procedures

In the MVT and MFT operating system configurations, reader/interpreters, and output writers are controlled by cataloged procedures. In MVT configurations initiators are also controlled by a cataloged procedure. This chapter describes the reader, initiator, and writer cataloged procedures that are supplied by IBM, and tells how to write your own.

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: Job Control Language* publication (Form C28-6539) contains information about job control statements and cataloged procedures.

The *IBM System/360 Operating System: Operator's Guide* publication (Form C28-6540) describes the START command used to start reader/interpreters, initiators, and output writers.

The *IBM System/360 Operating System: Storage Estimates* publication (Form C28-6551) contains information for estimating storage requirements.

The *IBM System/360 Operating System: Utilities* publication (Form C28-6586) tells how to add a cataloged procedure to the procedure library.

The *IBM System/360 Operating System: Supervisor and Data Management Services* publication (Form C28-6646) discusses the queued sequential access method (QSAM) that is used by reader/interpreters and output writers.

Reader/Interpreter, Initiator and Output Writer Cataloged Procedures

In the MVT and MFT operating system configurations, system reader/interpreters and output writers are controlled by cataloged procedures. Initiators are controlled by cataloged procedures in MVT configurations. These procedures reside in the procedure library (SYS1.PROCLIB) and provide the parameters required for operation of the readers and writers.

IBM supplies three cataloged procedures for reader/interpreters, one for initiators, and one for output writers. You can:

- Use the IBM-supplied procedures.
- Use the IBM-supplied procedures, and override given parameters.
- Write and use your own cataloged procedures.
- Write and use your own cataloged procedures, and override given parameters.

The START command starts a reader/interpreter, an initiator, or an output writer, and designates the cataloged procedure to be used. You can override given parameters in the cataloged procedure by specifying the desired parameters in the START command. (A complete description of the START command is contained in the Operator's Guide publication.)

Some of your installation's parameters may differ consistently from those in the IBM-supplied procedure. If so, you may wish to use your own cataloged procedure, rather than respecifying the parameters in every START command. You can use your own cataloged procedure by:

1. Writing the procedure in the required format.
2. Adding the procedure to the procedure library.
3. Specifying the procedure name in the START command.

Reader/Interpreter Procedures

A cataloged procedure for reader/interpreters requires four job control statements: an EXEC statement and three DD statements. The names and purposes of these statements are listed below:

- An EXEC statement with the step name IEFPROC specifies the reader/interpreter program.
- A DD statement named IEFRDER provides the reader/interpreter with a description of the input stream.
- A DD statement named IEFPDSI describes the procedure library.
- A DD statement named IEFDATA defines the CPP (concurrent peripheral processing) data set that is used for intermediate storage of input stream data. (In MVT, the attributes of the CPP data set must not be changed for a checkpoint restart if the data set was open and not completely read. The extents and number of extents do not have to remain the same.)

IBM-SUPPLIED PROCEDURES

The standard reader/interpreter procedure supplied by IBM is named RDR. It specifies a block size of 80 bytes for the CPP data set. The complete standard procedure is:

```
//IEFPROC   EXEC   PGM=IEFIRC,REGION=48K,                        X
//                 PARM='80103005001024905010SYSDA   '
//IEFRDER   DD     UNIT=2400,LABEL=(,NL),VOLUME=SER=SYSIN,       X
//                 DCB=(BLKSIZE=80,LRECL=80,BUFL=80,             X
//                 BUFNO=1,RECFM=F)
//IEFPDSI   DD     DSNAME=SYS1.PROCLIB,DISP=OLD
//IEFDATA   DD     UNIT=SYSDA,                                   X
//                 SPACE=(80,(500,500),RLSE,CONTIG),             X
//                 DCB=(BLKSIZE=80,LRECL=80,BUFL=80,             X
//                 BUFNO=2,RECFM=F)
```

Two other cataloged procedures for reader/interpreters are supplied by IBM. These provide different block size specifications for the CPP data set. One of these procedures is named RDR400, and provides a block size of 400 bytes for the CPP data set. The RDR400 procedure is:

```
//IEFPROC   EXEC   PGM=IEFIRC,REGION=50K,                        X
//                 PARM='80103005001024905010SYSDA   '
//IEFRDER   DD     UNIT=2400,LABEL=(,NL),VOLUME=SER=SYSIN,       X
//                 DCB=(BLKSIZE=80,LRECL=80,BUFL=80,             X
//                 BUFNO=1,RECFM=F)
//IEFPDSI   DD     DSNAME=SYS1.PROCLIB,DISP=OLD
//IEFDATA   DD     UNIT=SYSDA,                                   X
//                 SPACE=(80,(500,100),RLSE,CONTIG),             X
//                 DCB=(BLKSIZE=400,LRECL=80,BUFL=400,           X
//                 BUFNO=2,RECFM=FB)
```

The third IBM-supplied procedure for reader/interpreters is named RDR3200. It provides a block size of 3200 bytes for the CPP data set. The RDR3200 procedure is:

```
//IEFPROC   EXEC   PGM=IEFIRC,REGION=52K,                                 X
//                 PARM='80103005001024905010SYSDA    '

//IEFRDER   DD     UNIT=2400,LABEL=(,NL),VOLUME=SER=SYSIN,                X
//                 DCB=(BLKSIZE=80,LRECL=80,BUFL=80,                      X
//                 BUFNO=1,RECFM=F)

//IEFPDSI   DD     DSNAME=SYS1.PROCLIB,DISP=OLD

//IEFDATA   DD     UNIT=SYSDA,                                            X
//                 SPACE=(80,(500,12),RLSE,CONTIG),                       X
//                 DCB=(BLKSIZE=3200,LRECL=80,BUFL=3200,                  X
//                 BUFNO=1,RECFM=FB)
```

PROCEDURE REQUIREMENTS

When creating your own reader/interpreter procedure, you must conform to the procedure format and the statement requirements. Use the IBM-supplied procedures as examples. The statement requirements are explained individually in the following paragraphs.

The EXEC Statement

The EXEC statement specifies the reader/interpreter program and for MVT configurations its region size. It also passes a set of parameters to the reader/interpreter program. The format for the EXEC statement is:

```
//IEFPROC   EXEC   PGM=IEFIRC,REGION=nnnnnK,                              X

//                 PARM='bpptttooommmiiicccrlssssssss'
```

The step name must be IEFPROC, as shown. The parameter requirements are as follows:

PGM=IEFIRC
    specifies the reader/interpreter program. The name of the program must be IEFIRC, as shown.

REGION=nnnnnK (valid for MVT configurations only)
    specifies the region size for the reader/interpreter. The value nnnnn represents a number from one to five digits that is multiplied by K (K=1024 bytes) to designate the region size. The region requirement depends on the size of the buffers and on the reader/interpreter modules (if any) in the link pack area. The complete algorithm for estimating the required region is contained in the "Estimating the Dynamic Main Storage Requirement" section of the _Storage Estimates_ publication. An insufficient size specification will result in an abormal termination. If blocked procedure library has been specified, the region size will have to be increased by the block size rounded off to the next highest multiple of 2K. This is to allow for the increase in buffer size. In the event that double buffering is used, the region size must be increased by twice the block size, rounded off to the next highest multiple of 2K.

PARM='bppttttooommmiiicccrlssssssss'
is a set of parameters for the reader/interpreter program. This parameter field must consist of 28 characters. Their meanings are explained in the following text.

b -- character from 0 through 9 or A through F that indicates whether the job step can be rolled out by another job step, whether it can cause rollout of another job step, whether an account number is required or not, and whether a programmer name is required. The chart below shows the meaning of each possible character.

Character	Can Step Be Rolled Out?	Can Step Cause Rollout?	Accn't Info Required?	Pgmr Name Required?
0	no	no	no	no
1	no	no	no	yes
2	no	no	yes	no
3	no	no	yes	yes
4	no	yes	no	no
5	no	yes	no	yes
6	no	yes	yes	no
7	no	yes	yes	yes
8	yes	no	no	no
9	yes	no	no	yes
A	yes	no	yes	no
B	yes	no	yes	yes
C	yes	yes	no	no
D	yes	yes	no	yes
E	yes	yes	yes	no
F	yes	yes	yes	yes

pp -- two numeric characters from 00 to 14 indicating the default priority for jobs read from this input stream. When no priority is specified in the JOB statement, this default priority is assigned to the job.

ttt -- three numeric characters indicating the default for the maximum time (in minutes) that each job step may run. (This value is not used by MFT but must be present.)

ooo -- three numeric characters indicating the default for the primary number of tracks assigned for SYSOUT data sets. This primary allocation should be made sufficient for most of your needs, so that secondary allocation will not usually be needed.

mmm -- three numeric characters indicating the default for the secondary number of tracks assigned for SYSOUT data sets.

iii -- three numeric characters under 255 indicating the dispatching priority of this reader while it is processing JCL statements. (This value is not used by MFT but must be present.)

ccc -- three numeric characters indicating the default for the region size (specified as a number of 1024-byte blocks) assigned to job steps read from this input stream. (This value is not used by MFT but must be present.)

r -- a numeric character from 0 to 3 that specifies the disposition of commands read from this input stream. The reader/interpreter, if r is:

0 -- executes the command.
1 -- displays the command (via a WTO macro instruction), and executes it.
2 -- displays the command (via a WTOR macro instruction), but does not execute it until advised by the operator.
3 -- ignores the command (treated as no operation).

1 -- a numeric character 0 or 1 which specifies the bypass label processing options. 0 signifies that the BLP parameter in the label field of a DD statement is to be ignored. The label parameter is processed as NL. 1 signifies that BLP is not to be ignored. The label parameter is processed as it appears.

ssssssss -- eight alphameric characters specifying the default device for SYSOUT. This becomes the UNIT subparameter in the DD statement that defines SYSOUT (if the UNIT field is omitted from the DD statement). If the designation is less than eight characters, the ssssssss field must be padded to the right with blanks.

Note: This default device can be specified by its address, group, or type. However, the UNIT=type form may cause all units of that type to be used for system output, since the device allocation program spreads the data sets among all candidate devices. To preserve some devices for private volumes, you should define a UNIT group which is a subset of the available direct access devices. You may specify the name SYSOUT as the default unit name for the system output data sets if it was specified at system generation time; when this default is used, a unit count of 1 is implied. UNITNAME SYSOUT is fully described in the System Generation publication.

DD Statement for the Input Stream

Your procedure for the reader/interpreter must include a DD statement that describes the input stream. The format for this statement is:

```
//IEFRDER  DD  UNIT=device,LABEL=(,type),                          X

//             VOLUME=SER=SYSIN,                                   X

//             DCB=(list of attributes)[,DSNAME=name,DISP=disposition]
```

This statement must be named IEFRDER, as shown. The IEFRDER statement can be overridden with a START command. The parameter requirements are as follows:

UNIT=device
: specifies the device from which the input stream is read. This can be any device supported by the queued sequential access method (QSAM). The device can be specified by its address, type, or group.

LABEL=(,type)
: describes the data set label (needed only for tape data sets). If this parameter is omitted, a standard label is assumed.

VOLUME=SER=SYSIN
specifies the volume containing the input stream. This parameter is required for magnetic tape or direct access volumes. The serial SYSIN is recommended for identification of this volume, but other serials can be used.

DCB=(list of attributes)
specifies the characteristics of the input stream and the buffers. If the BLKSIZE, LRECL, and BUFL subparameters are not specified, an 80-byte value is assigned to each. Other subparameter fields may be specified as needed; otherwise, the QSAM default attributes are assigned, as follows:

BUFNO	--	two buffers. (In MFT if the procedure is to be used for transient readers, BUFNO=1 must be specified.)
RECFM	--	U-format, with no control characters.
TRTCH	--	odd parity, no data conversion, and no translation.
DEN	--	lowest density.

DSNAME=name,DISP=disposition
specifies the name and disposition of the input stream data set to be read, this keyword should be used only with direct access input stream.

DD Statement for the Procedure Library

Your procedure for the reader/interpreter must include a DD statement that defines the procedure library. This statement must follow the IEFRDER statement which describes the input stream. The format for this statement is:

```
//IEFPDSI  DD    DSNAME=SYS1.PROCLIB,DISP=OLD
```

This statement must be named IEFPDSI, as shown. The parameter requirements are as follows:

DSNAME=SYS1.PROCLIB
identifies the procedure library. To concatenate other data sets with the system library, you may follow the IEFPDSI DD statement with other unnamed DD statements thus expanding the system procedure library.

DISP=OLD
specifies that the procedure library is an existing data set. In the MVT environment, the procedure library is assigned the share status (SHR) when referred to by the reader/interpreter.

DD Statement for the CPP Data Set

Your procedure for the reader/interpreter must include a DD statement that defines the CPP (concurrent peripheral processing) data set. Two DCB parameters (BLKSIZE, and buffer number) may be overridden by parameters in the input stream on DD* and DD DATA statements. The CPP data set is used for intermediate storage of input stream data. The format for this statement is:

```
//IEFDATA  DD     UNIT=device,                                          X

//                SPACE=(units,(quantities),RLSE,CONTIG),               X

//                VOLUME=SER=volser,DISP=(status,disp),                 X

//                DCB=(list of attributes)
```

This statement must be named IEFDATA, as shown. The parameter requirements are as follows:

UNIT=device
: specifies one or more direct access devices on which data sets from the input stream will be written. If more than one device is provided, the different data sets are not necessarily written in a continuous manner from device to device. Instead, the different data sets might be "spread" among the available devices in accordance with a reader/interpreter algorithm that is based on priorities and optimum access. If you want all the input stream data sets written on the same device, use the VOLUME parameter in this DD statement to identify the specific volume. The DEFER option must not be used.

  CAUTION: Do not use UNIT group names unless the request is for no more than one device, or the group is defined to have devices of only one type.

SPACE=(units,(quantities),RLSE,CONTIG)
: specifies space allocation for the direct access volume. The RLSE subparameter releases all unused space to the system when the data set is closed. The CONTIG subparameter ensures that space is allocated in contiguous tracks or cylinders.

  Note: The first space allocation made by the system will be for the reader/interpreter program itself, which does not need or use the space.

VOLUME=SER=volser
: identifies a specific direct access volume. This parameter is not required, but you can use it to cause all input stream data sets to be written on the same volume. You should also use this parameter if you specify the DISP parameter.

DISP=(status,disp)
: specifies the status and disposition of the CPP data set. This parameter is not required, but can be used to bypass the first space allocation (as explained above). To do this, specify the parameter as DISP=OLD. The system then assumes that the data set exists, and does not allocate space for the reader/interpreter program. Subsequently, the reader/interpreter forces a DISP=NEW,PASS status for the CPP data set so that space is allocated on it for recording the input stream data sets.

DCB=(list of attributes)
: specifies the characteristics of the CPP data set and the buffers. The subparameters may be specified as needed. The BLKSIZE, LRECL, and BUFL subparameters must be specified in all cases. The BLKSIZE and BUFNO parameters may be overridden by specifying them on a DD*

or DD DATA statement in the reader input stream. However, the BLKSIZE and BUFNO values on the IEFDATA statement are always used as upper limits. Thus, if the overriding statements exceed these limits, the IEFDATA values are used. (An explanation of how to override these parameters is contained in the Job Control Language publication.) The BUFNO and RECFM subparameters, if not specified, assume the QSAM default attributes which are:

BUFNO -- two buffers.

RECFM -- U-format, with no control characters.

Initiator Procedures

A cataloged procedure for an initiator requires only one job control statement: an EXEC statement. Additional DD statements may be optionally added so that specific control volumes will be allocated to the initiator task.

- An EXEC statement with the step name IEFPROC specifies the initiator program and any job classes to be associated with the initiator if the START command does not specify job classes.
- Optional DD statements specify control volumes to be allocated to the initiator task.

IBM-SUPPLIED PROCEDURE

The standard initiator procedure supplied by IBM is named INIT. The INIT procedure is:

```
//IEFPROC   EXEC   PGM=IEFSD060,PARM=A
```

PROCEDURE REQUIREMENTS

When creating your own initiator procedures, you must conform to the procedure format and the statement requirements. The statement requirements are explained individually in the following paragraphs.

The EXEC Statement

The EXEC statement specifies the initiator program and passes a set of parameters to the initiator program. The format for the EXEC statement is:

```
//IEFPROC   EXEC   PGM=IEFSD060,PARM=xxxxxxxx
```

The step name must be IEFPROC, as shown. The parameter requirements are as follows:

PGM=IEFSD060
: specifies the initiator program. The name of the program must be IEFSD060, as shown.

PARM=xxxxxxxx
specifies from zero to eight (no padding required) single-character class names for job input. These specify the class of input that the initiator can process, and also establish the priority of the input job classes, with the highest priority on the left. If class name parameters are included in the START command, they override the entire set of class names in the cataloged procedure.

DD Statements

DD statements for control volumes are optional. The standard procedure INIT does not include a DD statement for a control volume. This optional facility is discussed in the next section "Mounting Control Volumes in MVT."

ADDITIONAL INITIATOR FACILITIES

Mounting Control Volumes in MVT

A control volume that will be referenced during a catalog search can be mounted before the search begins to avoid the possibility of a job failure because the necessary control volume was not mounted.

DD statements for control volumes may be included in initiator procedures cataloged in the procedure library (SYS1.PROCLIB). Such DD statements cause direct access volumes to be mounted and allocated for the life of the initiator. This facility is particularly useful when control volumes will be needed for departmental job batches.

Initiation by an initiator with a DD statement for a control volume ensures that the control volume will be mounted prior to a catalog search from the catalog on the system residence volume to the catalog on the control volume for a specified data set. If such DD statements for control volumes are not included in initiator procedures, attempt will be made to mount a required control volume if a catalog search could not be completed during allocation for a step. However, when control volumes are mounted in this manner, they are available for demounting immediately after the catalog search has completed and will not necessarily remain mounted for the life of the job or job step requiring them.

Initiator Action

By starting an initiator that includes a DD statement for a control volume, mounting is requested before the initiator is allowed to start initiating jobs. If the volume is already mounted, the initiator proceeds with initiation.

When a stop command is issued to the started initiator and the volume is demountable and PRIVATE, it will be demounted providing no other job steps or other initiators are allocated to the volume. the volume then would stay mounted until the last job step using it terminated or the initiators using it are stopped, at which time the volume would be demounted.

DD Statement Formats

As many volumes may be defined by DD statements in the initiator procedure as the user finds useful. The format follows the specifications contained in the Job Control Language publication. The following is an example of a DD statement that could be included in an initiator procedure for a control volume:

```
//ddname  DD  VOLUME=(PRIVATE,SER=ser#),                          X

                    ⎧address⎫
//            UNIT= ⎨type   ⎬,DISP=OLD
                    ⎩group  ⎭
```

VOLUME=(PRIVATE,SER=ser#),
 specifies the volume serial of the control volume. PRIVATE ensuresthat this volume will not be used to satisfy job step data set requests unless requested by the specify volume serial number. Also, unless already mounted and permanently resident or reserved, the volume will be demounted when the initiator is stopped or upon its last use by job steps being processed by other initiators, or when other initiators allocated to the volume are stopped.

```
      ⎧address⎫
UNIT= ⎨type   ⎬,
      ⎩group  ⎭
```

 specifies the unit address, unit type, or group on which the control volume is to be mounted.

DISP=OLD
 specifies that a temporary data set will not be allocated to the volume. A dsname will be generated for this data set and when the initiator is stopped a message will be written out on the system output that this data set (generated name) has been kept. This message can be ignored as no action needs to be taken.

Output Writer Procedures

A cataloged procedure for output writers requires two job control statements: an EXEC statement and a DD statement.

- An EXEC statement with the step name IEFPROC specifies the output writer program.
- A DD statement named IEFRDER defines the output data set. (In MVT, the attributes of the output data set must remain unchanged for a deferred checkpoint restart if the data set was opened but not completely written. The extents and number of extents do not have to be the same.)

IBM-SUPPLIED PROCEDURE

The standard output writer procedure supplied by IBM is named IEEVWPCR. The standard procedure is:

```
//IEFPROC  EXEC  PGM=IEFSD080,REGION=20K,                          X
//               PARM='PA'
//IEFRDER  DD    UNIT=1403,VOLUME=(,,,35),                         X
//               DSNAME=SYSOUT,DISP=(NEW,KEEP),                    X
//               DCB=(BLKSIZE=133,LRECL=133,BUFL=133,              X
//               BUFNO=2,RECFM=FM)
```

PROCEDURE REQUIREMENTS

When creating your own output writer procedure, you must conform to the procedure format and the statement requirements. Use the IBM-supplied procedure as an example. The statement requirements are explained individually in the following paragraphs.

The EXEC Statement

The EXEC statement specifies the output writer program and its region size. It also passes a set of parameters to the output writer program. The format for the EXEC statement is:

```
//IEFPROC  EXEC  PGM=IEFSD080,REGION=nnnnnK,                              X
//               PARM='cxxxxxxxx,seprname'
```

The step name must be IEFPROC, as shown. The parameter requirements are as follows:

PGM=IEFSD080
: specifies the output writer program. The name of the program must be IEFSD080, as shown.

REGION=nnnnnK (MVT configurations only)
: specifies the region size for the output writer. The value nnnnn represents a number from one to five digits that is multiplied by K (K=1024 bytes) to designate the region size. The region requirement depends on the size of the buffers, the data set writer used, and which modules of the output writer (if any) are in the link pack area. The complete algorithm for estimating the required region is contained in the "Estimating the Dynamic Main Storage Requirement" section of the _Storage Estimates_ publication. An insufficient size specification will result in an abnormal termination.

PARM='cxxxxxxxx,seprname'
: is a set of parameters for the output writer program. The first part of this parameter field can contain from two to nine characters. The second part of this parameter field, if specified, is separated from the first part by a comma, and contains a program name from one to eight characters. Both parts of this parameter field are explained below.

  c -- an alphabetic character, either P (for printer) or C (for punch), that specifies the type of control characters for the output of the writer.

  xxxxxxxx -- from one to eight (no padding required) single-character class names for system output. These specify the type of output that the writer can process, and also establish the priority of the output classes, with the highest priority on the left. If class name parameters are included in the START command, they override this entire set of class names in the cataloged procedure.

  seprname -- the name of the program (up to eight characters) that provides job separation in the output data set. The named program must reside in the link library (SYS1.LINKLIB). You can specify the name IEFSD094 to use the output separator supplied by IBM, or you can specify the name of your own program. This subparameter may be omitted, in which case no output separator is used. (Output separators are described in another chapter of this publication.)

DD Statement for the OUTPUT Data Set

Your procedure for the output writer must include a DD statement that defines the output data set. The format for this statement is:

```
//IEFRDER  DD     UNIT=device,LABEL=(,type),                      X

//                VOLUME=(,,,volcount),                           X

//                DSNAME=anyname,DISP=(NEW,KEEP),                 X

//                DCB=(list of attributes)
```

This statement must be named IEFRDER, as shown. The parameter requirements are as follows:

UNIT=device
specifies the printer, magnetic tape, or card punch device on which the output data set will be written. The devices that can be used are: 1403, 1442, 1443, 2400, 2400-1, 2400-2, 2400-3, 2420, or 2540.

LABEL=(,type)
describes the data set label (needed only for tape data sets). If this parameter is omitted, a standard label is assumed.

VOLUME=(,,,volcount)
limits the number of tape volumes that can be used by this writer during its entire operation (from the time it is started to the time it is stopped). This parameter is not required for printer or card punch devices.

DSNAME=anyname
specifies a name for the output data set (tape only), so that it can be referred to by subsequent job steps. This name is also necessary for specification of the KEEP subparameter in the DISP field.

DISP=(NEW,KEEP)
specifies the KEEP subparameter to prevent deletion of the output data set (tape only) at the conclusion of the job step.

DCB=(list of attributes)
specifies the characteristics of the output data set and the buffers. The BLKSIZE and LRECL subparameter fields must be specified in all cases. The BUFL subparameter field, if not specified, is calculated on the basis of the BLKSIZE value. Other subparameter fields may be specified as needed; otherwise, they will assume the QSAM default attributes which are:

BUFNO	-- three buffers for the 2540 device, two buffers for all other devices.
RECFM	-- U-format, with no control characters.
TRTCH	-- odd parity, no data conversion, and no translation.
DEN	-- lowest density.

By using a certain kind of procedure, it is possible to reduce the amount of CPU time needed by the writer. This is done by having the SYSOUT writer intercept PUT instructions and execute an EXCP only when all of a chain of buffers are full. This command chaining is provided if the writer procedure specifies all of the following conditions:

1. It uses more than 3 buffers.
2. It uses machine control characters in writing to the OUTPUT print or punch device.
3. It does not use PCI.
4. The OUTPUT device is a printer or punch.

It should be noted that if a command chaining procedure is used to a punch, there is no automatic punch recovery even though there are more than 3 buffers.

Cataloging the Procedure

You use the IEBUPDTE utility program to add your reader, initiator, or writer procedures to the cataloged procedure library (SYS1.PROCLIB). Use of this program is fully explained in the Utilities publication.

The following example shows the control statements needed to add a reader/interpreter procedure and an output writer procedure to the procedure library. For this example, the reader/interpreter procedure is named RDPROC4, and the output writer procedure is named WTPROC2.

The EXEC statement in this example specifies the IEBUPDTE program. The PARM=NEW parameter indicates that all input to the utility program is contained in the data set defined by the SYSIN statement.

The ADD control statement furnishes the name of the member to be added to the procedure library. The three numbers following the member name indicate:

- The level of modification (00 indicates first run).
- The source of the modification (0 indicates user-supplied).
- The printed output desired (ALL indicates print entire updated member and control statements).

The NUMBER statement specifies the sequence numbers for records within the new member. With this statement, the number 00000010 is assigned to the first record of the new procedure, and subsequent records are incremented by 00000010.

```
//NEWPROCS JOB     09#770,D.P.BROWN
//         EXEC    PGM=IEBUPDTE,PARM=NEW
//SYSPRINT DD      SYSOUT=A
//SYSUT2   DD      DSNAME=SYS1.PROCLIB,DISP=OLD
//SYSIN    DD      DATA
./         ADD     RDPROC4,LEVEL=00,SOURCE=0,LIST=ALL
./         NUMBER  NEW1=10,INCR=10
//IEFPROC  EXEC    PGM=IEFIRC,REGION=40K,                                X
//                 PARM='80101001501024905O1SYSDA  '
//IEFRDER  DD      UNIT=2400-2,LABEL=(,NL),                               X
//                 VOLUME=SER=SYSIN,                                      X
//                 DCB=(BLKSIZE=80,LRECL=80,BUFL=80,                      X
//                 BUFNO=1,RECFM=F,TRTCH=C,DEN=0)
//IEFPDSI  DD      DSNAME=SYS1.PROCLIB,DISP=OLD
//IEFDATA  DD      UNIT=2311,                                             X
//                 SPACE=(80,(500,500),RLSE,CONTIG),                      X
//                 VOLUME=SER=222222,DISP=OLD,                            X
//                 DCB=(BLKSIZE=80,LRECL=80,BUFL=80,                      X
//                 BUFNO=2,RECFM=F)
./         ADD     WTPROC2,LEVEL=00,SOURCE=0,LIST=ALL
./         NUMBER  NEW1=10,INCR=10
//IEFPROC  EXEC    PGM=IEFSD080,REGION=20K,                               X
//                 PARM='PAC'
//IEFRDER  DD      UNIT=2400-2,LABEL=(,NL),VOLUME=(,,,40),                X
//                 DSNAME=SYSOUT,DISP=(NEW,KEEP),                         X
//                 DCB=(BLKSIZE=133,LRECL=133,RECFM=F,                    X
//                 TRTCH=C)
/*
```

Automatic SYSIN Batching (ASB)

Reader/interpreters in MVT are usually resident and continuously active; they read and interpret job control language statements and place SYSIN data sets on direct access devices for later processing. Since the interpreting of job control language statements often requires only a small proportion of the total time used by the reader/interpreter yet remains resident even when inactive, you may save space by separating the interpretation of job control statements from the storing of SYSIN data sets. If the two functions are separated, the interpreter portion of the reader/interpreter does not have to be resident at all times and will be called into storage only after a certain number of job control language statements have been collected. Separating the two functions of the reader/interpreter is called automatic SYSIN batching (ASB).

IBM supplies a cataloged procedure that provides automatic SYSIN batching; this procedure is named RDRA and is invoked by issuing a START command. The procedure is shown and described in the following text; by using it as a model, you may write your own procedure, coding the parameters suited to your installation.

```
//IEFPROC      EXEC   PGM=IEFVMA,REGION=16K,
//   PARM='80103005001024905030SYSDAbbb,11012SYSDAbbb'
//IEFRDER      DD     UNIT=2400,LABEL=(,NL),VOLUME=SER=SYSIN,
//   DCB=(BLKSIZE=80,RECFM=F,BUFL=80,BUFNO=10)
//IEFPDSI      DD     DSNAME=SYS1.PROCLIB,DISP=OLD
//IEFDATA      DD     UNIT=SYSDA,SPACE=(3200,(15,15),RLSE,CONTIG),
//   DCB=(BLKSIZE=3200,BUFNO=2,RECFM=FB,BUFL=3200)
```

The EXEC statement for the SYSIN batcher is similar to the EXEC statement for the standard reader/interpreter. It specifies the SYSIN batcher program, the MVT region size and passes a set of parameters to the program. The format is as follows:

```
//IEFPROC      EXEC   PGM=IEFVMA,REGION=nnnnnK,
//   PARM='bpptttooommmiiicccrlssssssssejjaaddddddd'
```

The step name must be IEFPROC, as shown. The parameters are as follows:

PGM=IEFVMA
: specifies the automatic SYSIN batcher program. It must be IEFVMA as shown.

REGION=nnnnnK
: specifies the region size for the ASB reader. The value nnnnn represents a number from one to five digits that is multiplied by K (K=1024 bytes) to designate the region size. The region requirement depends on the size and number of input buffers and ASB reader modules (if any) in the link pack area. The complete algorithm for estimating the required region is contained in the "Estimating the Dynamic Main Storage Requirement" section of the Storage Estimates publication. An insufficient size specification will result in an abnormal termination.

PARM='bppttttooommmiiicccrlssssssss'
is a set of parameters for the reader/interpreter program. This parameter field must consist of 28 characters. Their meanings are explained in the following text.

b -- character from 0 through 9 or A through F that indicates whether the job step can be rolled out by another job step, whether it can cause rollout of another job step, whether an account number is required or not, and whether a programmer name is required. The following chart shows the meaning of each possible character.

Character	Can Step Be Rolled Out?	Can Step Cause Rollout?	Accn't Info Required?	Pgmr Name Required?
0	no	no	no	no
1	no	no	no	yes
2	no	no	yes	no
3	no	no	yes	yes
4	no	yes	no	no
5	no	yes	no	yes
6	no	yes	yes	no
7	no	yes	yes	yes
8	yes	no	no	no
9	yes	no	no	yes
A	yes	no	yes	no
B	yes	no	yes	yes
C	yes	yes	no	no
D	yes	yes	no	yes
E	yes	yes	yes	no
F	yes	yes	yes	yes

pp -- two numeric characters from 00 to 14 indicating the default priority for jobs read from this input stream. When no priority is specified in the JOB statement, this default priority is assigned to the job.

ttt -- three numeric characters indicating the default for the maximum time (in minutes) that each job step may run.

ooo -- three numberic characters indicating the default for the primary number of tracks assigned for SYSOUT data sets. This primary allocation should be made sufficient for most of your needs, so that secondary allocation will not usually be needed.

mmm -- three numeric characters indicating the default for the secondary number of tracks assigned for SYSOUT data sets.

iii -- three numeric characters under 255 indicating the dispatching priority of this reader while it is processing JCL statements.

ccc -- three numeric characters indicating the default for the region size (specified as a number of 1024-byte blocks) assigned to job steps read from this input stream.

r -- a numeric character from 0 to 3 that ordinarily specifies the disposition of commands read from this input stream. However, since the automatic SYSIN batcher handles all commands in the input stream, this field will have no effect on the reader/interpreter though it must be present.

l -- a numeric character 0 or 1 which specifies the bypass label processing options. 0 signifies that the BLP parameter in the label field of a DD statement is to be ignored. The label parameter is processed as NL. 1 signifies that BLP is not to be ignored. The label parameter is processed as it appears.

ssssssss -- eight alphameric characters specifying the default device for SYSOUT. This becomes the UNIT subparameter in the DD statement that defines SYSOUT (if the UNIT field is omitted from the DD statement). If the designation is less than eight characters, the ssssssss field must be padded to the right with blanks.

Note: This default device can be specified by its address, group, or type. However, the UNIT=type form may cause all units of that type to be used for system output, since the device allocation program spreads the data sets among all candidate devices. To preserve some devices for private volumes, you should define a UNIT group which is a subset of the available direct access devices. You may specify the name SYSOUT as the default unit name for the system output data sets if it was specified at system generation time; when this default is used, a unit count of 1 is implied. UNITNAME SYSOUT is fully describes in the System Generation publication.

e -- a numeric character from 0 to 3 that ordinarily specifies the disposition of commands read from this input stream. The SYSIN batcher, if e is:

0 -- executes the command.
1 -- displays the command (via a WTO macro instruction), and executes it.
2 -- displays the command (via a WTOR macro instruction), but does not execute it until advised by the operator.
3 -- ignores the command (treated as no operation).

jj -- two numeric characters which indicate the number of jobs to be read by the SYSIN batcher before control is to be passed to the reader/interpreter for interpreting the job control language and enqueueing the jobs onto the job input queue for execution.

aa -- the number of logical tracks on the SYS1.SYSJOBQE which can be used by the SYSIN batcher for later interpretation. The Storage Estimates publication contains information on estimating SYS1.SYSJOBQE work space.

dddddddd -- the unit type or name of the direct access device where the SYSIN batcher is to temporarily store all SYSIN data. This must be the same as that indicated in the IEFDATA UNIT parameter.

The DD statements are the same as those described for the standard reader/interpreter with the exceptions noted in the following text.

IEFRDER (DD statement for the input stream)
The parameter requirements are the same as those for the reader/interpreter except for the DCB parameter. This parameter specifies the characteristics of the input stream and the buffers.

If the BLKSIZE and BUFL subparameters are not specified, an 80-byte value is assigned to each. LRECL need not be specified because fixed length 80-byte records are the only input accepted by the ASB reader. Other subparameter fields may be specified as needed; otherwise, the QSAM default attributes are assigned as for the reader/interpreter.

IEFDATA (DD statement for the CPP data set)

If the BLKSIZE and BUFL subparameters are not specified, an 80-byte value is assigned to each. LRECL need not be specified because fixed length 80-byte records are the only input accepted by the ASB reader. The BLKSIZE and BUFNO parameters may be overridden by specifying them on a DD* or DD DATA statement in the reader input stream. However, the BLKSIZE and BUFNO values on the IEFDATA statement are always used as upper limits. Thus, if the overriding statements exceed these limits, the IEFDATA values are used. In addition, the ASB reader always uses one buffer for IEFDATA. Therefore, the BUFNO value specified applies only as a default.

SYSIN and SYSOUT Data Blocking

Significant performance advantages can be gained by blocking of SYSIN and SYSOUT data. Blocking reduces interference on the devices containing the intermediate data and improves direct access space use. The IBM-supplied reader procedures provide three levels of SYSIN blocking; you should review the blocking provided by the cataloged procedures of the various processors. Figure 8 shows the data blocking that is accepted by processors operating under MVT and MFT configurations.

Blocking is obtained by including in the appropriate DD statement DCB information in the general form

```
DCB=(RECFM=x,LRECL=x,BLKSIZE=x)
```

The various programmer's guides should be consulted to determine options that need not be specified in individual cases. LRECL must be specified for the PL/I and FORTRAN H SYSLIN DD cards, and the COBOLF SYSPUNCH DD card, when these files are blocked. Assembler F, COBOL F, and FORTRAN G and H are effectively unlimited. Sort is limited by assembled-in values. The utilities and RPG are limited by assembled-in values of LRECL but may have a blocking factor other than 1. SYSIN and SYSOUT for the FORTRAN E compiler cannot be blocked through the system input reader and output writer, although the SYSOUT DD cards must include DCB=BLKSIZE=121.

When you institute data blocking, you must consider the following variables:

SIZE option
REGION values
MINPART value
Default REGION value provided by the reader procedure

The FORTRAN H SIZE parameter is independent of blocking and buffering considerations, although the REGION value must be 8K larger than the SIZE value.

Notes to Figure 8:
(Data Blocking Accepted by Processors under MVT and MFT)

For compile-load-go cases, only the compile step must include complete SYSIN (SYSGO) DCB specifications.

F=Fixed, FA=Fixed, USASI control characters, FB=Fixed blocked, FBA=Fixed blocked, USASI control characters, FBM=Fixed blocked, machine control characters, VBA=Variable blocked, USASI control characters, FT=Full track, U=Undefined.

Region and partition sizes must be adequate to accommodate the specified blocking. The user should consult the individual programmer guides.

	LRECL RECFM BLKSIZE			
Processor	SYSPRINT	SYSPUNCH	SYSIN (IEFDATA)	SYSLIN (≤3200)
Assembler F	121 FBM FT	80 FB FT	80 FB FT	80 FB FT
COBOL F	121 FBA FT	80 FB FT	80 FB FT	80 FB FT
FORTRAN E (with PRFRM option)	121 FM 121	80 F 80	80 FB FT	80 FB FT
FORTRAN G	120 FBA FT	80 FB FT	80 FB FT	80 FB FT
FORTRAN H	137 VBA FT	80 FB FT	80 FB FT	80 FB FT
PL/I F	125 VBA FT	80 FB FT	80 FB FT	80 FB FT
Linkage Editor E15,E18	121 FM 121			80 F,FS 80
Linkage Editor F44	121 FM,FBM 605			80 F,FS,FB,FBS 400
Linkage Editor F88,F128	121 FM,FBM FT≤4840			80 F,FS,FB,FBS 3200
Sort	 U 120		80 FB FT	
RPG	121 FA 121	80 F 80	80 FB FT	80 F 80
Utilities	121 FBA FT	NA	80 FB FT	

Figure 8. Data Blocking Accepted by Processors under MVT and MFT

Blocking the Procedure Library

You may, in some cases, improve the use of direct access space and gain performance advantages by blocking the procedure library. It may be blocked at system generation or subsequently by using the operating system utilities. Block size must be a multiple of 80. Increased buffer size necessary for a blocked procedure library must be provided for in the region parameter of the reader procedures for MFT and MVT. The region size must be increased by the block size rounded to the next higher multiple of 2K. The PCP scheduler correspondingly requires more storage at each of its design levels.

In cases where the region size has been increased for blocked SYSIN/SYSOUT in excess of that actually required (due to rounding) and the excess is greater than the block size for the procedure library, a further increase in region size may not be necessary for processing blocked records from the procedure library.

The following example shows the control statements needed to block the procedure library using the IEBCOPY and IEHPROGM utility programs. Step C1 of job BLOCK copies the procedure library and blocks it to 400. It deletes the old copy and catalogs the new copy under the name of LIBCOPY. Step R1 renames the procedure library to SYS1.PROCLIB and catalogs it under that name.

```
//BLOCK    JOB  ACCT,D82,MSGLEVEL=1

//C1       EXEC PGM=IEBCOPY

//SYSUT1   DD   DSNAME=SYS1.PROCLIB,UNIT=2311,DISP=(OLD,DELETE,KEEP)

//SYSUT2   DD   DSNAME=LIBCOPY,UNIT=2311,VOLUME=SER=111111,           X

//              DISP=(NEW,CATLG,DELETE),DCB=(RECFM=FB,LRECL=80,       X

//              BLKSIZE=400),SPACE=(TRK,(50,1,10))

//SYSPRINT DD   SYSOUT=A

//SYSIN    DD   DUMMY

/*


//R1       EXEC PGM=IEHPROGM

//DD1      DD   UNIT=2311,VOLUME=SER=111111,DISP=OLD

//SYSPRINT DD   SYSOUT=A

//SYSIN    DD   *

  RENAME DSNAME=LIBCOPY,VOL=2311=111111,NEWNAME=SYS1.PROCLIB

  CATLG DSNAME=SYS1.PROCLIB,VOL=2311=111111

/*
```

Writing Rollout/Rollin Installation Appendages

This chapter explains how to write rollout/rollin appendages for MVT configurations of the operating system and how to insert them into the operating system before or after system generation. The four exits to user-written appendages and their functions are explained. The chapter also presents sample coding for an appendage.

Additional information on insertion of these appendages at system generation is contained in the publication IBM System 360 Operating System: System Generation, Form C28-6554.

The publication IBM System 360 Operating System: Job Control Language, Form C25-6539 explains how to indicate that a job step may be rolled out or may cause rollout of another job step.

Writing Rollout/Rollin Installation Appendages

The rollout/rollin feature of IBM System/360 Operating System is used with MVT configurations as an aid to main storage management. Rollout/rollin allows the temporary, dynamic expansion of your job step beyond its originally specified region. When your job step needs more space, rollout/rollin attempts to obtain unassigned storage for its use. If there is no such unassigned storage, another job step is rolled out -- transferred to auxiliary storage (IBM 2301, 2311, 2314 or 2321 -- so that its region may be used by your job step. When released by your job step, this additional storage is again available, either as unassigned storage, if that was its source, or to receive the job step to be transferred back into main storage (rolled in). (Note: Teleprocessing jobs which use the Autopoll option should not be marked eligible for rollout. A rolled-out job which is using the Autopoll option cannot be restarted properly.)

During the course of normal rollout processing, exits are taken to installation-written routines, so that you can dynamically control various aspects of the rollout function. The routines you write must be serially reusable; they will reside as part of the resident nucleus and will be entered by a branch entry. IBM has supplied a dummy module which resolves the appendage exits during system generation.

A copy of the IBM supplied dummy user appendage module, IEAQRAPG, in symbolic form will reside in SYS1.SAMPLIB. You may use this as a basic module in which to incorporate your code for one or more appendages. You may extract the symbolic deck for inclusion into one of your own symbolic libraries, or you may enter your changes directly onto the module in SYS1.SAMPLIB through the use of IEBUPDTE. To replace the dummy module before system generation, the object module which results from the assembly of the updated appendage routine should be link edited into the SYS1.CI505. To replace the assembled dummy appendage module after system generation, you should link edit your new appendage module as a CSECT replacement in IEANUC01.

It may be necessary for the appendages to address the jobname; however, unless the job has issued an ATTACH, SYSINIT will appear in the jobname, and the actual jobname will appear in the stepname. Therefore, an appendage checking for a specific jobname should also check for SYSINIT; if it is encountered, the appendage should further check the stepname for the actual jobname.

There are four installation exits; their functions and the linkage to them are discussed in the following paragraphs.

Linkage to User Appendages

1. Register 15 contains the base address of the routine.

2. Register 14 contains the return address.

3. Register 13 contains the address of an 18-word save area in which you must save any registers that you will use. You must restore registers before exiting.

4. Register 1 contains the address of the TCB for the task that invoked rollout. (Exception: on entry to Appendage IV, register 1 contains the address of the PQE for the region selected for rollout.)

5. Register 0 contains the address of a three-fullword area. The first two bytes of the first word contain the number of rollouts now in effect. The third and fourth bytes of the first word contain the number of requestors now queued for rollout. The rollout queue is ordered according to dispatching priority. The second word contains the address of the queue origin for queued rollout requests (IEAROQUE). The third word is the address of the parameter list for the task that invoked rollout. The first word of the two-word parameter list contains the address of the TCB for the task that invoked rollout, and the second word contains a hexadecimal number which represents the length, in bytes, of the originally requested main storage area.

APPENDAGE I: IEAQAPG1

The exit to Appendage I is taken when the current request for additional storage has invoked rollout, and at least one other job step has already invoked rollout. You can determine, using your own criteria, whether to override the normal rollout procedure of allowing only one job step to invoke rollout at any given time. If you do allow multiple (successive) rollouts, you are responsible for preventing system interlocks such as occur if each of two job steps needed two-thirds of main storage at the same time. (Your obvious escape from this situation would be to arbitrarily cancel one of the steps.) If you do not elect to allow multiple rollouts, the requesting task is placed upon the queue of tasks that have requested and are waiting for rollout. From the linkage information passed in the registers, you must decide whether or not to make an immediate attempt at rollout for the requesting step. If you do not desire an immediate attempt at rollout, you should return the TCB address passed in register 1 without change. If you do desire an immediate attempt at rollout, you should return the address of the requesting task in complement form. If you use the IBM-supplied Appendage I, your request will be queued and no multiple rollout will occur.

APPENDAGE II: IEAQAPG2

The Appendage II exit is taken whenever neither enough free space nor a rolloutable job step of lower dispatching priority than the job step that invoked rollout exists. No attempt is made to find a higher dispatching priority step to roll out. You have the option of requesting that the rollout function attempt to find a job step of higher dispatching priority that can be rolled out.

If you do not want to attempt to find a higher dispatching priority step to roll out, return the address of the requesting task without change. If you do desire the higher dispatching priority pass, return the address in complement form.

APPENDAGE III: IEAQAPG3

The exit to Appendage III is taken after the rollout function has determined, through the use of both its own and (optionally) your criteria, that a job step suitable for rollout does not exist. Through this appendage you can select either the step which requested the unavailable storage or any other job step in the system for abnormal termination (ABEND). If you do not select a job step for ABEND (or if you use the IBM-supplied Appendage III), the requestor is placed on the rollout queue. If a job step other than the requestor is selected by the appendage, ABEND of the selected job step is initialized, and the requestor is queued for rollout.

If you do not desire to initiate an ABEND, you must set register 1 to zero before exiting. The requestor is then queued for rollout. If you do desire an ABEND, you must return in register 1 the address of the job step TCB for the task to be ABENDed. (The address you return will be checked to ensure that it is a job step TCB. If it is not, it is ignored and the requestor is queued for rollout.) If the address is valid and is not the address of the requesting step, ABEND is initiated and the requestor is queued for rollout. If it is the address of the requesting step, ABEND is initiated and the requestor's IQE is returned to the available queue. If you use the IBM-supplied Appendage III, no ABEND occurs.

APPENDAGE IV: IEAQAPG4

The Appendage IV exit is taken each time a job step has been selected as a candidate for rollout. This appendage gives you the opportunity to apply your criteria to each job step that the rollout function has found to be eligible for rollout. Job steps are considered for rollout eligibility beginning with the job step of lowest dispatching priority, and continuing upward until all eligible job steps with a lower dispatching priority (than that of the requesting job step) have been presented to your appendage. If you have supplied an appendage which permits job steps of higher dispatching priority to be eligible for rollout, these will also be presented to your appendage beginning with the job step of next highest dispatching priority (than that of the requesting step), and continuing upward until all eligible job steps with a higher dispatching priority have been presented.

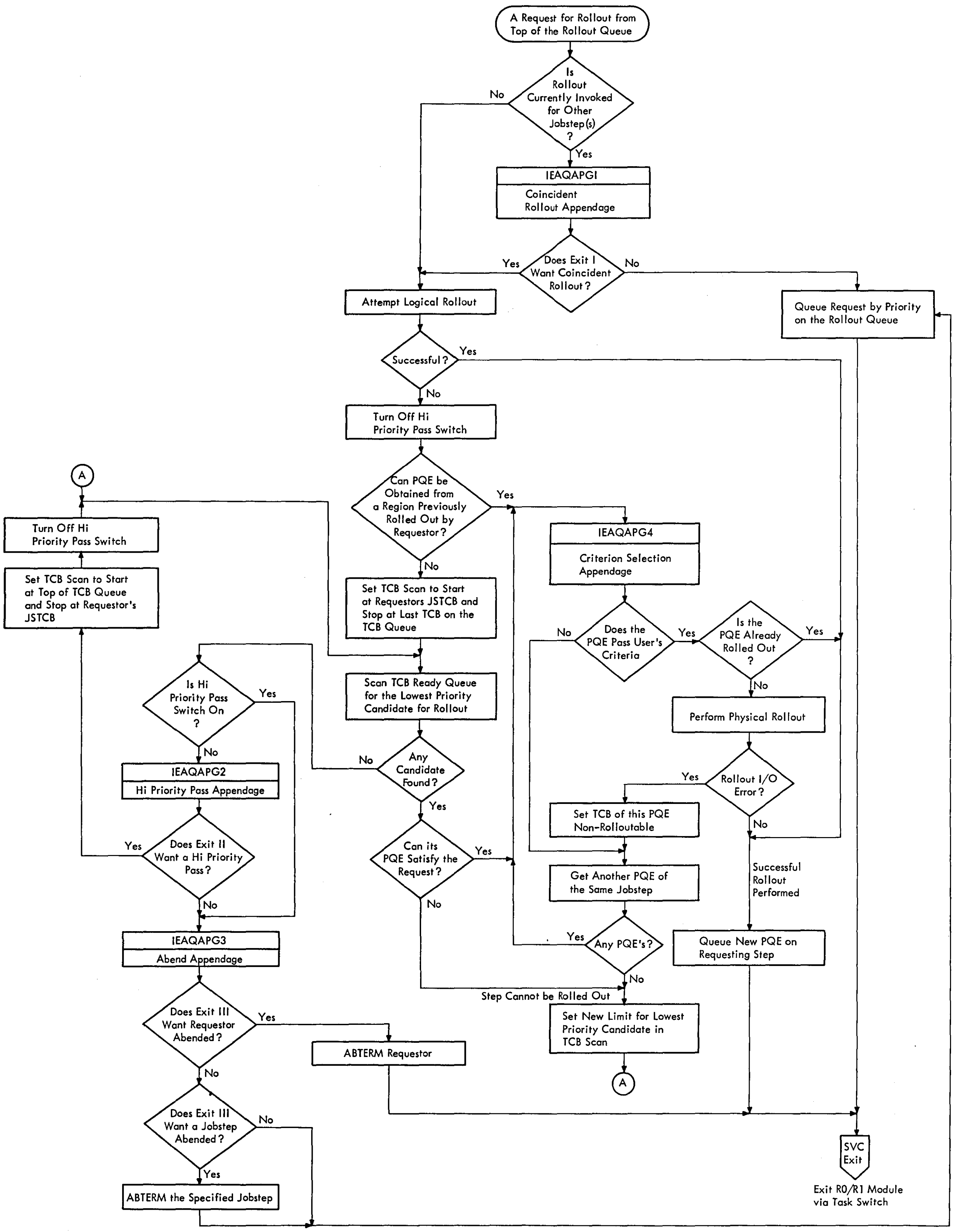
The process of presenting job steps to your appendage for approval continues either until a job step is approved for rollout by the appendage, or until all eligible job steps have been examined and disapproved by the appendage.

SAMPLE CODING OF APPENDAGES

The following pages contain sample coding illustrating the linkage to the appendages. In the example given, an Appendage II which approves the rollout of job steps with a higher priority than the requesting job step is used to illustrate appendage coding.

GENERAL FLOW OF ROLLOUT PROCESSING

The flowchart in Figure 9 depicts the overall flow of control through the various user appendages and the Rollout module.



A Request for Rollout from Top of the Rollout Queue
Is Rollout Currently Invoked for Other Jobstep(s) ?
No
Yes
IEAQAPG1
Coincident Rollout Appendage
Does Exit I Want Coincident Rollout ?
Yes
No
Attempt Logical Rollout
Queue Request by Priority on the Rollout Queue
Successful ?
Yes
No
Turn Off Hi Priority Pass Switch
Can PQE be Obtained from a Region Previously Rolled Out by Requestor ?
Yes
No
A
Turn Off Hi Priority Pass Switch
Set TCB Scan to Start at Top of TCB Queue and Stop at Requestor's JSTCB
IEAQAPG4
Criterion Selection Appendage
Set TCB Scan to Start at Requestors JSTCB and Stop at Last TCB on the TCB Queue
Does the PQE Pass User's Criteria
No
Yes
Is the PQE Already Rolled Out ?
Yes
No
Is Hi Priority Pass Switch On ?
Yes
No
Scan TCB Ready Queue for the Lowest Priority Candidate for Rollout
Perform Physical Rollout
IEAQAPG2
Hi Priority Pass Appendage
Any Candidate Found ?
No
Yes
Rollout I/O Error ?
Yes
No
Set TCB of this PQE Non-Rolloutable
Does Exit II Want a Hi Priority Pass ?
Yes
No
Can its PQE Satisfy the Request ?
Yes
No
Get Another PQE of the Same Jobstep
Successful Rollout Performed
IEAQAPG3
Abend Appendage
Any PQE's ?
Yes
No
Queue New PQE on Requesting Step
Step Cannot be Rolled Out
Does Exit III Want Requestor Abended ?
Yes
No
Set New Limit for Lowest Priority Candidate in TCB Scan
ABTERM Requestor
A
Does Exit III Want a Jobstep Abended ?
No
Yes
SVC Exit
Exit R0/R1 Module via Task Switch
ABTERM the Specified Jobstep

Figure 9. General Flow of Rollout/Rollin Processing

```
  SOURCE STATEMENT

IEAQAPG2 CSECT
┌──────────────────────────────────────────────────────────────────────┐
|THIS ROUTINE WILL APPROVE THE ROLLOUT OF JOBSTEPS WITH A HIGHER       |
|PRIORITY THAN THE REQUESTING JOBSTEP.  IT IS ENTERED FROM USER        |
|APPENDAGE - IEAQAPG2 - WHICH IS RESIDENT IN THE NUCLEUS AS PART OF THE|
|ROLLOUT/ROLLIN CODE.                                                  |
|                                                                      |
|IT WILL WRITE TO THE OPERATOR INDICATING THE FOLLOWING:               |
|  • ROLLOUT STATUS (NUMBER OF ROLLOUTS IN EFFECT AND THE NUMBER OF    |
|    ROLLOUT REQUESTS QUEUED.)                                         |
|  • THE NAME OF THE JOB REQUESTING ROLLOUT.                           |
|  • APPROVAL OF THE REQUEST.                                          |
└──────────────────────────────────────────────────────────────────────┘
R1       EQU    1
R2       EQU    2
R3       EQU    3
R4       EQU    4
R5       EQU    5
R8       EQU    8
R12      EQU   12
R13      EQU   13
R14      EQU   14
         STM   R14,R12,12(R13)
         BALR  R12,0
         USING *,R12
         LR    R14,R13
         ST    R13,SAVEAREA+4
         LA    R13,SAVEREA
         ST    R13,8(R14)
         LR    R2,0
         LR    R3,R1
         USING TCB,R3
         L     R4,TIOTA          GET ADDRESS OF TASK I/O TABLE
         USING TIOT,R4
         MVC   WTLENTER+27(8),JOBNAME
WTLENTER WTO   'IEAQAPG2 ENTERED            REQUESTS ROLLOUT'
         USING ROSTATUS,R2
         LH    R8,INEFFECT       GET NBR OF ROLLOUTS IN EFFECT
         CVD   R8,WORK
         UNPK  WTLEXIT+29(2),WORK
         LH    R8,QUEUED         GET NBR OF ROLLOUT REQUESTS QUEUED
         CVD   R8,WORK
         UNPK  WTLEXIT+51(2),WORK
         MVC   WTLEXIT+74(3),YES
WTLEXIT  WTO   'ROLLOUTS IN EFFECT -        ROLLOUTS QUEUED -   REQUEST
               APPROVED -    '
         L     R13,SAVEAREA+4
         LM    R14,R12,12(R13)
         LCR   R1,R1
         BR    R14
         DS    0D
SAVEAREA DS    18F
WORK     DS    FL8
YES      DC    C'YES'
ROSTATUS DSECT
INEFFECT DS    H
QUEUED   DS    H
TCB      DSECT
         ORG   *+12
TIOTA    DS    F
TIOT     DSECT
JOBNAME  DS    FL8
         END
```

Adding a Universal Character Set Image to the System Library

This chapter provides a detailed description of how to add either an IBM UCS character set image or a user-designed character set image to the SYS1.SVCLIB.

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

REFERENCE PUBLICATIONS

IBM 2821 Control Unit, Form A24-3312 contains the information necessary to create a user-designed chain/train.

IBM System/360 Operating System, Supervisor and Data Management Macro Instructions, Form C28-6647 describes the SETPRT macro instruction that loads a UCS image into the UCS buffer.

The IBM System/360 Operating System, Job Control Language, Form C28-6539 describes the UCS parameters that may be specified in a DD statement to load a UCS image into the UCS buffer at OPEN time.

How to Add a UCS Image to the System Library

The IBM standard character set images listed in the following table may be included in the SYS1.SVCLIB at SYSGEN time if a printer with the UCS feature is specified. (These character set images are described in IBM 2821 Control Unit, Form A24-3312.) The member name on the system library (SVCLIB) is developed by prefixing the character set code shown in the table with UCS1 (e.g., UCS1AN or UCS1YN).

AN	alphameric
HN	alphameric
PCAN	alphameric*
PCHN	alphameric*
PN	alphameric (PL1) PL/I
QNC	alphameric (PL1 PL/I - Commercial)*
QN	alphameric (PL1 PL/I - Scientific)*
RN	FORTRAN - COBOL - Commercial*
SN	text printing*
TN	text printing
XN	high speed alphanumeric
YN	high speed alphanumeric*
*preferred character set	

IBM Standard Character Set Codes

You may add a user-designed character image to the system library or make an existing image a default image by following these rules,

1. The member name must be the four characters UCS1 followed by a uniquely assigned character set code which is from 1-4 bytes long. This character set code may be any valid combination of letters and numbers according to the rules for ordinary symbols in the assembler language, except the single letter U or C which are abbreviations for special conditions recognized by the system.

2. This uniquely assigned character set code must be specified in the DD card or in the SETPRT macro instruction.

3. The first byte of the load module of a character image must specify if the image is a default image. A default image must have X'80' in the first byte. When loaded, the default image is used when you do not specify a UCS parameter in the DD statement. X'00' specifies that an image is not to be used as a default.

4. The second byte of the load module of a character image indicates the number of lines (n) to be printed for the image verification print-out. This corresponds to the number of times the basic character set is repeated in the image.

5. Each byte of the next n bytes indicates the number of characters to be printed on each line.

6. The 240 characters UCS image must follow the previously described fields. Two apostrophes and two ampersands must be used to represent an apostrophe or an ampersand respectively that is part of a character set image specified in a DC statement.

As an example to add the image YN to the system library, the following code may be used.

```
//ADDUCS     JOB  MSGLEVEL=1
//STEP       EXEC PROC=ASMFCL,PARM.ASM='NODECK,LOAD',                  X
//                PARM.LKED='LIST,NCAL,NE,OL'
//ASM.SYSIN DD    *
UCS1YN       CSECT
             DC   X'80'        (this is a default image)
             DC   AL1(6)       (number of lines to be printed)
             DC   AL1(39)      (39 characters printed on 1st line)
             DC   AL1(42)      (42 characters printed on 2nd line)
             DC   AL1(39)      (39 characters printed on 3rd line)
             DC   AL1(39)      (39 characters printed on 4th line)
             DC   AL1(42)      (42 characters printed on 5th line)
             DC   AL1(39)      (39 characters printed on 6th line)
             DC   C'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*..'
             DC   C'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*,.#-$'
             DC   C'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*,.'
             DC   C'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*,.'
             DC   C'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*,.#-$'
             DC   C'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*,.'
             END
/*
//LKED.SYSLMOD DD DSNAME=SYS1.SVCLIB(UCS1YN),DISP=OLD
```

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

The Shared Direct Access Device Option

This chapter describes the Shared Direct Access Device Option (Shared DASD) of the System/360 Operating System. It describes the functions of the option, its operating environment, and volume acceptability. Sections also explain operating procedures and data set considerations that the systems programmer must be aware of in using the option. An appendix to the chapter describes a procedure for finding unit control block addresses necessary for using the RESERVE macro instruction: it also shows an assembler language subroutine that issues a RESERVE and can be called by a higher level language.

IBM System/360 Operating System: Operator's Guide, Form C28-6540 provides information on operator responsibility when the Shared DASD facility is being used in a system; this should be read before using the Shared DASD option.

IBM System/360 Operating System: Concepts and Facilities, Form C28-6535 discusses the purposes of the Shared DASD facility.

IBM System/360 Operating System: Storage Estimates, Form C28-6551 provides information on the storage requirements for the option.

IBM System/360 Operating System: System Generation, Form C28-6554 explains how the option is included in a system.

IBM System/360 Operating System: Supervisor and Data Management Macro Instructions, Form C28-6647 provides information on the use of the DEQ macro instruction.

THE SHARED DASD OPTION

The Shared DASD option allows computing systems to share direct access storage devices. Systems can share common data and consolidate data when necessary; no change to existing records, data sets, or volumes is necessary to use the facility. However, reorganization of volumes may be desirable to achieve better performance. Briefly, the sharing is accomplished by a two-channel switch which allows a shared control unit to be switched between two channels from different systems. (With certain hardware configurations sharing between a maximum of four systems is possible.) The switching is controlled by program use of the RESERVE macro instruction which reserves a shared device or volume for the use of one system until it is freed by the program's issuing a DEQ macro instruction. During this time the device is under the exclusive control of the system which issued the RESERVE.

The Shared DASD facility can only be included in a system at system generation time. This facility is shown diagrammatically in Figure 10.

SYSTEM CONFIGURATION

The Shared DASD option can be used with any combination of PCP, MFT, and MVT configurations of the operating system. Identical operating system configurations are not necessary for systems to share devices unless they share the system data set SYS1.LINKLIB. The option requires no additional equipment except the two-channel switch or the IBM 2844 Auxiliary Storage Control unit, which does not require the two-channel switch. Any of your installation's applications data sets can be shared; SYSCTLG can be shared when it does not reside on a systems residence volume. The following system data sets cannot be shared:

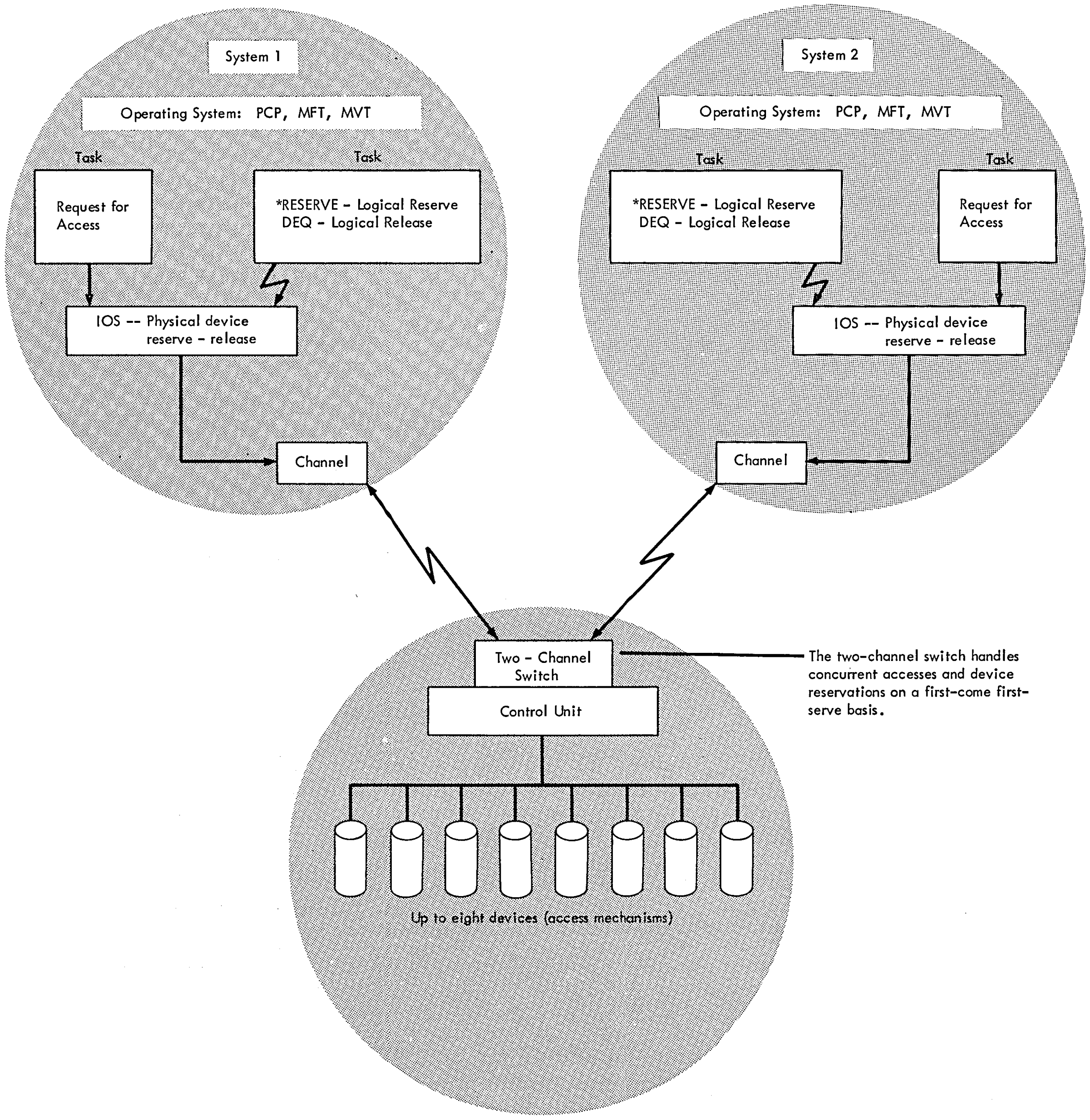
SYS1.SVCLIB	SYS1.SYSJOBQE
SYS1.NUCLEUS	PASSWORD data set
SYS1.LOGREC	SYSCTLG (on system residence volume)
SYS1.SYSLOGX (MVT only)	SYS1.ROLLOUT
SYS1.SYSLOGY (MVT only)	SYS1.ACCT

DEVICES THAT CAN BE SHARED

The following control units and devices are supported by the Shared DASD option:

1. IBM 2841 Storage Control Unit equipped with two-channel switch -- IBM 2311 Disk Storage Drive, 2303 Drum Storage, and 2321 Data Cell.
2. IBM 2314 Direct Access Storage Facility equipped with the two-channel switch -- IBM 2314 Disk Storage Module.
3. IBM 2314 Direct Access Storage Facility combined with the IBM 2844 Auxiliary Storage Control -- IBM Disk Storage Module. Device reservation and release are supported by this combination with or without the presence of the two-channel switch. Two channels -- one from System A and one from System B -- may be connected to the combination. In addition, the two-channel switch may be installed in either or both of the control units, thus permitting as many as four systems to share the devices. Only one channel to the device from each system is permitted.
4. IBM 2820 Control Unit with two-channel switch -- IBM 2301 Drum Storage.

Alternate channels to a device from any one system may only be specified for the IBM 2314 Direct Access Storage Facility.


System 1
Operating System: PCP, MFT, MVT
Task
Task
Request for Access
*RESERVE - Logical Reserve
DEQ - Logical Release
IOS -- Physical device reserve - release
Channel
System 2
Operating System: PCP, MFT, MVT
Task
Task
*RESERVE - Logical Reserve
DEQ - Logical Release
Request for Access
IOS -- Physical device reserve - release
Channel
Two - Channel Switch
Control Unit
Up to eight devices (access mechanisms)
The two-channel switch handles concurrent accesses and device reservations on a first-come first-serve basis.

* In multiprogramming systems (MFT, MVT), the RESERVE macro instruction also serializes use of the same resource between tasks in the system. In a single task (PCP) system, RESERVE effects device reservation only.

Figure 10. General Shared DASD Environment

VOLUME/DEVICE STATUS

The Shared DASD option requires that certain combinations of volume characteristics and device status be in effect for shared volumes or devices. One of the following combinations must be in effect for a volume or device:

	System A	Systems B,C,D
1.	Permanently resident	Permanently resident
2.	Reserved	Reserved
3.	Removable	Offline
4.	Offline	Removable or reserved

If a volume/device is marked removable on any one system, the device must be in off-line status on all other systems. The mount characteristic of a volume and/or device status may be changed on one system as long as the resulting combination is valid for other systems sharing the device. No other combination of volume characteristics and device status is supported or detected if present.

VOLUME HANDLING

Volume handling on the Shared DASD option must be clearly defined since operator actions on the sharing systems must be performed in parallel. You should make sure that operators understand the following rules when the Shared DASD option is in effect:

1. Operators should initiate all shared volume mounting and dismounting operations. The system will dynamically allocate devices unless they are in reserved or permanently resident status. Only the former of the two can be changed by the operator.

2. Mounting and dismounting operations must be done in parallel on all sharing systems. A VARY OFFLINE must be effected on <u>all</u> systems before a device may be dismounted.

3. Valid combinations of volume mount characteristics and device status for all sharing systems must be maintained. To IPL a system, a valid combination must be established before device allocation can proceed. This valid combination is established either by

   a. Specifying mount characteristics of shared devices in PRESRES (See the chapter "The PRESRES Volume Characteristics List.")

   b. Varying all sharable devices off line prior to issuing start commands and then following parallel mount procedures described in the chapter "How to Use the Shared DASD Option" in the <u>Operator's Guide</u> publication.

SHARING APPLICATION DATA SETS

As indicated previously, all application data sets can be shared, but you must give special consideration to the classification of these data sets. It is recommended that you classify your shared data sets as read only or read/write. A read-only data set may be read by all sharing systems but is never updated by them. A read/write data set may be read or written -- updated by all sharing systems. Read-only data sets are not reserved for the duration of their use; read/write data sets must be reserved for data set protection.

If a data set is seldom updated, but is read often, it is wise to classify it as read only. Minimizing reservation of devices will minimize the interference between systems.

A shared data set may be updated, effecting a device reservation for the write operation only, if the records being read are independent of each other. An example of such a data set with independent records is a private job library. Such a library may be reserved for the write operation only as long as members are not being deleted.

A system update time should be defined for updates to read-only data sets. For system update time the operator must vary offline, on all but one system, the device upon which the data set resides. Then the system update may be performed on the system to which that device is dedicated without any need to reserve the device. Processing of data sets by the linkage editor and utility programs constitutes update runs -- the data sets they process are regarded as read/write data sets. You may want to prepare a routine that will issue a RESERVE macro instruction, invoke the program to be executed, and issue a DEQ macro instruction after program execution.

There is no protection for shared data sets across job steps. That is, the RESERVE and DEQ for a data set must be done within each step (task); if devices are still reserved at the end of a task, device release is effected. Therefore, it is possible for one system to reserve a device and update a data set on that device between the execution of two steps in the other systems which are using that data set. There is no guarantee that a data set will remain unchanged between execution of steps.

RESERVING DEVICES

The RESERVE macro instruction is used to reserve a device for use by a particular system; it must be issued by each task needing device reservation. The RESERVE macro instruction protects the issuing task from interference by other tasks in the system. Each task issuing the RESERVE macro instruction must also use the DEQ macro instruction to release the device; two RESERVE instructions for the same resource without an intervening DEQ will result in an abnormal termination unless the second one specifies the keyword parameter RET=. (If a restart occurs when a RESERVE is in effect for devices, the system will not restore the RESERVE; the user's program must reissue the RESERVE.) Even if a DEQ is not issued for a particular device, termination routines in all operating system configurations will release devices reserved by a terminating task. The sample program described in the System Generation publication shows the use of the RESERVE and DEQ macro instructions. (In PCP configurations DEQ is treated as a NOP when used with ENQ; however, it is not a NOP when used with RESERVE.)

The Set-Must-Complete (SMC) parameter available with the ENQ macro instruction may also be used with RESERVE; this parameter is discussed in the chapter "The Must Complete Function of ENQ/DEQ."

The use of the RESERVE macro instruction is explained below:

$$\text{[symbol]}\quad \text{RESERVE}\quad \text{(qname address,rname address,}\begin{bmatrix}\underline{\text{E}}\\ \text{S}\end{bmatrix}\text{,}$$

$$\text{[rname length],SYSTEMS)}\left[\text{,RET=}\begin{Bmatrix}\text{TEST}\\ \text{USE}\\ \text{HAVE}\end{Bmatrix}\right]\text{,UCB=pointer address}$$

qname
: is the address in main storage of an eight-character name. Every task (within the system) issuing RESERVE against the same resource (data and device) must use the same qname-rname combination to represent the resource. The qname should not start with SYS.

rname address
: is the address in main storage of a name used in conjunction with the qname to represent the resource. The rname can be qualified, and may be 1 to 255 bytes in length.

$\begin{bmatrix}E\\S\end{bmatrix}$
: specify either exclusive control of the resource (E); or shared control with other tasks in the system (S). E is the default condition.

rname length
: is the length, in bytes, of rname. If omitted, the assembled length of rname is used. If zero (0) is specified, the length of rname must be contained in the first byte of the field designated by the rname address.

SYSTEMS
: specifies that the resource represented by qname-rname is known across systems as well as within the system whose task is issuing RESERVE, i.e., the resource is shared between systems.

RET=
: specifies a conditional request for all of the resources named in the RESERVE macro instruction. If the operand is omitted, the request is unconditional. The types of conditional requests are as follows:

  TEST
  : tests the availability status of the resources but does not request control of the resources.

  USE
  : specifies that control of the resources be assigned to the active task only if the resources are immediately available. If any of the resources are not available, the active task is not placed in a wait condition.

  HAVE
  : specifies that control of the resources is requested only if a request has not been made previously for the same task.

  Return codes are provided by the control program only if RET=TEST, RET=USE, or RET=HAVE is designated; otherwise, return of the task to the active condition indicates that control of the resource has been assigned to the task. Return codes are identical to those supplied by the ENQ macro instruction (see the Supervisor and Data Management Macro Instructions publication).

UCB=pointer address
: This keyword specifies either:

  1. The address of a fullword that contains the address of the Unit Control Block (UCB) for the device to be reserved.

  2. A general register (2-12) that points to a fullword containing the address of the unit control block for the device to be reserved.

The EXTRACT macro instruction is used to obtain the address of the task input/output table (TIOT) from which the UCB address can be obtained. The Appendix to this chapter explains some procedures for finding the UCB address.

To use the Shared DASD option in higher level languages, you may wish to write an assembler language subroutine to issue the RESERVE macro instruction. You should pass to this subroutine the following information: ddname, qname address, rname address, rname length, and RET parameter.

RELEASING DEVICES

The DEQ macro instruction is used in conjunction with RESERVE just as it is used with ENQ. It must describe the same resource and its scope must be stated as SYSTEMS; however, the UCB=pointer address parameter is not required. If the DEQ macro instruction is not issued by a task which has previously reserved a device, the system will free the device when the task is terminated.

PREVENTING INTERLOCKS

Certain precaution must be taken to avoid system interlocks when the RESERVE macro instruction is used. The more often device reservations occur in each sharing system, the greater the chance of interlocks occurring. Allowing each task to reserve only one device minimizes the exposure to interlock. The system cannot detect interlocks caused by program use of the RESERVE macro instruction and enabled wait states will occur on the system(s).

VOLUME ASSIGNMENT

Since exclusive control is by device, not by data set, you must consider which data sets reside on the same volume. In this environment it is quite possible for two tasks in two different systems -- processing four different data sets on two shared volumes -- to become interlocked. For example, data sets X_1 and X_2 reside on device X and data sets Y_1 and Y_2 reside on device Y. Task A in system A reserves device X in order to use data set X_1; task B in system B reserves device Y in order to use data set Y_1. Now task A in system A tries to reserve device Y in order to use data set Y_2 and task B in system B tries to reserve device X in order to use data set X_2. Neither can ever regain control and thus, will never complete normally. In a PCP or MFT environment, or in an MVT environment without job step timing, the job(s) should be canceled. In an MVT environment in which job step time limits are specified, the task(s) in the interlock would be abnormally terminated when the time limit expires. Moreover, an interlock could mushroom, encompassing new tasks as these tasks try to reserve the devices involved in the existing interlock.

Appendix

This appendix provides some procedures for finding the UCB address for use with the RESERVE macro instruction; it also shows a sample assembler language subroutine which issues the RESERVE and DEQ macro instructions and can be called by higher level languages.

PROVIDING THE UNIT CONTROL BLOCK ADDRESS TO RESERVE

The EXTRACT macro instruction is used to obtain information from the Task Control Block (TCB). The address of the TIOT can be obtained from the TCB in response to an EXTRACT in all configurations of the operating system. Prior to issuing an EXTRACT macro instruction, the user sets up an answer area in main storage which is to receive the requested information. One full word is required for each item to be provided by the control program. If the user wishes to obtain the TIOT address he must issue the following form of the macro instruction:

```
     EXTRACT answer-area address, FIELDS=TIOT
```

The address of the TIOT is then returned by the control program, right-adjusted, in the full word answer area.

The TIOT is constructed by job management routines and resides in main storage during step execution. The TIOT consists of one or more DD entries, each of which represents a data set defined by a DD statement for the jobstep. Each entry includes the DD name. Associated with each DD entry is the UCB address of the associated device. In order to find the UCB address, the user must locate the DD entry in the TIOT corresponding to the DD name of the data set for which he intends to issue the RESERVE macro instruction.

The UCB address may also be obtained via the DEB and DCB. The Data Control Block (DCB) is the block within which data pertinent to the current use of the data set is stored. The address of the Data Extent Block (DEB) is contained at offset 44 decimal after the DCB has been opened. The DEB contains an extension of the information in the DCB. Each DEB is associated with a DCB, and the two point to each other.

The DEB contains information concerning the physical characteristics of the data set and other information that is used by the control program. A device dependent section for each extent is included as part of the DEB. Each such extent entry contains the UCB address of the device to which (that portion of) the data set has been allocated. In order to find the UCB address the user must locate the extent entry in the DEB for which he intends to issue the RESERVE macro instruction. (In disk addresses of the form MBBCCHHR, the M indicates the extent number starting with 0.)

Following are suggested procedures for finding the UCB address of the device to be reserved.

If the data set is a multi-volume sequential data set, it must be assumed that all jobs will process that data set in a sequential manner starting with the first volume of the data set. In this case, by issuing a RESERVE for the first volume only, the user effectively reserves all the volumes of the data set.

For data sets using the queued access methods in the update mode or for unopened data sets:

1. Extract the TIOT from the TCB.

2. Search the TIOT for the DD name associated with the shared data set.

3. Add 16 to the address of the DD entry found in step 2. This results in a pointer to the UCB address in the TIOT.

4. Issue the RESERVE macro specifying the address obtained in step 3 as the operand of the UCB keyword.

For opened data sets:

1. Load the DEB address from the DCB field labeled DCBDEBAD.

2. Load the address of the field labeled DEBDVMOD in the DEB obtained in step 1. The result is a pointer to the UCB address in the DEB.

3. Issue the RESERVE macro specifying the address obtained in step 2 as the operand of the UCB keyword.

For BDAM data sets the user may reserve the device at any point in his processing in the following manner:

1. Open the data set successfully.

2. Convert the block address used in the READ/WRITE macro to an actual device address of the form MBBCCHHR. (A conversion method is discussed in the XDAP macro instruction section.)

3. Load the DEB address from the DCB field labeled DCBDEBAD.

4. Load the address of the field labeled DEBDVMOD in the DEB.

5. Multiply the "M" of the direct access address by 16.

6. The sum of steps 4 and 5 is the address of the correct extent entry in the DEB for the next READ/WRITE operation. The sum is also a pointer to the UCB address for this extent.

7. Issue the RESERVE macro specifying the address obtained in step 6 as the operand of the UCB keyword.

If the data set is an ISAM data set, QISAM in the load mode should be used only at system update time. Further, if it is a multi-volume ISAM data set, it must be assumed that all jobs will access the data set through the highest level index. The indexes should never reside in main storage when the data set is being shared. In this case, by issuing a RESERVE macro for the volume on which the highest level index resides, the user effectively reserves the volumes on which the prime data and independent overflow areas reside. The following procedures may be used to achieve this:

1. Open the data set successfully.

2. Locate the actual device address (MBBCCHH) of the highest level index. This address can be obtained from the DCB.

3. Load the DEB address from the DCB field labeled DCBDEBAD.

4. Load the address of the field labeled DEBDVMOD in the DEB.

5. Multiply the "M" of the actual device address located in step 2 by 16.

6. The sum of steps 4 and 5 is the address of the correct extent entry in the DEB for the highest level index not in core. This extent entry is also a pointer to the UCB address.

7. Issue the RESERVE macro specifying the address obtained in step 6 as the operand of the UCB keyword.

RES AND DEQ SUBROUTINES

The following assembler language subroutine may be used by FORTRAN, COBOL, or assembler language programs to issue the RESERVE and DEQ macro instructions. Parameters that must be passed to the RESDEQ routine if the RESERVE macro instruction is to be issued are

DDNAME
: the eight character name of the DDCARD for the device that you wish to reserve

QNAME
: an eight character name

RNAME LENGTH
: one byte (a binary integer) that contains the RNAME length value

RNAME
: a name from 1 to 255 characters in length

The DEQ macro instruction does not require the UCB=pointer address as a parameter. If the DEQ macro is to be issued, a full word of binary zeros must be placed in the DDNAME field before control is passed.

```
RESDEQ    CSECT
          SAVE   (14,12),T      SAVE REGISTERS
          BALR   2,0            SET UP ADDRESSABILITY
          USING  *,2
          ST     13,SAVE+4
          LA     11,SAVE       ADDRESS OF MY SAVE AREA IS STORED
          ST     11,8(13)      IN THIRD WORD OF CALLER'S SAVE AREA
          LR     13,11         ADDRESS OF MY SAVE AREA
          LR     9,1           ADDRESS OF PARAMETER LIST
          L      3,0(9)        DDNAME PARAMETER OR WORD OF ZEROS
          CLC    0(4,3),=F'0'  WORD OF ZEROS IF DEQ IS REQUESTED
          BE     WANTDEQ
*PROCESS FOR DETERMINING THE UCB ADDRESS USING THE TIOT
          XR     11,11         REGISTER USED FOR DD ENTRY
          EXTRACT  ADDRTIOT,FIELDS=TIOT
          L      7,ADDRTIOT    ADDRESS OF TASK I/O TABLE
          LA     7,24(7)       ADDRESS OF FIRST DD ENTRY
NEXTDD    CLC    0(8,3),4(7)   COMPARE DDNAMES
          BE     FINDUCB
          IC     11,0(7)       LENGTH OF DD ENTRY
          LA     7,0(7,11)     ADDRESS OF NEXT DD ENTRY
          CLC    0(4,7),=F'0'  CHECK FOR END OF TIOT
          BNE    NEXTDD
          ABEND  200,DUMP      DDNAME IS NOT IN TIOT, ERROR
```

```
FINDUCB   LA     8,16(7)       ADDRESS OF WORD IN TIOT THAT
*                              CONTAINS ADDRESS OF UCB
*PROCESS FOR DETERMINING THE QNAME REQUESTED
WANTDEQ   L      7,4(9)        ADDRESS OF QNAME
          MVC    QNAME(8),0(7)  MOVE IN QNAME
*PROCESS FOR DETERMINING THE RNAME AND THE LENGTH OF RNAME
          L      7,8(9)        ADDRESS OF RNAME LENGTH
          MVC    RNLEN+3(1),0(7) MOVE BYTE CONTAINING LENGTH
          L      7,RNLEN
          STC    7,RNAME       STORE LENGTH OF RNAME IN THE
*                              FIRST BYTE OF RNAME PARAMETER
*                              FOR RES/DEQ MACROS
          L      6,12(9)       ADDRESS OF RNAME REQUESTED
          BCTR   7,0           SUBTRACT ONE FROM RNAME LENGTH
          EX     7,MOVERNAM      MOVE IN RNAME
          CLC    0(4,3),=F'0'
          BE     ISSUEDEQ
          RESERVE (QNAME,RNAME,E,0,SYSTEMS),UCB=(8)
          B      RETURN
ISSUEDEQ  DEQ    (QNAME,RNAME,0,SYSTEMS)
RETURN    L      13,SAVE+4     RESTORE REGISTERS AND RETURN
          RETURN (14,12),T
          BCR    15,14
MOVERNAM  MVC    RNAME+1(0),0(6)
ADDRTIOT  DC     F'0'
SAVE      DS     18F
QNAME     DS     2F
RNAME     DS     CL256
RNLEN     DC     F'0'
          END
```

The Time Slicing Facility

This chapter describes the time slicing facility, a system generation option available with the MFT and MVT control programs of the IBM System/360 Operating System. Use of this facility allows the grouping of tasks of equal priority or partitions into a time-slice group so that each task within the group is limited to a fixed interval of CPU time each time it is given control. The facility is included in the system mainly to provide a method of controlling response time of a task.

Included in the chapter are a description of the facility, how it fits into the system, and the applications for which it is most effective. Other sections describe the prerequisite actions that must be taken, the use of the time slicing facility, and its operating characteristics.

IBM System/360 Operating System: Supervisor and Data Management Services, Form C28-6646 discusses task priority information that the system programmer must be aware of for effective use of this facility; it also provides a formula to derive dispatching priority from the job priority.

IBM System/360 Operating System: System Generation, Form C28-6554 describes the procedures to follow to include the facility in your system.

IBM System/360 Operating System: Job Control Language, Form C28-6539 discusses the CLASS and PRTY parameters of the JOB statement, which are used to invoke the facility.

IBM System/360 Operating System: Supervisor and Data Management Macro Instructions, Form C28-6647 discusses the ATTACH and CHAP macro instructions which can be used in MVT to change from one time-slice priority group to another; or from a task which is not a member of a time-slice group to one that is.

IBM System/360 Operating System: Operator's Guide, Form C28-6540 provides information on the messages and responses necessary to alter system generation specifications at system initialization time and, with MFT, at DEFINE time.

The Time Slicing Facility

The time slicing facility allows the user to establish a group of tasks (called the time-slice group) or partitions that are to share the use of the CPU, each for the same, fixed interval of time. When a member of the time-slice group has been active for the fixed interval of time, it is interrupted and control is given to another member of the group, which will, in turn, have control of the CPU for the same length of time. In this way, all member tasks are given an equal slice of CPU time, and no task or partition within the group can monopolize the CPU. In MVT only tasks in the group are time sliced, and they are time sliced only when the priority level of the group is the highest priority level that has a ready task. Dispatching of tasks continues within the group until

1. All tasks are in a waiting state, or

2. A task of higher priority than the one assigned to the group becomes ready.

In MFT, only partitions that are assigned to the time-slice group will be time-sliced, and they are time sliced only when the first partition in the group is the highest-priority ready task. Dispatching of the partitions continues within the group until all the partitions are in a waiting state, or until a partition with a higher priority is in a ready state.

The group of tasks to be time sliced (selected by priority or partition range) and the length of the time slice are specified by the installation at system generation time. This can be modified in MVT at system initialization time and in MFT through the DEFINE command. Any task or partition in the system that is not defined within the time-slice group is dispatched under the current priority structure; that is, the task or partition is dispatched only when it is the highest priority ready task or partition on the TCB queue.

SYSTEM CONFIGURATION AND SYSTEM RELATIONSHIPS

The time slicing facility can be used with any MFT or MVT configuration of the IBM System/360 Operating System. The time slicing facility is especially useful in a graphics environment or in any application of a conversational nature where concurrent tasks may involve conversation between the user and the problem program through a terminal. Establishing a time-slice group within this environment enables those tasks to be performed with a uniform response time.

PREREQUISITE ACTIONS

Time slicing is specified in the TMSLICE parameter of the CTRLPROG system generation macro instruction. The group(s) of tasks or partitions to be time sliced and the length of the time slice are specified in this parameter.

In MVT, a job priority defines the tasks that are to be time sliced. That is, all tasks that are executed in the system at the specified priority are to be time sliced. For example, time slice groups for MVT might be specified during system generation, as follows:

```
     CTRLPROG     TYPE=MVT
                  TMSLICE=(13,SLC-100,7,SLC-500)
```

In this example, two time-slice groups are defined. All jobs running at job priority 13 will be members of a time-slice group and will each have a slice of 100 milliseconds. All jobs running at job priority 7 will be members of a time-slice group and will each have a slice of 500 milliseconds. (See the section "Using the Time Slicing Facility" for a discussion of _job_ and _dispatching_ priority.)

In MFT, a group of contiguous partitions defines the time-slice group. All tasks scheduled into those partitions are time sliced and are treated as though they had the same dispatching priority. In MFT, only _one_ group of tasks can be specified to be time sliced. For example, a time-slice group for MFT might be specified during system generation, as follows:

```
    CTRLPROG     TYPE=MFT
                 TMSLICE=(P4-P6,SLC-256)
```

In this example, partitions P4, P5, and P6 make up the time-slice group and are assigned a time slice of 256 milliseconds for each and every task executing in these partitions.

System Initialization Time

If time slicing has been selected during system generation, the group (or group*s* in MVT) of tasks to be time sliced and the length of the time slice can be modified during system initialization. In MVT, the modifications are limited by the number of groups specified during system generation. The values specified at system initialization supersede all those specified during system generation. (NOTE: If the operator communication is desired during NIP, the parameter OPTIONS= COMM must be specified in the SUPRVSOR system generation macro instruction.)

In MFT, modifications to the time-slicing specifications are made in much the same way as other partition modifications. At system initialization, changes can be indicated by replying 'YES' to the message: 'IEE801D CHANGE PARTITIONS?'. After system initialization, changes can be indicated through the DEFINE command. In both cases, changes are actually made by responding to the message: 'IEE002A ENTER DEFINITIONS' or IEE803A CONTINUE DEFINITION' with the new TMSL reply. With this reply, the operator can request a list of current time-slicing specifications, change the range of time-slicing partitions and the time interval, or cancel time-slicing specifications altogether.

In MVT, the time-slicing specifications can be modified at system initialization time or they can be cancelled altogether. The modification can be accomplished by means of a new TMSL parameter in response to the system message "SPECIFY SYSTEM PARAMETERS".

HOW TO INVOKE THE TIME SLICING FACILITY

In MFT, a task is assigned to a time-slicing partition through the CLASS parameter on the JOB statement. The position of the time-slicing partitions with respect to other partitions in the system determines when the time-slice tasks gain control of the CPU.

In MVT, if the priority specified in the PRTY parameter of the JOB statement is the same as the priority specified at system generation and/or NIP time, that job (or the task representing that job) will be time sliced.

TIME SLICING'S EFFECT ON THE ATTACH AND CHAP MACRO INSTRUCTIONS

In MVT new tasks can be introduced into a time-slice group through the use of the ATTACH and CHAP macro instructions, when the attaching or new priority selected is equal to that of a time-slice group. These new tasks conform to all the rules for time slicing.

The CHAP macro instruction may remove a task from a time-slice group. If it does, this terminates all that task's time-slice characteristics. The ATTACH macro instruction may create a task that is not a member of a time-slice group, even though the originating task was.

Using the Time Slice Facility

In MFT, the time slice group is composed of a group of contiguous partitions and all tasks scheduled into those partitions are time sliced. Also, each partition in the system is assigned to at least one job class. Since a job is scheduled into a partition according to the CLASS parameter on the JOB statement, careful consideration should be given to the job-class assignment in order to enable the user to control the use of time slicing at his installation. For example,

1. Partitions P0-P2 have been assigned as the time-slice partition

2. The partitions have been assigned the following job classes:

```
P0=G
P1=G
P2=G,D
P3=B
P4=B,C,D
```

In this example, the user can ensure that a job will be time sliced by specifying CLASS=G on the JOB statement. This specification guarantees that the scheduler will initiate the job only into a partition assigned to CLASS G, i.e., P0, P1, or P2. Since P0-P2 have been designated as time-slice partitions, that job will be time sliced.

<u>CAUTION</u>: Note that if the CLASS parameter of a job was D, the job may or may not be time sliced, depending on whether it is initiated into partition P2 or P4. See the <u>Operator's Guide</u> publication for information on warning the operator about such situations.

In MVT, the job priority number (0-13) is specified on the PRTY parameter of the JOB statement. This job priority number is used as the scheduling priority number (that is, this number determines the initiation sequence of jobs on the input work queue, within their job classes); the job priority number is also used to derive the dispatching priority number, that is, it is used to determine where a TCB is placed on the queue of ready TCBs. Any task that has a dispatching priority number equal to the time-slicing dispatching priority number will be a member of the time-slice group. A task can have a dispatching priority number equal to that of the time-slice group as a result of the PRTY parameter of the JOB statement or by specifying parameters on the ATTACH or CHAP macro instructions. You should remember that where job priorities differ by 1, corresponding dispatching priorities differ by 16. Therefore, if a job step uses CHAP to change from one priority time-slicing group to another group, it must change the dispatching priority by 16, not just by 1. A full discussion of task priorities and the formula to derive a dispatching priority from a job priority is found in the <u>Supervisor and Data Management Services</u> publication.

OPERATING CHARACTERISTICS

The time-slicing mechanism operates within the structure of the current dispatcher. A priority is assigned to a group of tasks that are to be time sliced. The time slicing occurs among the tasks in the group only when the priority level of the group is the highest priority level that has a ready task. Each task or partition in the group is dispatched for the specified time slice. The time slicing continues until either all tasks or partitions are waiting, or a task or partition of higher priority than that of the group becomes ready.

In both MFT and MVT, the dispatcher will recognize that a priority level is one that is being time sliced; it will determine which task or partition within the group is to be dispatched and then dispatch that task or partition for the maximum time interval. If the time slice task loses control prior to the expiration of its interval (because an implicit or explicit wait is issued, or because a higher priority task or partition becomes ready), the remainder of the time is not saved. That is, when control returns to the time-slice group, the next ready task or partition in the group is given control, not the interrupted task or partition.

EFFECT OF SYSTEM TASKS ON TIME-SLICE GROUPS

The time slicing option is included in the system mainly to provide a method of controlling response time of a task. However, since it is being implemented in a priority dispatcher, any task of a higher priority than that of the time-slice group will be dispatched first, if it is ready. Note also that the time-slicing mechanism applies only to the problem program priorities, 0-13. Priorities 14 and 15 are reserved for the system and cannot be time sliced. Therefore, the response time of a time-slice task can be affected by the processing of system tasks, such as Readers, Writers, Master Scheduler, etc., which will always run at a higher priority than the time-slice group. Therefore, to guarantee response time, the time slice group should be defined, with MFT, in the high priority partitions, or, with MVT, at a high dispatching priority.

In MFT configurations non-interactive jobs should not be run concurrently and time sliced since this may significantly decrease performance.

Graphic Job Processor Procedures

The Graphic Job Processor is an IBM-provided program that enables users to define and initiate jobs directly from the IBM 2250 Graphic Display Units. If your system includes the Graphic Job Processor, you must write cataloged procedures which are used in starting major parts of the program.

This chapter provides information on writing and cataloging GFX and GJP procedures; it also provides information on allocating space and cataloging data sets for GJP. A section explains how to write cataloged procedures to be invoked through the Graphic Job Processor. The preparation of accounting routines to be used with the Graphic Job Processor is explained.

IBM System/360 Operating System User's Guide for Job Control from the IBM 2250 Display Unit, Form C27-6933 provides a description of the Graphic Job Processor.

Initialization of the Operating System for GJP

To make the Graphic Job Processor available when requested by the system operator with a START GFX command, several initialization actions must be taken. These actions are

- Adding a cataloged procedure for the Graphics Interface Task (GFX) to the procedure library (SYS1.PROCLIB).
- Adding a Graphic Job Processor cataloged procedure to the procedure library (SYS1.PROCLIB) for each 2250 display unit that is to be used with GJP.
- Allocating space for data sets that are required for each 2250 display unit to be used with GJP and cataloging these data sets on any convenient system volume.

The GFX and GJP cataloged procedures may be added to the procedure library (SYS1.PROCLIB) either before or after system generation using the IEBUPDTE utility program. Before system generation, the procedures must be added to the procedure library (SYS1.PROCLIB) of the starter system. After system generation, the procedures are added directly to SYS1.PROCLIB on the new system. Similarly, the space allocations and cataloging of the data sets for each 2250 may be added to any convenient system volume either before or after system generation using the IEHPROGM utility program. It is usually more convenient to perform these initializations after system generation.

CATALOGING GFX AND GJP PROCEDURES

The GFX cataloged procedure consists of an EXEC statement and several DD statements. The exact number of DD statements depends on the number of 2250 display units that may use the Graphic Job Processor. The name of the procedure must be GFX.

The following is the coding for the GFX cataloged procedure that you must provide on SYS1.PROCLIB:

```
//GFXEXEC   EXEC  PGM=IKAGFX,REGION=10K
//GJPnnn    DD    DSNAME=SYS1.JCLnnn,UNIT=SYSDA,DISP=SHR
             .
             .
             .
      (DD statement in the format above for each 2250 to be used)
             .
             .
             .
//SYSABEND  DD    SYSOUT=z
```

Where nnn is the address of the display unit being defined, and z is the output class to which printed output is assigned for abnormal terminations. A separate DD statement is required for each 2250 to identify the JCL data set for that device. In an MFT configuration the REGION parameter is ignored and the GFX Task is executed in a partition whose size is 10K or larger.

The GJP cataloged procedure consists of an EXEC statement and 11 DD statements. A separate cataloged procedure is required for each 2250 display unit that may use the Graphic Job Processor. The procedure name for each procedure must be in the form GJPnnn, where nnn is the address of a specific display unit.

The following is the coding for each GJP cataloged procedure. (Three separate procedures would be required if three display units were desired. The address used in the procedure name must be the same as that specified on the DD statements for the GFX cataloged procedure.)

```
//GJPEXEC  EXEC  PGM=IKAGJP,REGION=60K
//GJP2250  DD    UNIT=nnn
//GJPDIA   DD    DSNAME=SYS1.DIAnnn,UNIT=SYSDA,DISP=(OLD,KEEP)
//GJPEXT   DD    DSNAME=SYS1.EXTnnn,UNIT=SYSDA,DISP=(OLD,KEEP)
//GJPEXT1  DD    DSNAME=SYS1.EXTnnnA,UNIT=SYSDA,DISP=(OLD,KEEP)
//GJPJCL   DD    DSNAME=SYS1.JCLnnn,UNIT=SYSDA,DISP=SHR
//GJPPROC  DD    DSNAME=SYS1.PROCLIB,UNIT=SYSDA,DISP=SHR
//IEFPDSI  DD    DSNAME=SYS1.PROCLIB,UNIT=SYSDA,DISP=SHR
//GJPOUT   DD    SYSOUT=z
//SYSABEND DD    SYSOUT=z
//IEFRDER  DD    DUMMY
//IEFDATA  DD    UNIT=SYSDA,SPACE=(80,(500,500),RLSE,CONTIG),        X
//               DCB=(BUFNO=2,LRECL=80,BLKSIZE=80,RECFM=F,BUFL=80)
```

Where nnn is the address of the specific display unit to be used, and z is the output class to which printed output is assigned for abnormal terminations. The 60K value in the REGION parameter is the minimum size region or partition that may be specified; larger values are permissible. In an MFT configuration the REGION parameter is ignored and GJP is executed in a partition whose size is 60K or larger.

Note: In the //IEFDATA DD statement, the user may vary the SPACE requirements depending on the amount of SYSIN data that will be entered on the ENTER DATA and DESCRIBE DATA frames. If data will not be entered on the 2250, a DUMMY parameter may be used as follows:

```
//IEFDATA   DD   DUMMY
```

The following sample coding could be used to catalog both the GFX and GJP cataloged procedures in the procedure library (SYS1.PROCLIB) after system generation using the IEBUPDTE utility program. The example assumes that two 2250 display units will use GJP. The use of the IEBUPDTE utility program is fully explained in IBM System/360 Operating System: Utilities, Form C28-6586.

```
//UPDATE    JOB
//          EXEC    PGM=IEBUPDTE,PARM=NEW
//SYSPRINT  DD      SYSOUT=A
//SYSUT2    DD      DSNAME=SYS1.PROCLIB,DISP=OLD
//SYSIN     DD      DATA
./          ADD     LIST=ALL,NAME=GFX,LEVEL=00,SOURCE=0
./          NUMBER  NEW1=10,INCR=10
            .
            .
            .
      (GFX  cataloged procedure)
            .
            .
            .
./          ADD     LIST=ALL,NAME=GJP1E0,LEVEL=00,SOURCE=0
./          NUMBER  NEW1=10,INCR=10
            .
            .
            .
      (GJP  cataloged procedure for first 2250)
            .
            .
            .
./          ADD     LIST=ALL,NAME=GJP1D3,LEVEL=00,SOURCE=0
./          NUMBER  NEW1=10,INCR=10
            .
            .
            .
      (GJP  cataloged procedure for second 2250)
            .
            .
            .
./          ENDUP
/*
```

CATALOGING AND ALLOCATING SPACE FOR DATA SETS USED BY GJP

Four data sets are required for each 2250 Display unit to be used with GJP. The name of these data sets must be SYS1.DIAnnn, SYS1.EXTnnn, SYS1.EXTnnnA, and SYS1.JCLnnn, where nnn is the address of the specific display unit. The data sets may reside on any convenient system volume. The space allocations and cataloging may be accomplished using the IEHPROGM utility program. The actual space allocations required depends on the users problem program. However, the following allocations are suggested for most graphics programs.

Data Set Name	Allocation
SYS1.DIAnnn	SPACE=(TRK,(3,3))
SYS1.EXTnnn	SPACE=(TRK,(5,5))
SYS1.EXTnnnA	SPACE=(TRK,(5,5))
SYS1.JCLnnn	SPACE=(TRK,(5,5))

The following sample coding could be used to allocate space and catalog the data sets using IEHPROGM. The use of the IEHPROGM utility program is fully explained in IBM System/360 Operating System: Utilities, Form C28-6586.

```
//jobstep  JOB
//STEP     EXEC PGM=IEHPROGM,PARM=NEW
//SYSPRINT DD   SYSOUT=A
//DIA1E0   DD   DSNAME=SYS1.DIA1E0,VOLUME=(,RETAIN,SER=111111),        X
//              UNIT=SYSDA,DISP=(,KEEP),SPACE=(TRK,(3,3))
//EXT1E0   DD   DSNAME=SYS1.EXT1E0,VOLUME=(,RETAIN,SER=111111),        X
//              UNIT=SYSDA,DISP=(,KEEP),SPACE=(TRK,(5,5))
//EXT1E0A  DD   DSNAME=SYS1.EXT1E0A,VOLUME=(,RETAIN,SER=111111),       X
//              UNIT=SYSDA,DISP=(,KEEP),SPACE=(TRK,(5,5))
//JCL1E0   DD   DSNAME=SYS1.JCL1E0,VOLUME=(,RETAIN,SER=111111),        X
//              UNIT=SYSDA,DISP=(,KEEP),SPACE=(TRK,(5,5))
           .
           .
           .
       (DD statements for other 2250s in the above format)
           .
           .
           .
//SYSIN    DD   *
         CATLG CVOL=SYSDA=111111,VOL=SYSDA=111111,DSNAME=SYS1.DIA1E0
         CATLG CVOL=SYSDA=111111,VOL=SYSDA=111111,DSNAME=SYS1.EXT1E0
         CATLG CVOL=SYSDA=111111,VOL=SYSDA=111111,DSNAME=SYS1.EXT1E0A
         CATLG CVOL=SYSDA=111111,VOL=SYSDA=111111,DSNAME=SYS1.JCL1E0
           .
           .
           .
       (CATLG statements for other 2250s in the above format)
/*
```

Writing Cataloged Procedures to be Invoked Through the Graphic Job Processor

A problem program that refers to the display unit either contains a DCB macro instruction for the display unit, or the data control block is generated as a result of statements written in a higher-level language. In addition, the operating system requires that a DD statement for the display unit be included in the job control statements for each job step associated with the display unit.

In writing cataloged procedures to be invoked through the Graphic Job Processor, the programmer should include DD statements for the display unit in the procedure as follows:

For a Single-Step Procedure: You should include a DD statement containing the parameter UNIT=unit name as the first DD statement following the EXEC statement, where "unit name" is either "2250-1" (for a 2250 Model 1 Display Unit) or "2250-3" (for a 2250 Model 3 Display Unit). The Graphic Job Processor replaces the "unit name" with the 3-digit unit address of the actual display unit at which the user is sitting. However, if you specify the 3-digit address of a particular display unit in the first DD statement, the Graphic Job Processor will not override the address.

If you do not provide a DD statement for the display unit as the first DD statement in a single-step procedure, the Graphic Job Processor creates a DD statement for the unit and inserts the 3-digit address of the display unit at which the user is sitting. The name field of the DD statement will contain a name in the form stepname.ddname, where ddname is either the system generation default or the name has been entered as a DISPLAY UNIT REFERENCE parameter on the SPECIFY JOB STEP frame.

For a Multi-Step Procedure: You should include a DD statement for the display unit in each step of the procedure. The DD statement in the first step should include the parameter UNIT=unit name, where "unit name" is either "2250-1" (for a 2250 Model 1 Display Unit) or "2250-3" (for a 2250 Model 3 Display Unit). The DD statement for the display unit in each succeeding step of the procedure should refer back to the statement in the first step by means of the DSNAME=*.stepname.ddname parameter.

To override the display unit DD statements in all steps of the procedure, you need only override the display unit DD statement in the first step of the procedure. However, to override the display unit DD statement in the second or a succeeding step of the procedure, you can employ the appropriate stepname.ddname combination in the name field of the statement.

Note that failure to provide display unit DD statements in a multi-step procedure, means the Graphic Job Processor creates such a statement for the first step of the procedure only as described in the single-step procedure above.

For additional information on overriding statements in cataloged procedures, see the publication IBM System/360 Operating System: Job Control Language, Form C28-6539.

Requesting Dumps: The Graphic Job Processor does not generate a SYSABEND DD statement for procedures invoked with GJP operations. Thus, if a dump is desired when the problem program is abnormally terminated, the programmer must include a SYSABEND DD statement in his procedure.

Preparation of User-Written Accounting Routines

An accounting routine receives control from the Graphic Job Processor when a user performs the LOG ON and LOG OFF operations. The accounting module in the distributed Graphic Job Processor is a dummy routine that performs no processing; the routine merely returns to the LOG ON and LOG OFF processors with a return code (4) that indicates a normal return. To perform accounting functions at LOG ON or LOG OFF, the user must write his own accounting routine following the conventions described below.

Entry to the Accounting Routine: The entry point of the accounting routine must be named IKAACCTG. This name is specified in either a CSECT statement or an ENTRY statement.

Input to the Accounting Routine: Bit 0 of register 1 is on (1) is entry to the accounting routine was from the LOG ON processor; bit 0 is off (0) if entry was from LOG OFF. Bits 8-31 of register 1 contain the address of a 28-byte parameter list, structured as follows:

Byte	
0	One-byte condition code for IKAACCTG.
1	Three-character unit address.
4	Address of a 20-byte area containing the user's name.
8	Address of a 20-byte area containing the account number.
12	Address of a 20-byte area containing other accounting information.
16	Address of a 20-byte area where the accounting routine can place data.
20	Address of a 72-byte area where the accounting routine can place a message to be displayed.
24	Address of a 72-byte area which contains the text entered on the LOG OFF frame.

The condition code mentioned above contains one of the following codes to indicate the condition of entry to IKAACCTG:

Hexadecimal Code	Meaning
00	This is the initial entry (for LOG ON or LOG OFF frame) to IKAACCTG.
04	The LOG OFF frame has been canceled.
08	The LOG OFF frame has been completed.

Output From the Accounting Routine: Upon return from the user's accounting routine, register 15 must contain a return code to indicate the results of the accounting routine processing. The codes that may be returned are as follows:

Hexadecimal Code	Meaning
0	Normal return -- Text for a message to be displayed on the frame has been provided in the 72-byte area. The 2250 user must perform the END function to acknowledge the message.
4	Normal return -- No message is to be displayed.
8	Error return -- Text for an error message to be displayed on the LOG ON frame has been provided in the 72-byte area. The 2250 user must correct the information and perform the END function again. The accounting routine will again receive control to perform a new check of the information.
C	Invalid user's name -- The Graphic Job Processor is to display an appropriate error message.
10	Invalid user's account number -- The Graphic Job Processor is to display an appropriate error message.

The user's accounting routine can also use the Write To Operator (WTO) or a Write To Operator With Reply (WTOR) macro instruction to write a message to the system operator.

Exit From the Accounting Routine: A RETURN macro instruction restores the contents of the registers and returns control to the Graphic Job Processor with the return code in register 15.

Inserting an Accounting Routine: The accounting routine can be inserted into the Graphic Job Processor either before or after the system generation process.

To insert an accounting routine before system generation, link edit it into the module library (SYS1.RC541), thereby replacing the existing module named IKAACCTG.

To insert an accounting routine after system generation, link edit the accounting routine with the IKAPLON0 and IKAPLOG0 modules. The Graphic Job Processor modules are in the linkage library (SYS1.LINKLIB). The linkage editor control statements necessary to insert the accounting routine in the IKAPLON0 and IKAPLOG0 modules are as follows (card input is assumed):

```
//jobname  JOB   parameters
//stepname EXEC  PGM=IEWL,parameters
//SYSPRINT DD    SYSOUT=A
//SYSOUT1  DD    UNIT=SYSDA,SPACE=parameters
//SYSLMOD  DD    DSNAME=SYS1.LINKLIB,DISP=OLD
//SYSLIN   DD    *
           .
           .
           .
      (accounting routine object deck)
           .
           .
           .
      INCLUDE SYSLMOD (IKAPLON0)
      ENTRY IKAPLON0
      NAME IKAPLON0(R)
           .
           .
           .
      (accounting routine object deck; identical to deck above)
           .
           .
           .
      INCLUDE SYSLMOD (IKAPLOG0)
      ENTRY IKAPLOG0
      NAME IKAPLOG0(R)
/*
```

Buffer Storage Considerations for 2250 Display Unit, Model 3

When two or more 2250 Model 3 display units are operated from the same IBM 2840 Display Control Unit, buffer storage is shared among the associated 2250 display units. Buffer storage is assigned to a specific display unit when the executing program issues an ASGNBFR macro instruction. Assignments are made in 256-byte increments (called sections) on a first come, first served basis. When requests for buffer storage are made, the sections are assigned from contiguous storage; if the number of requested sections are not available, no storage is assigned and a return code is provided to the requesting program.

Buffer sections may be reserved (guaranteed) for a particular display unit at system generation time with the NUMSECT operand of the IODEVICE macro instruction. Once assigned, guaranteed sections cannot be shared with other display units.

The Graphic Job Processor requires that seven buffer sections be available to the display unit. These sections are requested dynamically when GJP is initiated and are released before GJP transfers control. When seven sections are not available, GJP will modify its request and request one section. If one section is available, GJP will write a message to the 2250 user informing him that GJP is unable to execute because sufficient buffer storage is not available. If one section is unavailable, the 2250 screen will remain blank and the 2250 user will be unaware of the reason. However, a message about this condition will be written to the system operator.

To assure that the 2250 GJP user is provided with the minimal response of a message, at least one buffer section must be available, whether guaranteed or not. An installation may guarantee the availability of at least one buffer section at system generation time with the IODEVICE macro instruction. Unless an installation must ensure that GJP is started every time that it is requested from a 2250 Model 3, it is usually not advantageous to reserve more than one buffer section for each unit.

Satellite Graphic Job Processor Procedures

The Satellite Graphic Job Processor (SGJP) is a program that facilitates job control from a remote 1130/2250 subsystem. SGJP enables a user at an 1130/2250 subsystem (attached to a System/360 via a telecommunication line) to define and initiate jobs to be processed in the System/360. The jobs defined with SGJP can be run under the operating system independently or in conjunction with a related program in the 1130.

This chapter explains how to initialize the system for SGJP, how to write cataloged procedures to be invoked through SGJP, and how to write accounting routines for use with SGJP.

Writing Cataloged Procedures to be Invoked Through SGJP

The IBM System/360 Operating System treats an 1130/2250 subsystem as a data set. The operating system identifies the subsystem by the telecommunications line that links the subsystem to the System/360. A problem program that communicates with the subsystem contains either a DCB macro instruction for the telecommunications line, or the data control block is generated as a result of statements written in a higher-level language. In addition, the operating system requires that a DD statement for the subsystem be included in the job control statements for each job step that communicates with the subsystem.

In writing cataloged procedures to be invoked through SGJP, you should include DD statements for the subsystem as indicated in the following sections.

For a Single-Step Procedure: Include a DD statement containing the parameter UNIT=1130. In producing the final job control statement, SGJP replaces the unit name 1130 with the 3-digit unit address of the subsystem at which the user is located.

If the user fails to provide a DD statement for the subsystem or provides a DD statement containing the address of a particular telecommunications line (other than the line to the subsystem at which the user is located), SGJP creates a new DD statement in the form:

```
//stepname.lineref  DD  UNIT=address
```

where

stepname
    is the name of the job step in which the statement appears.

lineref
    is the default parameter provided in the LINEREF operand of the GJOBCTL macro instruction at system generation.

address
    is the 3-digit address of the telecommunications line to the subsystem at which the user is located.

For a Multi-Step Procedure: Include a DD statement for the subsystem in each step of the procedure that requires communication with the system. The DD statement in the first step should include the parameter UNIT=1130. The DD statement for the subsystem in each succeeding step should refer back to the statement in the first step by means of the DSNAME=*.stepname.ddname parameter.

This makes it easier for the user to override the subsystem DD statement in the procedure. To override the subsystem DD statements in all steps of the procedure, the user need override only the subsystem DD statement in the first step. To override the subsystem DD statement in the second or a succeeding step of the procedure, you can employ the appropriate stepname.ddname combination in the name field of the statement.

You should note that, if you fail to provide subsystem DD statements in a multi-step procedure, SGJP creates such a statement for the first step of the procedure only. The statement is in the form:

```
//stepname.lineref   DD  UNIT=address
```

as described above.

(For additional information on overriding statements in cataloged procedures, see the publication IBM System/360 Operating System: Job Control Language, Form C28-6539.)

PREPARATION OF USER-WRITTEN ACCOUNTING ROUTINES

An accounting routine receives control from the Satellite Graphic Job Processor when a user performs the LOG ON or LOG OFF operation. The accounting module in the distributed Satellite Graphic Job Processor is a dummy routine that performs no significant processing; the routine merely returns to the LOG ON or LOG OFF processor with a return code (04) indicating a normal return. To perform accounting functions at LOG ON or LOG OFF, you must write your own accounting routine following the conventions described below.

Entry to the Accounting Routine: The entry point of the accounting routine must be named IKAACCTG. This name is specified in either a CSECT statement or an ENTRY statement.

Input to the Accounting Routine: Bit 0 of register 1 is on (1) if entry to the accounting routine was from the LOG ON processor; bit 0 is off (0) if entry was from LOG OFF. Bits 8-31 of register 1 contain the address of a 28-byte parameter list, structured as follows:

Byte	
0	One-byte condition code for IKAACCTG.
1	Three-character unit address.
4	Address of a 20-byte area containing the user's name.
8	Address of a 20-byte area containing the account number.
12	Address of a 20-byte area containing other accounting information.
16	Address of a 20-byte area where the accounting routine can place data.
20	Address of a 72-byte area where the accounting routine can place a message to be displayed.
24	Address of a 72-byte area which contains the text entered on the LOG OFF frame.

The condition code mentioned above contains one of the following codes to indicate the condition of entry to IKAACCTG:

Hexadecimal Code	Meaning
00	This is the initial entry (for LOG ON or LOG OFF frame) to IKAACCTG.
04	The LOG OFF frame has been canceled.
08	The LOG OFF frame has been completed.

Output From the Accounting Routine: Upon return from your accounting routine, register 15 must contain a return code indicating the results of the accounting routine processing. The acceptable codes are:

Hexadecimal Code	Meaning
00	Normal return -- Text for a message to be displayed on the frame has been provided in the 72-byte area. The user must perform the END function to acknowledge the message.
04	Normal return -- No message is to be displayed.
08	Error return -- Text for a message to be displayed on the frame has been provided in the 72-byte area.
0C	Error return -- The name supplied by the user is invalid.
10	Error return -- The account number supplied by the user is invalid.

Your accounting routine can also use the Write To Operator (WTO) or the Write To Operator With Replay (WTOR) macro instruction to write a message to the system operator.

Exit From the Accounting Routine: A RETURN macro instruction restores the contents of the registers and returns control to the LOG ON or LOG OFF processor with the return code in register 15.

Inserting an Accounting Routine: The accounting routine can be inserted into the Satellite Graphic Job Processor either before or after system generation.

To insert an accounting routine before system generation, link edit it into the module library (SYS1.RC541), thereby replacing the existing module named IKAACCTG.

To insert an accounting routine after system generation, link edit the accounting routine into the IKAPLON0 and IKDPLOF0 modules. The Satellite Graphic Job Processor modules are in the link library (SYS1.LINKLIB). The linkage editor control statements necessary to insert the accounting routine in the IKAPLON0 and IKDPLOF0 modules are as follows:

```
//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 (IKAPLON0)
     ENTRY   IKAPLON0
     NAME    IKAPLON0(R)
                .
                .
                .
     (accounting routine object deck)
                .
                .
                .
     INCLUDE SYSLMOD (IKDPLOF0)
     ENTRY   IKDPLOF0
     NAME    IKDPLOF0(R)
/*
```

Initialization Requirements for the System/360 Operating System

To prepare the operating system for SGJP operations, the following initialization actions must be performed:

- A GFX cataloged procedure (which is used to start the GFX Task) must be added to the procedure library (SYS1.PROCLIB) unless the procedure has already been placed in the library for the Graphic Job Processor operations.

- An SGJP cataloged procedure (which is used to start an Initial Processor) must be added to the procedure library for each telecommunication line address that was included in the GJOBCTL system generation macro instruction. (Use of Initial Processors is optional in an MFT system. If use of Initial Processors is not specified, no SGJP cataloged procedures are required.)

- A GJP cataloged procedure (which is used to start the System/360 SGJP routines) must be added to the procedure library for each telecommunication line address that was included in the GJOBCTL system generation macro instruction.

- Space for four data sets must be allocated for each telecommunication line, and the data sets must be cataloged.

The cataloged procedures can be added to the procedure library either before or after system generation by using the IEBUPDTE utility program. If the cataloged procedures are added beforehand, they can be transferred to the procedure library during system generation. The alternative is to add the procedures directly to the procedure library after system generation.

Similarly, the space allocations and cataloging of data sets for each telecommunication line can be performed either before or after system generation by using the IEHPROGM utility program. It is usually more convenient to allocate space for and catalog the data sets after system generation.

The GFX Procedure

The GFX procedure is used to start the GFX Task when the system operator issues the START GFX command. The procedure consists of an EXEC statement, a series of DD statements, and a SYSABEND DD statement. One GFX procedure must exist on the procedure library.

The statements in the GFX procedure are shown and explained in Figure 11.

```
¹//GFXEXEC    EXEC  PGM=IKAGFX,REGION=10K
²//GJPnnn     DD    DSNAME=SYS1.JCLnnn,UNIT=SYSDA,DISP=SHR
               .
               .    (Additional DD statements.  One DD statement in
               .    the format shown above must be provided for each
               .    telecommunication line used for SGJP operations.)
               .
               .
³//SYSABEND   DD    SYSOUT=w
```

Note: The procedure must be named GFX.

[1]When the procedure is executed in an MFT system, the REGION parameter is ignored and the GFX Task is executed in a small partition.

[2]One DD statement in this format must be included for each telecommunication line that was specified for SGJP operations in the GJOBCTL system generation macro instruction. (For a description of the GJOBCTL macro instruction, see the publication IBM System/360 Operating System: System Generation, Form C28-6554.)

Each statement defines a data set (called the JCL data set) that will be used by GFX to pass system messages to the appropriate SGJP routines. The "nnn" in the ddname and in the data set name must be the 3-digit address of the telecommunication line for which the data set is being defined.

[3]This statement defines the system output class for printed output if the GFX Task is abnormally terminated. The "w" must be the alphabetic or numeric character that represents an output class for printed output. Any printed output class can be specified.

Figure 11. Statements in the GFX Cataloged Procedure

The SGJP Procedures

Upon receipt of a VARY ONGFX command containing the address of a telecommunication line, the operating system starts an Initial Processor (if specified) that will handle the first message received on that line. An SGJP cataloged procedure to be used in starting the Initial Processor (for that line) must be provided for each telecommunication line address included in the GJOBCTL system generation macro instruction.

The statements that must be included in each SGJP cataloged procedure are shown and explained in Figure 12. (These SGJP cataloged procedures are always required in an MVT system. They are only required in an MFT system if use of Initial Processors has been specified in the GJOBCTL system generation or in the START GFX command. For further information on use of Initial Processors, see the publication IBM System/360 Operating System and 1130 Disk Monitor System: User's Guide for Job Control From an IBM 2250 Display Unit Attached to an IBM 1130 System, Form C27-6938.)

```
1//SGJPEXEC  EXEC  PRG=IKDINPRO,REGION=10K
2//SYBSYS    DD    UNIT=nnn
3//SYSABEND  DD    SYSOUT=x
```

Note: Each procedure must be named SGJPnnn where "nnn" is the 3-digit address of the telecommunication line for which the procedure is being provided.

1When the procedure is executed in an MFT system, the REGION parameter is ignored and the Initial Processor is executed in the small partition.

2The statement defines the telecommunication line for which the procedure is being provided. The "nnn" is the 3-digit address of the line.

3This statement defines the print output class for printed output if the Initial Processor is abnormally terminated. The "x" must be an alphabetic or numeric character that represents an output class for printed output. This class may be the same or different from the abnormal termination output class specified in the GFX procedure.

Figure 12. Statements in the SGJP Cataloged Procedures

The GJP Procedures

A GJP procedure is required to start the System/360 SGJP routines for each telecommunication line. These routines are started after a message is received from the subsystem indicating that the 2250 user has completed the LOG ON frame. One GJP procedure must be provided for each telecommunication line address included in the GJOBCTL system generation macro instruction.

Each GJP procedure consists of an EXEC statement and 11 DD statements. The statements that must be included in each GJP cataloged procedure are shown and explained in Figure 13.

```
1//GJPEXEC    EXEC  PGM=IKDSGJP,REGION=60K                                   X
2//FT99F001   DD    UNIT=nnn
3//GJPDIA     DD    DSNAME=SYS1.DIAnnn,UNIT=SYSDA,DISP=(OLD,KEEP)
4//GJPEXT     DD    DSNAME=SYS1.EXTnnn,UNIT=SYSDA,DISP=(OLD,KEEP)
5//GJPEXT1    DD    DSNAME=SYS1.EXTnnnA,UNIT=SYSDA,DISP=(OLD,KEEP)
6//GJPJCL     DD    DSNAME=SYS1.JCLnnn,UNIT=SYSDA,DISP=(OLD,KEEP)
7//IEFPDSI    DD    DSNAME=SYS1.PROCLIB,UNIT=SYSDA,DISP=SHR
8//GJPPROC    DD    DSNAME=SYS1.PROCLIB,UNIT=SYSDA,DISP=SHR
9//GJPOUT     DD    SYSOUT=y
10//SYSABEND  DD    SYSOUT=z
11//IEFRDER   DD    DUMMY
12//IEFDATA   DD    UNIT=SYSDA,SPACE=(80,(500,500),,CONTIG),                  X
                    DCB=(BUFNO=2,LRECL=80,BLKSIZE=80,RECMF=F,BUFL=80
```

Note: Each procedure must be named GJPnnn where "nnn" is the 3-digit address of the telecommunication line for which the procedure is being provided.

[1]When the procedure is executed in an MFT system, the REGION parameter is ignored and the SGJP routines are started in a problem program partition.

[2]This statement defines the telecommunication line as a data set and associates the line with the partition or region in which the SGJP routines are being executed. The "nnn" in this statement and in succeeding statements must be the same as the 3-digit address in the name of the procedure.

[3]This statement defines a data set (called the Diary data set) used by the SGJP routines.

[4]This statement defines a data set (called the Extract data set) used by the SGJP routines.

[5]This statement defines a data set (called the Alternate Extract data set) used by the SGJP routines.

[6]This statement defines a data set (called the JCL data set) used by the SGJP routines.

[7]This statement defines the procedure library for use by the reader/interpreter (a component of the operating system).

[8]This statement defines the procedure library for use by the SGJP routines.

[9]This statement defines the system output class to which PRINTED RECORD output is to be assigned for jobs defined over this telecommunication line. The "y" must be an alphabetic or numeric character that represents an output class for printed output.

[10]This statement defines the system output class to which printed output is to be assigned if the SGJP routines for this telecommunication line are abnormally terminated. The "z" must be an alphabetic or numeric character that represents an output class for printed output. The output class can be the same or different from the one assigned in statement 9. If the same output class is assigned, PRINTED RECORD output and abnormal termination output will appear in the same printed listing.

[11]This statement is required by the operating system.

[12]This statement is required if the user wishes to enter SYSIN data from the ENTER DATA frame; the space requirements may be varied depending on the amount of SYSIN data that will be entered. If data will not be entered, a dummy parameter may be used as follows:

```
    //IEFDATA DD DUMMY
```

Figure 13. Statements in the Cataloged Procedure Used for Each Telecommunications Line Used With SGJP

CATALOGING THE PROCEDURES

The following sample coding could be used to catalog the GFX, GJP and SGJP cataloged procedures in the procedure library (SYS1.PROCLIB) after system generation, using the IEBUPDTE utility program. The example assumes that two telecommunication lines (with the addresses 024 and 025) will be used for SGJP operations. The use of the IEBUPDTE utility program is fully explained in the publication IBM System/360 Operating System: Utilities, Form C28-6586.

```
//UPDATE     JOB
//           EXEC     PGM=IEBUPDTE,PARM=NEW
//SYSPRINT   DD       SYSOUT=A
//SYSUT2     DD       DSNAME=SYS1.PROCLIB,DISP=OLD
//SYSIN      DD       DATA
./           ADD      LIST=ALL,NAME=GFX,LEVEL=00,SOURCE=0
             .
./           NUMBER   NEW1=10,INCR=10
             .        (GDX cataloged procedure)
             .
./           ADD      LIST=ALL,NAME=SGJP024,LEVEL=00,SOURCE=0
             NUMBER   NEW1=10,INCR=10
             .
             .        (SGJP cataloged procedure for first telecommunica-
             .        tion line)
./           ADD      LIST=ALL,NAME=SGJP025,LEVEL=00,SOURCE=0
             NUMBER   NEW1=10,INCR=10
             .
             .        (SGJP cataloged procedure for second telecommuni-
             .        cation line)
./           ADD      LIST=ALL,NAME=GJP024,LEVEL=00,SOURCE=0
./           NUMBER   NEW1=10,INCR=10
             .
             .        (GJP cataloged procedure for first telecommunica-
             .        line)
./           ADD      LIST=ALL,NAME=GJP025,LEVEL=00,SOURCE=0
             NUMBER   NEW1=10,INCR=10
             .
             .        (GJP cataloged procedure for second telecommunica-
             .        tion line)
./           ENDUP
/*
```

CATALOGING AND ALLOCATING SPACE FOR DATA SETS USED BY SGJP

Four data sets are required for each telecommunication line to be used with SGJP. The names of these data sets must be SYS1.DIAnnn, SYS1.EXTnnn, SYS1.EXTnnnA, and SYS1.JCLnnn, where "nnn" is the address of the specific display unit. The data sets may reside on any convenient system volume. The space allocations and cataloging may be accomplished by using the IEHPROGM utility program. The amount of space to be allocated depends on user job definition requirements. However, the following allocations are suggested for most graphics programs.

Data Set Name	Allocation
SYS1.DIAnnn	SPACE=(TRK,(10,5))
SYS1.EXTnnn	SPACE=(TRK,(20,5))
SYS1.EXTnnnA	SPACE=(TRK,(20,5))
SYS1.JCLnnn	SPACE=(TRK,(5,5))

The following sample coding could be used to allocate space and catalog the data sets using IEHPROGM. The use of the IEHPROGM utility program is fully explained in the publication IBM System/360 Operating System: Utilities, Form C28-6586.

```
//jobstep   JOB
//STEP      EXEC   PGM=IEHPROGM,PARM=NEW
//SYSPRINT  DD     SYSOUT=A
//DIA024    DD     DSNAME=SYS1.DIA024,VOLUME=(,RETAIN,SER=111111),    X
                   UNIT=SYSDA,DISP=(,KEEP),SPACE=(TRK,(10,5))
//EXT024    DD     DSNAME=SYS1.EXT024,VOLUME=(,RETAIN,SER=111111),    X
                   UNIT=SYSDA,DISP=(,KEEP),SPACE=(TRK,(20,5))
//EXT024A   DD     DSNAME=SYS1.EXT024A,VOLUME=(,RETAIN,SER=111111),   X
                   UNIT=SYSDA,DISP=(,KEEP),SPACE=(TRK,(20,5))
//JCL024    DD     DSNAME=SYS1.JCL024,VOLUME=(,RETAIN,SER=111111),    X
                   UNIT=SYSDA,DISP=(,KEEP),SPACE=(TRK,(5,5))
             .     (DD statements in the above formats to allocate
             .     space for the same data sets for each telecommuni-
             .     cation line to be used for SGJP operations.)
//SYSIN     DD     *
                   CATLG CVOL=SYSDA=111111,VOL=SYSDA=111111,
                   DSNAME=SYS1.DIA024
                   CATLG CVOL=SYSDA=111111,VOL=SYSDA=111111,
                   DSNAME=SYS1.EXT024
                   CATLG CVOL=SYSDA=111111,VOL=SYSDA=111111,
                   DSNAME=SYS1.EXT024A
                   CATLG CVOL=SYSDA=111111,VOL=SYSDA=111111,
                   DSNAME=SYS1.JCL024
             .     (CATLG statements in the above formats to catalog
             .     the data sets for the other telecommunication
             .     lines.)
/*
```

Index

C28-6550-5

YOUR COMMENTS PLEASE . . .

This publication is one of a series which serves as reference for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

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]

READER'S COMMENT FORM

IBM System/360 Operating System
System Programmer's Guide

Form C28-6550-5

- 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 ....

- 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)	☐ Deletion on page(s) ....
☐ Addition on page(s)	☐ Error on page(s) ....

Explanation:

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

C28-6550-5

YOUR COMMENTS PLEASE . . .

This publication is one of a series which serves as reference for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

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

Attention: Programming Systems Publications
Department D58

Fold Fold

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]