# IBM System/3 Model 6 Operation Control Language and Disk Utility Programs Reference Manual

Program Number 5703-SC1

GC21-7516-3

## Preface

This publication is intended for use by programmers who are doing either of the following:

1. Writing Operation Control Language (OCL) statements needed to run programs in the system.

2. Writing utility control statements necessary to run disk utility programs supplied by the system.

*Note:* In this publication there are some references to support of 24K and 32K bytes of main storage. A System/3 Model 6 with these main storage sizes is available only as an RPQ. Your IBM Marketing Representative can provide information about this.

### Prerequisite Publications

*IBM System/3 Model 6 Introduction,* GA21-9122

*IBM System/3 Model 6 System Programmer's Guide,* GC21-7530

### Other Publications Referenced in This Manual

*IBM System/3 Model 6 Operator's Guide,* GC21-7501

*IBM System/3 Disk Sort Reference Manual,* SC21-7522

*IBM System/3 Model 6 Conversational Utility Programs Reference Manual,* SC21-7528

*IBM System/3 Model 6 Utility Program for IBM 1255 Magnetic Character Reader Reference Manual,* SC21-7527

*IBM System/3 Model 6 RPG II Reference Manual,* SC21-7517

### Machine Requirements

Conversational OCL and all utility programs except Library Maintenance can be done using the minimum configuration of System/3 Model 6.

The minimum configuration is as follows:

- IBM 5406 Processing Unit, Model B2 (8K bytes) — including keyboard

- IBM 5444 Disk Storage Drive

- IBM 5213 Printer

OCL statements from cards and Library Maintenance functions involving cards require an additional unit: IBM 5496 Data Recorder, Model 1 with System/3 Model 6 Attachment Feature or 129 Card Data Recorder with Card Input/Output Attachment Feature.

**Fourth Edition (March 1973)**

This is a major revision of, and obsoletes, GC21-7516-2 and Technical Newsletters GN21-7622 and GN21-7646.
This manual has been completely revised and should be reviewed in its entirety.

This edition applies to version 08, modification 00 of the IBM System/3 Model 6 and to all subsequent versions and modifications 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 IBM System/3 Newsletter Order Number GN20-2228 for the editions that are applicable and current.

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

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Publications, Department 245, Rochester, Minnesota 55901.

# Contents

# How to Use This Manual

This publication contains two parts. Part I describes Operation Control Language (OCL) statements. Part II describes disk utility programs.

**Part I**

Refer to Part I if you want to know:

1. What an OCL statement is.

2. How to enter the OCL statements required to run your jobs.

**Part II**

Refer to Part II if you want to know:

1. What disk utility programs are supplied with the system.

2. The function of each disk utility program.

3. The Operation Control Language (OCL) statements and utility control statements necessary to request each disk utility program.

# PART I
# OPERATION CONTROL LANGUAGE

## Introduction to OCL

Before the IBM System/3 Model 6 can run a program, it must know what you want it to do and where to find the information it will need to do the job. You supply the what and where information in a series of *OCL (operation control language)* statements. The system can't run any of your programs unless each one is accompanied by a series of OCL statements. A series of OCL statements is called an *OCL cycle.* There are four OCL cycles: LOAD, BUILD, BUILDC, and CALL.

Part I of this manual is designed to help you select an OCL cycle and fill out the OCL guide sheets your operator will use in response to the OCL prompting for each job. You can either design an operator's OCL guide sheet for your installation or use the pre-printed form that is available (see *Operator's OCL Guide*).

### HOW TO USE PART I

The Conversational OCL section of this manual contains information on responding to OCL prompting. There are three levels of information for the four OCL cycles.

Here is how to use each level:

- Use the KEYWORD SEQUENCES for an overall understanding of the OCL cycle. The sequences show the order of the OCL keywords for a cycle and indicate which keywords require responses.

- Use the KEYWORD-RESPONSE SUMMARIES for a quick recall of all possible entries for each OCL statement. In the responses column of the summary charts:

  - Words or letters in all capital letters (FORMS, BUILD, R1) represent actual entries.

  - Words or letters not in all capital letters (mmddyy, Disk Name) represent information you must supply.

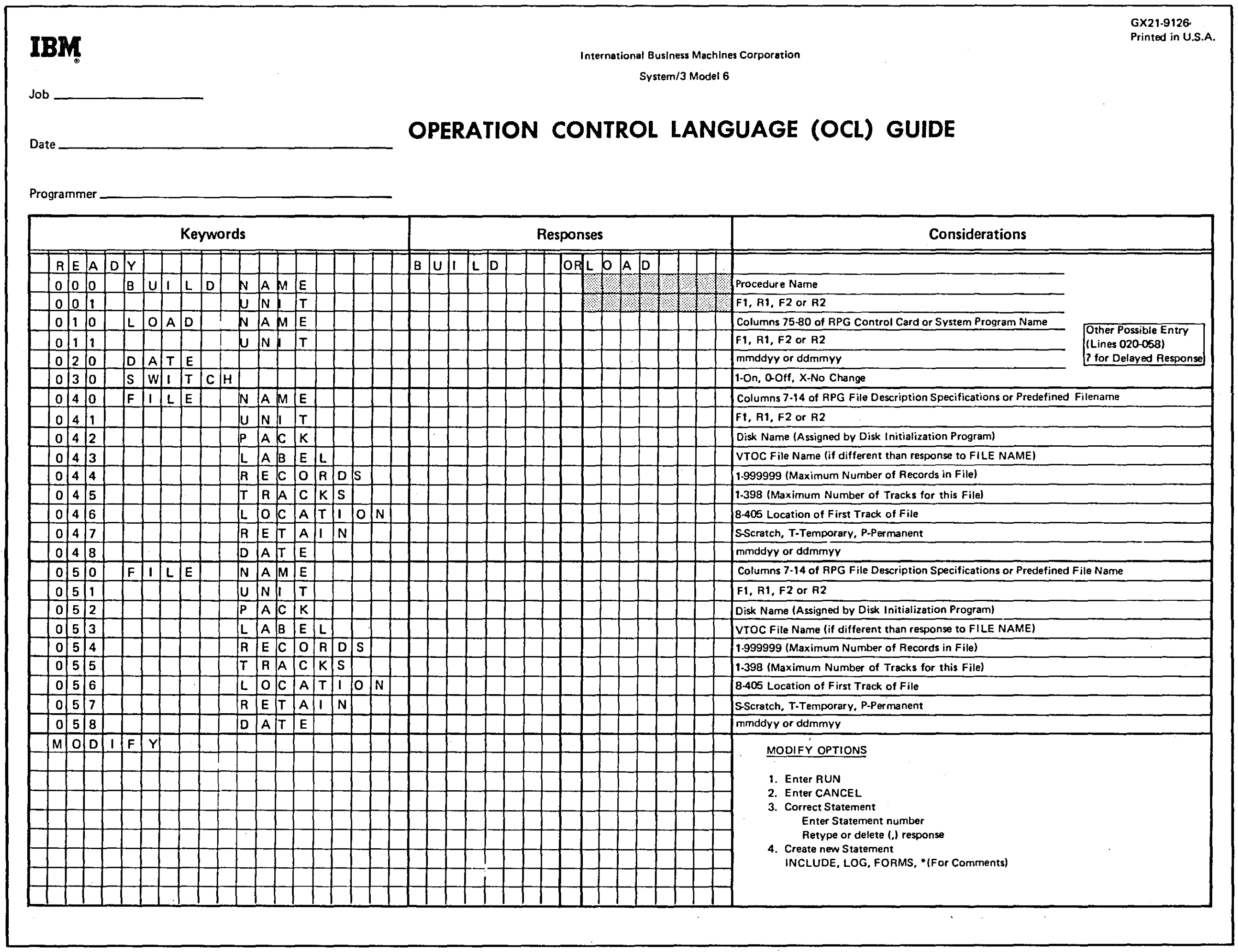
- Use the KEYWORD DESCRIPTIONS when you need a detailed explanation of a particular keyword.

The section titled *Using OCL* contains information on programming OCL for complex jobs and special features or devices.

## OPERATOR'S OCL GUIDE

The operator's OCL guide is available for you to use to tell your operator how to respond to the OCL prompting for a job. The CALL cycle is not included on the guide because the OCL prompting for that cycle is so short.

For information on filling out the OCL guide, see *IBM System/3 Model 6 System Programmer's Guide,* GC21-7530.



**IBM**

GX21-9126
Printed in U.S.A.

International Business Machines Corporation
System/3 Model 6

Job ______

Date ______

Programmer ______

**OPERATION CONTROL LANGUAGE (OCL) GUIDE**

| Keywords | | | Responses | Considerations |
|---|---|---|---|---|
| READY | | | BUILD OR LOAD | |
| 000 | BUILD | NAME | | Procedure Name |
| 001 | | UNIT | | F1, R1, F2 or R2 |
| 010 | LOAD | NAME | | Columns 75-80 of RPG Control Card or System Program Name |
| 011 | | UNIT | | F1, R1, F2 or R2 |
| 020 | DATE | | | mmddyy or ddmmyy |
| 030 | SWITCH | | | 1-On, 0-Off, X-No Change |
| 040 | FILE | NAME | | Columns 7-14 of RPG File Description Specifications or Predefined Filename |
| 041 | | UNIT | | F1, R1, F2 or R2 |
| 042 | | PACK | | Disk Name (Assigned by Disk Initialization Program) |
| 043 | | LABEL | | VTOC File Name (if different than response to FILE NAME) |
| 044 | | RECORDS | | 1-999999 (Maximum Number of Records in File) |
| 045 | | TRACKS | | 1-398 (Maximum Number of Tracks for this File) |
| 046 | | LOCATION | | 8-405 Location of First Track of File |
| 047 | | RETAIN | | S-Scratch, T-Temporary, P-Permanent |
| 048 | | DATE | | mmddyy or ddmmyy |
| 050 | FILE | NAME | | Columns 7-14 of RPG File Description Specifications or Predefined File Name |
| 051 | | UNIT | | F1, R1, F2 or R2 |
| 052 | | PACK | | Disk Name (Assigned by Disk Initialization Program) |
| 053 | | LABEL | | VTOC File Name (if different than response to FILE NAME) |
| 054 | | RECORDS | | 1-999999 (Maximum Number of Records in File) |
| 055 | | TRACKS | | 1-398 (Maximum Number of Tracks for this File) |
| 056 | | LOCATION | | 8-405 Location of First Track of File |
| 057 | | RETAIN | | S-Scratch, T-Temporary, P-Permanent |
| 058 | | DATE | | mmddyy or ddmmyy |
| MODIFY | | | | MODIFY OPTIONS<br>1. Enter RUN<br>2. Enter CANCEL<br>3. Correct Statement<br>Enter Statement number<br>Retype or delete (,) response<br>4. Create new Statement<br>INCLUDE, LOG, FORMS, *(For Comments) |

Other Possible Entry
(Lines 020-058)
? for Delayed Response

## Conversational OCL

Every job run on the Model 6 requires a set of Operation Control Language (OCL) statements to give the system information about the job to be run (such as what program to use, what files to use, what job date to use, etc.). An OCL statement consists of a keyword and a response.

The OCL for the Model 6 is called *conversational OCL* because a question and answer procedure is used. The system prints the question called a *keyword*, and the operator supplies the answer called a *response*. The keyword tells the operator the type of information required by the system. For example, the keyword FILE NAME indicates that the name of one file used in the program must be supplied. By printing a keyword, the system is *prompting* the operator for a response.

The operator responds to each keyword that applies to the job by typing in the relevant information. (When the system prompts FILE NAME, for example, the operator types the name of one file that the job uses.) If the system prompts a keyword that doesn't apply to the job, the operator bypasses the response.

### THE JOB CYCLE

The system will prompt READY when it is ready to run jobs. (For information on preparing the system to run jobs, see the *IBM System/3 Model 6 Operator's Guide,* GC21-7501.) The response to READY tells the system what type of OCL cycle you want to run.

There are four OCL cycles: LOAD, BUILD, BUILDC, and CALL. Of the four cycles, only the LOAD cycle is independent; that is, you can run a job by responding just to the keywords in that cycle. The other three cycles are interrelated; to run a job you must use two or more of them.

The OCL cycle you choose to use should be based on frequency of program use and whether the program will be run alone or with a group of programs.

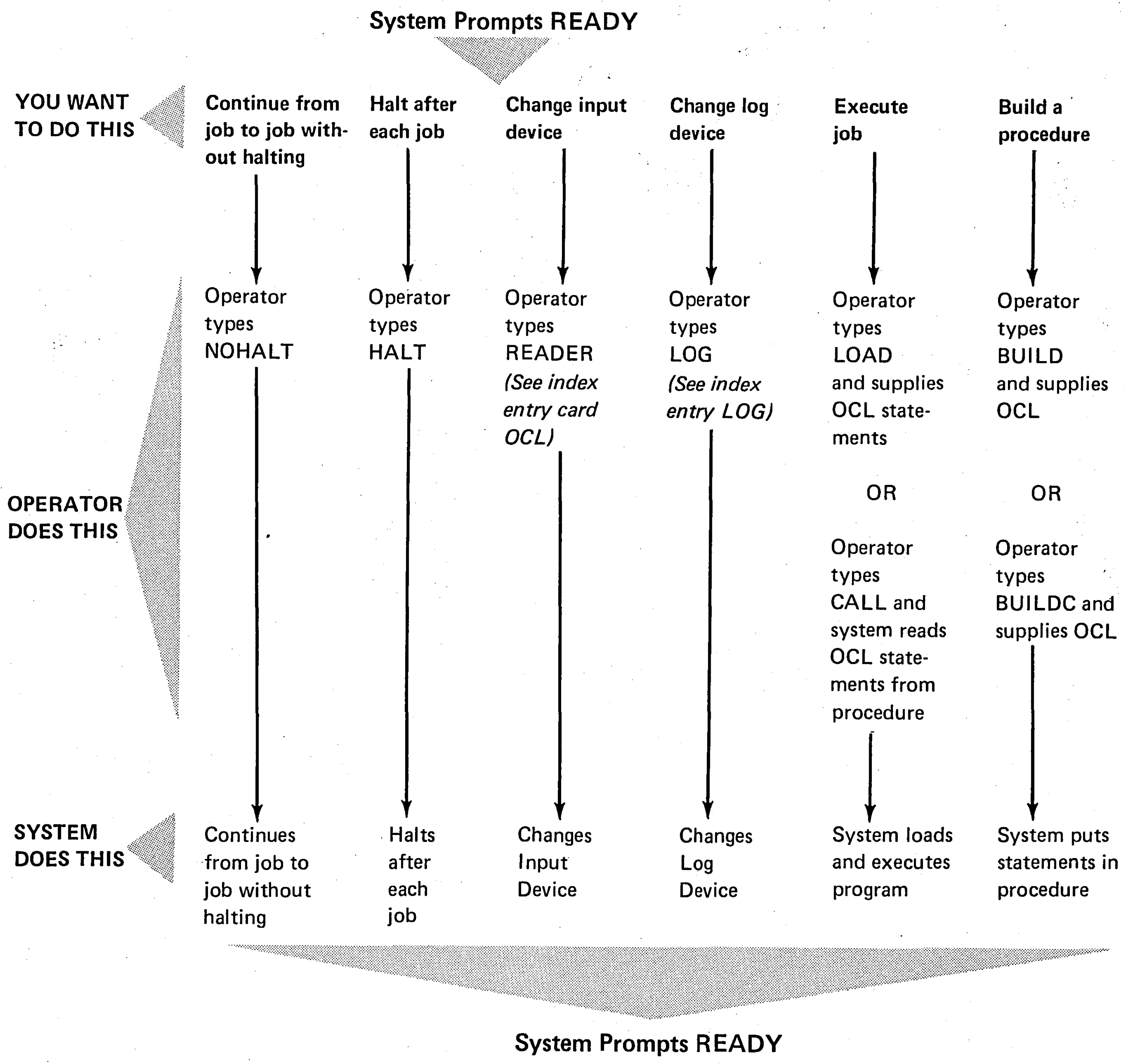
For infrequent jobs use:

| | |
|---|---|
| LOAD | This provides the OCL statements needed to run the job. |

For frequent jobs use one of these:

| | |
|---|---|
| BUILD | This puts the OCL statements for a job into a source library procedure. |
| BUILDC | This chains the procedures. |
| CALL | This calls a procedure from the source library. |

*Note:* A set of OCL statements in a source library is called a procedure.


System Prompts READY
YOU WANT TO DO THIS
Continue from job to job without halting
Halt after each job
Change input device
Change log device
Execute job
Build a procedure
OPERATOR DOES THIS
Operator types NOHALT
Operator types HALT
Operator types READER (See index entry card OCL)
Operator types LOG (See index entry LOG)
Operator types LOAD and supplies OCL statements
OR
Operator types CALL and system reads OCL statements from procedure
Operator types BUILD and supplies OCL
OR
Operator types BUILDC and supplies OCL
SYSTEM DOES THIS
Continues from job to job without halting
Halts after each job
Changes Input Device
Changes Log Device
System loads and executes program
System puts statements in procedure
System Prompts READY

### The LOAD Cycle

When you use a LOAD cycle, you're telling the system:

1. Here are the OCL statements for my program.
2. Go to the disk drive I specify and find the program I want to run.
3. Load the program into the processing unit.
4. Run my program.

The LOAD cycle OCL statements are not saved. If you want to run the same job again, your operator must respond to all the keywords in the LOAD cycle again. It's best to use the LOAD cycle for jobs you run infrequently because this cycle has many keywords and takes quite a while for responses.

### The BUILD Cycle

When you use a BUILD cycle, you're telling the system:

1. Here are the LOAD cycle OCL statements for job xxxx.
2. Store the LOAD cycle statements on disk so that they can be used whenever I want to run the program.
3. Do not run the program now.

Once the set of OCL statements is written on a disk, the set of statements is referred to as a *procedure.* The process of writing the statements on the disk is referred to as *building a procedure.* You use the BUILD cycle to build a procedure.

Although the BUILD cycle is the longest of all the OCL cycles in terms of operator time required, it doesn't run a job. Its function is to save the OCL statements for a job by writing them on one of the disks. The advantage of the BUILD cycle is that once the OCL statements are stored on the disk, the program can be run using them rather than by keying all the required statements.

#### *Delayed Responses in the BUILD Cycle*

Responding to a keyword by typing a question mark is referred to as a *delayed response.* Delayed responses are valid only in the BUILD cycle and only after keywords that contain a delayed response in the keyword-response chart (see *Keyword-Response Summary – Build Cycle*). A delayed response to any of these BUILD keywords will do the following:

- Cause the system to reprompt the keyword during the CALL cycle.
- Force the operator to respond to the keyword when it is reprompted during the CALL cycle. (The system won't continue the CALL prompting cycle until the operator types a valid response.)

#### *Control Statements in Procedures*

OCL statements that control the entering of other OCL statements are not valid in procedures. These statements, HALT, NOHALT, and LOG, are ignored when read from procedures during the CALL cycle and are not put into a procedure during a BUILD cycle.

### The BUILDC Cycle

When you use a BUILDC cycle, you're telling the system:

1. I want to prepare a procedure to run a series of jobs which are always executed one after the other with no interruption.
2. The OCL statements for each job in the group are in procedures stored on disk.
3. Here are the names and disk drive locations of the procedures for each job in the group.
4. Build a chained procedure, establishing a sequence in which the individual procedures are run.

A *chained procedure* is a list of the procedures for each job in a group, in the order you want to run them. The list contains:

1. The name of the procedure for each job.
2. The disk drive on which the procedure is located.

The process of writing the list on a disk is referred to as *building a chained procedure.* BUILDC stands for build chained.

### The CALL Cycle

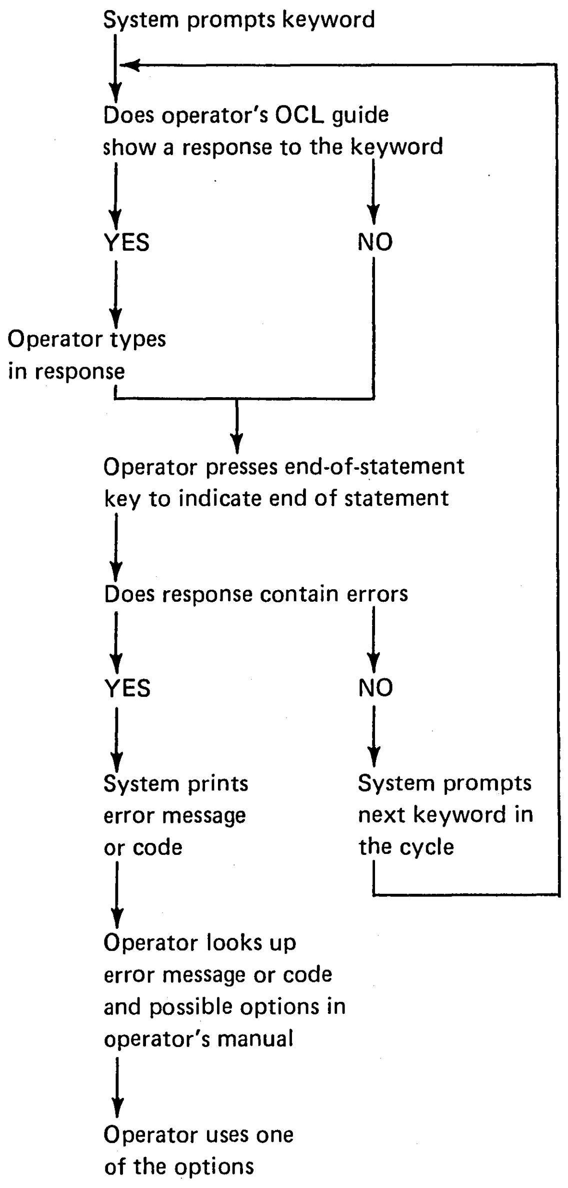
CALL is the shortest OCL cycle, having only four keywords. When you use a CALL cycle, you're telling the system:

1. Locate, on disk, the procedure I built for job xxxx.
2. Use it to run job xxxx.

The CALL cycle is always linked to a BUILD or a BUILDC cycle.

## SYSTEM-OPERATOR INTERACTION DURING KEYWORD PROMPTING

The system analyzes the operator's response to each keyword. If the response contains a formatting error (such as invalid characters or duplicate procedure names), the system prints an error message and reprompts the keyword. If the operator does not know the correct response, entering /* as a response to any prompt will cancel the job and cause READY to be prompted.


System prompts keyword
Does operator's OCL guide show a response to the keyword
YES
NO
Operator types in response
Operator presses end-of-statement key to indicate end of statement
Does response contain errors
YES
NO
System prints error message or code
System prompts next keyword in the cycle
Operator looks up error message or code and possible options in operator's manual
Operator uses one of the options

## Keyboard

Command Key Lights

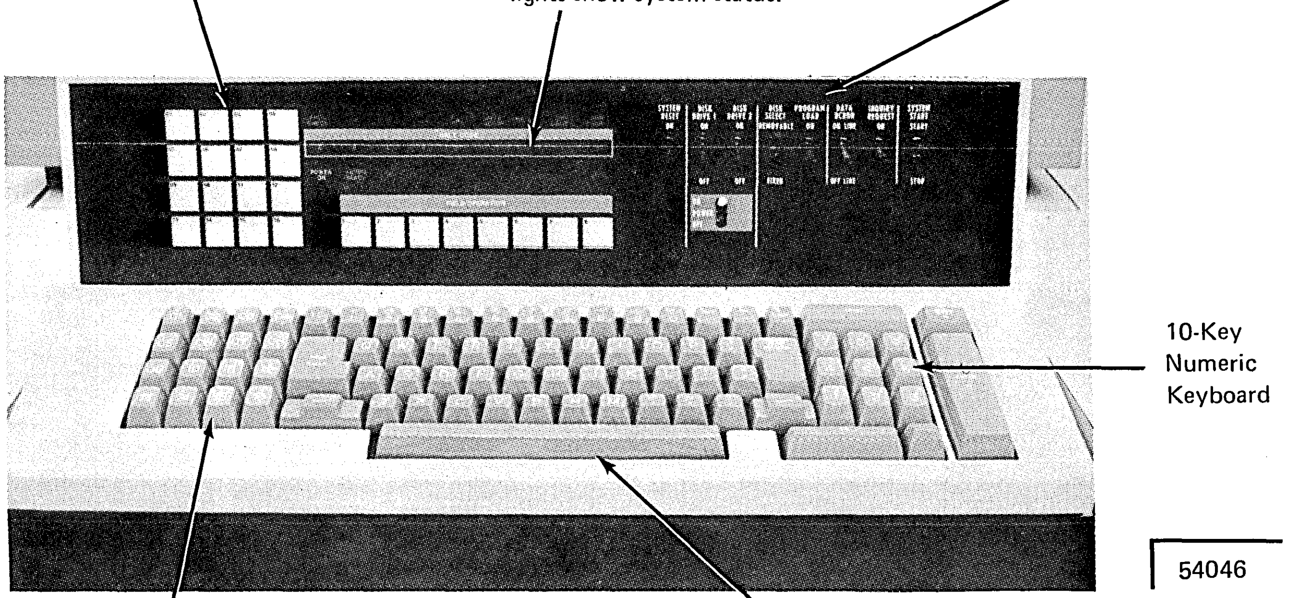
These lights tell the operator which command keys have been turned on.

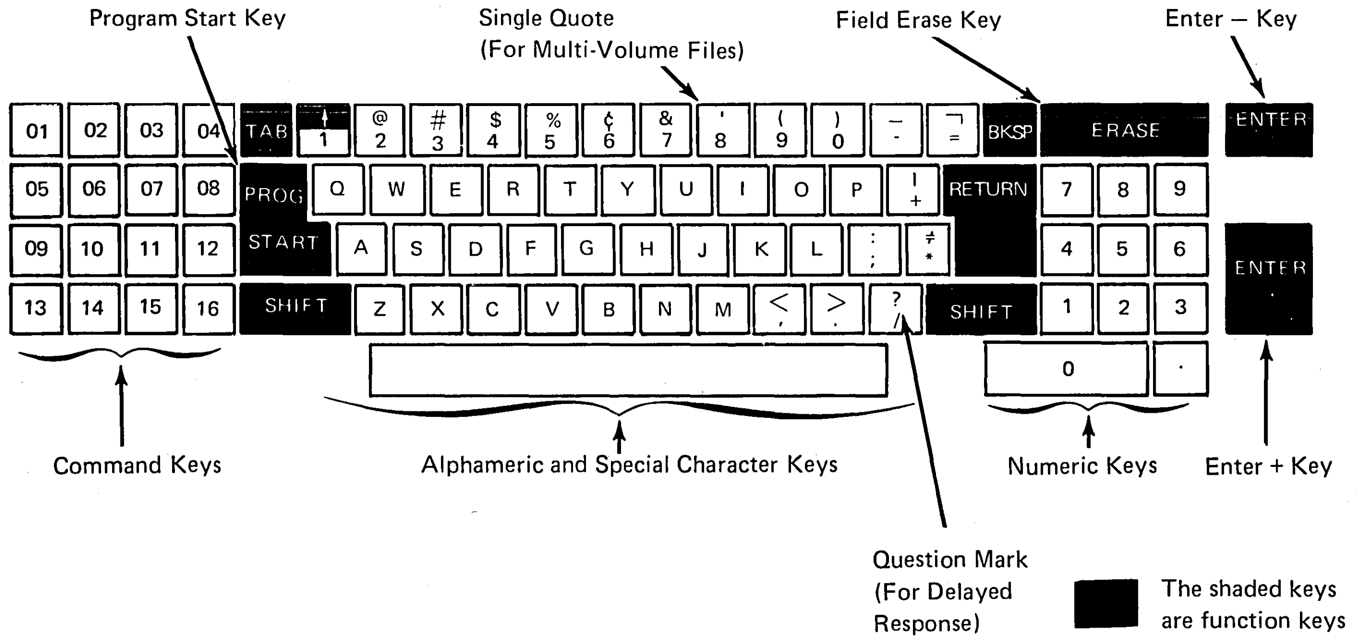
System Status Lights

I/O attention lights indicate those devices that need operator attention. The halt code and field/operation lights show system status.

System Control Switches

These switches start and control the system.




Program Start Key
Single Quote (For Multi-Volume Files)
Field Erase Key
Enter – Key
01 02 03 04 TAB 1 @ 2 # 3 $ 4 % 5 ¢ 6 & 7 ' 8 ( 9 ) 0 _ - ¬ = BKSP ERASE ENTER
05 06 07 08 PROG Q W E R T Y U I O P ! + RETURN 7 8 9
09 10 11 12 START A S D F G H J K L : ; " * 4 5 6 ENTER
13 14 15 16 SHIFT Z X C V B N M < , > . ? / SHIFT 1 2 3
0 .
Command Keys
Alphameric and Special Character Keys
Numeric Keys
Enter + Key
Question Mark (For Delayed Response)

■ The shaded keys are function keys

### End-Of-Statement Keys

The operator must respond to each keyword that the system prompts. The operator responds to a keyword by typing the required information (if the keyword applies to the job) and by pressing an end-of-statement key. The end-of-statement key can be either PROG START or ENTER – The Keyword-Response Summary charts in Appendix A explain the effect of end-of-statement keys on the prompting sequence.

*Program Start (PROG START) or Enter Plus (ENTER+)*

Pressing the PROG START or ENTER+ key tells the system that the response is complete and to prompt the next keyword.

*Enter Minus (ENTER-)*

Pressing the ENTER– key to end a response causes different processing depending on what keyword was prompted and what type of OCL cycle is being run.

*Pressing ENTER– after LOAD NAME or UNIT in a LOAD Cycle:* If the ENTER– key is used as an end-of-response to the LOAD NAME or UNIT prompts in a LOAD cycle, the remaining keywords in the cycle will be bypassed and MODIFY prompted.

*Pressing ENTER– after LOAD NAME or UNIT in a BUILD Cycle:* If the ENTER– key is used as an end-of-response to the LOAD NAME or UNIT prompts in a BUILD cycle, the system will prompt COMPILE OBJECT, SOURCE, or UNIT.

*Pressing ENTER– after FILE NAME:* If the ENTER– key is used as an edd-of-response to the FILE NAME prompt, the system prompts KEY LENGTH and HIKEY for multi-volume indexed files (see *Multi-Volume Files* in Appendix A).

*Pressing ENTER– after CALL NAME or UNIT in a CALL Cycle:* If the ENTER– key is used as an end-of-response to the CALL NAME or UNIT prompts in a CALL cycle, the OCL and any included control statements in the called procedure are not displayed. However, OCL statements with delayed responses are displayed and the system waits for a response. MODIFY is not prompted after either the OCL statements or the included control statements.

*Pressing ENTER– in the File Keywords:* If the operator responds to FILE NAME, he must also respond to the next two file keywords: UNIT and PACK. He can, however, bypass any or all of the rest of the file keywords. To bypass a single keyword he presses the PROG START key as a response. To bypass all of the remaining file keywords he presses the ENTER– key either as an end-of-response or as a response. Pressing the ENTER– key causes the system to prompt FILE NAME for the next file.

### Statement Numbers in an OCL Cycle

Statement numbers are assigned by the system to statements in an OCL cycle. These statement numbers are used by the operator when using MODIFY to reference previous OCL statements.

Each OCL statement, except READY and MODIFY, is assigned a three digit number. The first number in a BUILD or CALL cycle is 000, and in a LOAD cycle 010.

The statement number is incremented by 10 for each major keyword (DATE, SWITCH, COMPILE OBJECT, FILE NAME, etc.), and by one for each minor keyword (UNIT, PACK, LABEL, RECORDS, etc.).

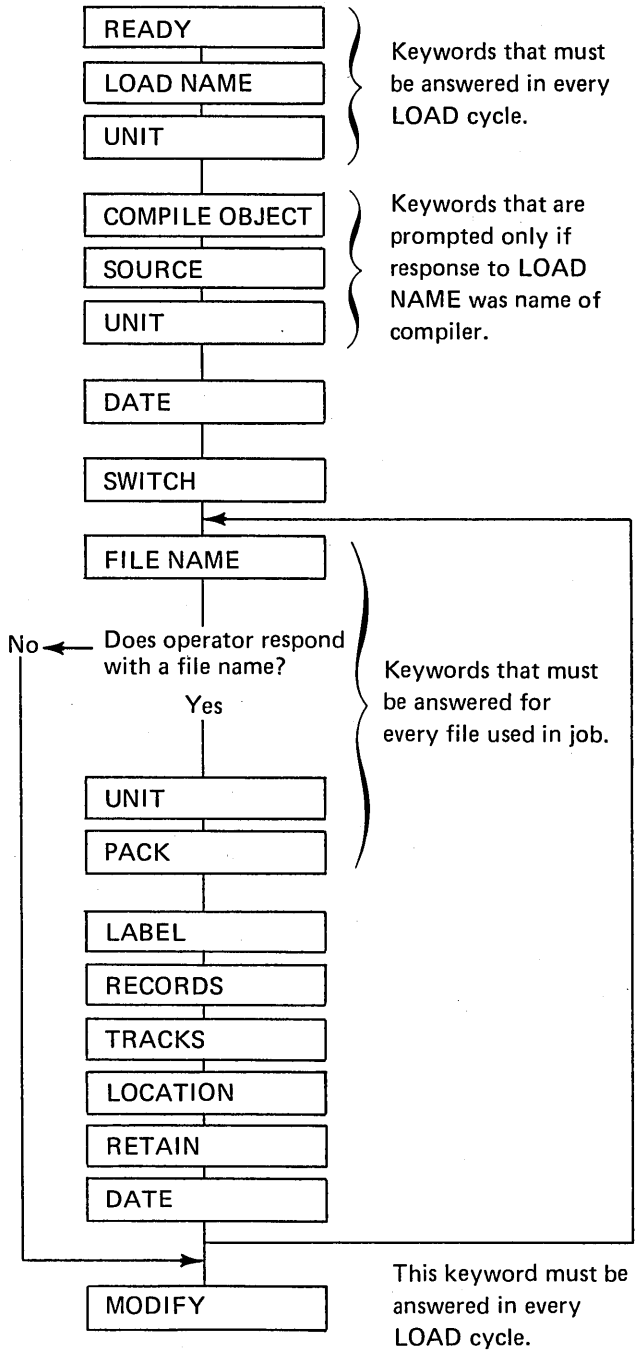
When the INCLUDE keyword is used to add utility control statements or sort source statements to a procedure, these included statements are assigned two-digit statement numbers. These statement numbers start with 00 and are incremented by one for each included statement.

The sample OCL jobs show the statement numbers assigned under various OCL cycles.

## Comments

Comments can be entered after any response on the same line if at least one space is left between the response and the comment (see *Modify: Entering Comments* under *MODIFY* in Part I to add comments during MODIFY time).

## Keyword Sequence for OCL Load Cycle


READY
LOAD NAME
UNIT
Keywords that must be answered in every LOAD cycle.
COMPILE OBJECT
SOURCE
UNIT
Keywords that are prompted only if response to LOAD NAME was name of compiler.
DATE
SWITCH
FILE NAME
No
Does operator respond with a file name?
Yes
Keywords that must be answered for every file used in job.
UNIT
PACK
LABEL
RECORDS
TRACKS
LOCATION
RETAIN
DATE
MODIFY
This keyword must be answered in every LOAD cycle.

**Keyword-Response Summary (Load Cycle)**

| Keyword | Response | Consideration |
|---|---|---|
| READY | LOAD | None |
| | Press PROG START | System prompts LOAD NAME |
| LOAD NAME | Program Name (Not Compiler) | Name of program to be run |
| | Press PROG START | System prompts DATE after UNIT |
| | OR | |
| | Press ENTER– | System prompts MODIFY after UNIT |
| OR | Compiler Program Name | Name of compiler to be run ($RPG for RPG II Compiler) |
| | Press PROG START | System prompts COMPILE OBJECT after UNIT |
| UNIT | R1, R2, F1, or F2 | Location of the disk whose object library contains the program to be run. |
| | Press PROG START | System prompts next keyword (see LOAD NAME in this chart) |
| | OR | |
| | Press ENTER– | System prompts MODIFY if not compiler |
| COMPILE OBJECT | R1, R2, F1, or F2 | Your system has more than one object library and you don't want to put the compiled program in the object library which contains the compiler. |
| | Press PROG START | System prompts SC'JRCE |
| OR | No Response | |
| | Press PROG START | System will put the compiled program in the object library which contains the compiler. Prompt SOURCE |
| SOURCE | Name of Source Program | Name assigned to RPG II source program when the KSE or Library Maintenance Utility Program put it in a source library |
| | Press PROG START | System prompts UNIT |

For information about the KSE Program see the *IBM System/3 Model 6 Conversational Utility Programs Reference Manual,* SC21-7528.
For information about the Library Maintenance Program see Part II of this manual.

**Keyword-Response Summary (Load Cycle) (Continued)**

| Keyword | Response | Consideration |
|---|---|---|
| UNIT | R1, R2, F1, or F2 | Location of the disk whose source library contains the RPG II source program |
| | Press PROG START | System prompts DATE |
| DATE | mmddyy or ddmmyy | Required when job date is not the same as the system date. (Responses must follow format established during system.) |
| | Press PROG START | System prompts SWITCH |
| | **OR** | |
| | No Response | |
| | Press PROG START | Either no date is required for the job **OR** you're going to use the system date. System prompts SWITCH. |
| SWITCH (XXXXXXXX) | 8-position setting (combination of 1's, 0's, and X's) | Required to change external indicators in RPG programs. Three choices for each position: 1 = turn indicator on; 0 = turn indicator off; X = leave indicator as is |
| | Press PROG START | System prompts FILE NAME |
| | **OR** | |
| | No Response | |
| | Press PROG START | Job does not use external indicators or you want to use the current setting. System prompts FILE NAME |
| FILE NAME | File name of file used by program | Columns 7-14 of RPG File Description Specifications, or predefined file name for system programs |
| | Press PROG START | System prompts UNIT |
| | **OR** | |
| | Press ENTER– | System prompts KEY LENGTH (see *Multi-Volume Files* in Appendix A) |
| | **OR** | |
| | No Response | |
| | Press PROG START | Either your job uses no files at all **OR** you have already described all the files the job uses. You want the system to prompt MODIFY |

| Keyword | Response | Consideration |
|---|---|---|
| UNIT | R1, R2, F1, or F2 | During a file creation run – location of disk where you want to write the file. During other runs – location of disk which contains the file to be processed. |
| | Press PROG START | System prompts PACK. |
| PACK | Disk Name | During a file creation run – the name which identifies the disk on which you want to write the file. During other runs – name which identifies the disk on which the file is located. |
| | Press PROG START | System prompts LABEL. |
| | OR | |
| | Press ENTER– | System prompts FILE NAME for next file. |
| LABEL | VTOC Filename | Required when VTOC Filename is different than response to FILE NAME. |
| | Press PROG START | System prompts RECORDS |
| | OR | |
| | Press ENTER– | System prompts FILE NAME for next file. |
| | OR | |
| | No Response | |
| | Press PROG START | You don't want to respond to this keyword; you want the system to prompt RECORDS |
| RECORDS [1] | 1-999999 | Number of records in the file. |
| | Press PROG START | System prompts LOCATION. |
| | OR | |
| | Press ENTER– | System prompts FILE NAME for next file. |
| | OR | |
| | No Response | |
| | Press PROG START | You don't want to respond to this keyword; you want system to prompt TRACKS. |

[1] At file creation time, *either* the number of records *or* the number of tracks must be specified.

Keyword-Response Summary (Load Cycle) (Continued)

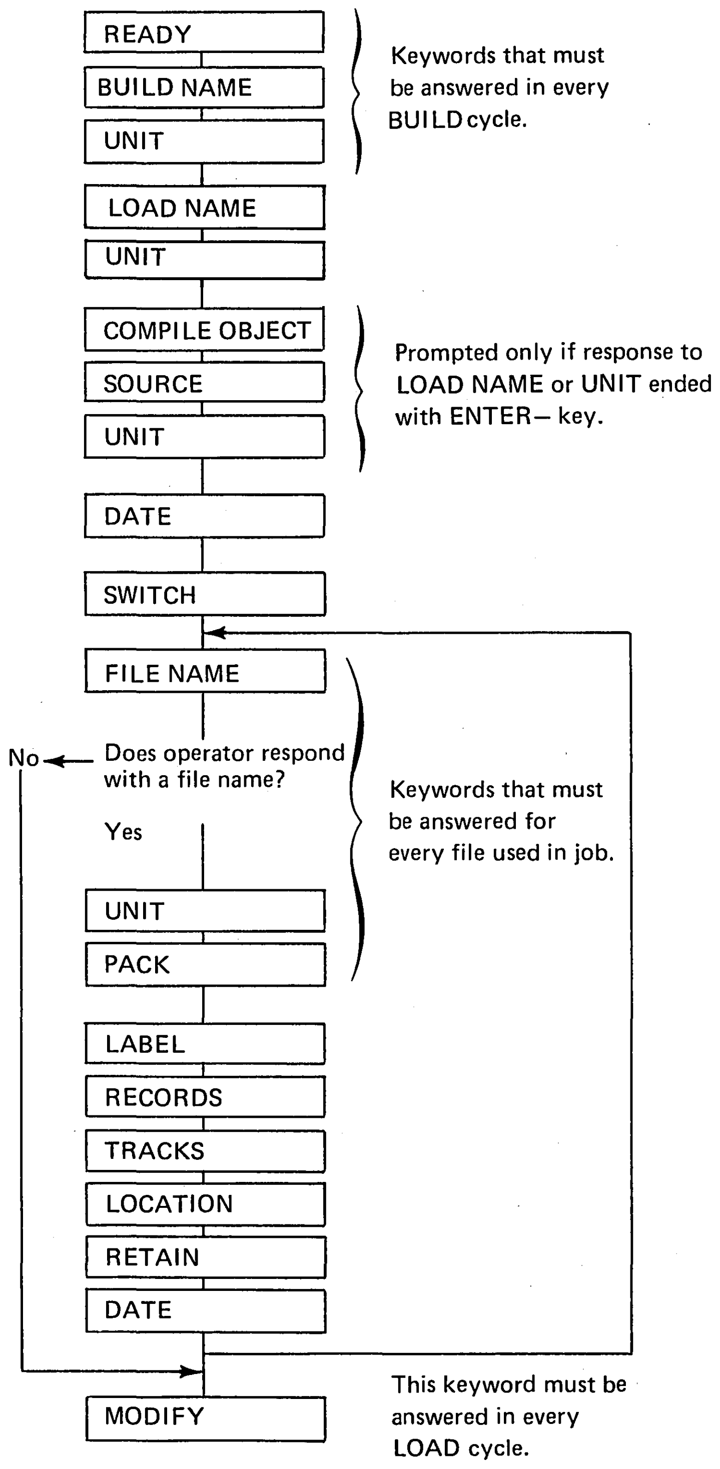
| Keyword | Response | Considerations |
|---|---|---|
| TRACKS [1] | 1-398 | Number of tracks the file will occupy. |
| | Press PROG START | System prompts LOCATION. |
| | OR | |
| | Press ENTER– | System prompts FILE NAME for next file. |
| | OR | |
| | No Response | |
| | Press PROG START | You don't want to respond to this keyword; you want system to prompt LOCATION. |
| LOCATION | 8-405 | Use during file creation runs if you want to specify a beginning track location for the file. |
| | Press PROG START | System prompts RETAIN. |
| | OR | |
| | Press ENTER– | System prompts FILE NAME for next file. |
| | OR | |
| | No Response | |
| | Press PROG START | You don't want to respond to this keyword; you want system to prompt RETAIN. |
| RETAIN | P, T, S, or A | P – permanent<br>T – temporary<br>S – scratch<br>A – activate scratch |
| | Press PROG START | System prompts DATE. |
| | OR | |
| | Press ENTER– | System prompts FILE NAME for next file. |
| | OR | |
| | No Response | |
| | Press PROG START | If file is being created, file designation will be T. System prompts DATE. |

[1] At file creation time, *either* the number of records *or* the number of tracks must be specified. If operator entered number of RECORDS, TRACKS will not be prompted.

Keyword-Response Summary (Load Cycle) (Continued)

| Keyword | Response | Considerations |
|---|---|---|
| DATE | mmddyy or ddmmyy | Required when job uses a file whose name and label are the same as those of another file on the same disk. (Response must follow format established during sysgen.) |
| | Press PROG START | System prompts FILE NAME for next file. |
| | OR | |
| | No Response | |
| | Press PROG START | You don't have to respond to this keyword; you want system to prompt FILE NAME, for next file. |
| MODIFY (Operator can use one, all, or a combination of the responses.) | LOG | Used only if CRT display or 22" printer on system (see Appendixes D and E). |
| | Press PROG START | System prompts LOG DEVICE. |
| | CANCEL | Cancel job. |
| | Press PROG START | System prompts READY or displays end-of-job halt. |
| | FORMS | Change lines per page printed output for system programs. |
| | Press PROG START | System prompts FORMS DEVICE. |
| | Asterisk (*) Followed by comments | Enter comment. |
| | Press PROG START | System waits for next MODIFY response. |
| | Statement number and comma | To delete statement |
| | Press PROG START | System waits for next MODIFY response. |
| | Statement number | To correct statement (LOAD NAME cannot be changed). |
| | Press PROG START | System waits for correct statement. |
| | RUN | Tells system — a. The LOAD cycle is complete. b. Run the job. |
| | Press PROG START | System runs job |

## Keyword Sequence for OCL Build Cycle


READY
BUILD NAME
UNIT
Keywords that must be answered in every BUILD cycle.
LOAD NAME
UNIT
COMPILE OBJECT
SOURCE
UNIT
Prompted only if response to LOAD NAME or UNIT ended with ENTER– key.
DATE
SWITCH
FILE NAME
No
Does operator respond with a file name?
Yes
UNIT
PACK
Keywords that must be answered for every file used in job.
LABEL
RECORDS
TRACKS
LOCATION
RETAIN
DATE
MODIFY
This keyword must be answered in every LOAD cycle.

**Keyword-Response Summary (Build Cycle)**

| Keyword | Response | Considerations |
|---|---|---|
| READY | BUILD | None |
| | Press PROG START | System prompts BUILD NAME |
| BUILD NAME | Procedure Name | Maximum of six alphameric characters. Must begin with alphabetic characters. Must not be DIR, SYSTEM, or ALL |
| | Press PROG START | System prompts UNIT. |
| UNIT | R1, R2, F1, or F2 | Location of the disk where you want to put procedure. (Procedure is placed in the source library of the disk operator specifies.) |
| | Press PROG START | System prompts LOAD NAME |
| LOAD NAME | Program Name | Name of program to be run. |
| | Press PROG START | System prompts DATE after UNIT. |
| | OR | |
| | Compiler Name | Name of compiler to be run ($RPG for RPG II compiler). |
| | Press ENTER– | System prompts UNIT then COMPILE OBJECT, SOURCE, UNIT |
| UNIT | R1, R2, F1, or F2 | Location of disk whose object library contains program |
| | Press PROG START | System prompts DATE |
| | OR | |
| | Press ENTER– | System prompts COMPILE OBJECT |
| | OR | |
| | ? (Delayed Response) | Forces operator to respond to unit during CALL cycle. |
| | Press PROG START | System prompts DATE. |
| | OR | |
| | Press ENTER– | System prompts COMPILE OBJECT |

**Keyword-Response Summary (Build Cycle) (Continued)**

| Keyword | Response | Considerations |
|---|---|---|
| COMPILE OBJECT | R1, R2, F1, or F2 | Your system has more than one object library and you don't want to put the compiled program in the object library which contains the compiler. |
| | OR | |
| | Press PROG START | System prompts SOURCE. |
| | No Response | |
| | OR | |
| | Press PROG START | System will put the compiled program in the object library which contains the compiler. System prompts SOURCE. |
| | ? (Delayed Response) | You will tell the system where to put the compiled program during the CALL cycle. |
| | Press PROG START | System prompts SOURCE. |
| SOURCE | Name of Source Program | Name assigned to source program when the KSE or Library Maintenance Utility Program put it in a source library. |
| | OR | |
| | Press PROG START | System prompts UNIT. |
| | ? (Delayed Response) | You will supply the name of the source program during the CALL cycle. |
| | Press PROG START | System prompts UNIT. |
| UNIT | R1, R2, F1, or F2 | Location of the disk whose source library contains the RPG source program |
| | OR | |
| | Press PROG START | System prompts DATE. |
| | ? (Delayed Response) | You will supply the location of the source program during the CALL cycle. |
| | Press PROG START | System prompts DATE. |

**Keyword-Response Summary (Build Cycle) (Continued)**

| Keyword | | Response | Considerations |
|---|---|---|---|
| DATE | | mmddyy or ddmmyy | To put a job date in the procedure. (Response must follow format established during system.) |
| | OR | Press PROG START | System prompts SWITCH. |
| | | ? (Delayed Response) | Forces operator to supply DATE during CALL cycle. |
| | OR | Press PROG START | System prompts SWITCH. |
| | | No Response | |
| | | Press PROG START | If no date is necessary for job or system date is acceptable. DATE will not be part of procedure. |
| SWITCH | | 8-position setting (combination of 1's, 0's, and X's) | Required to change external indicators in programs. Three choices for each position:<br>1 = turn indicator on<br>0 = turn indicator off<br>X = leave indicator as is |
| | OR | Press PROG START | System prompts FILE NAME. |
| | | ? (Delayed Response) | Forces operator to respond to SWITCH during CALL cycle |
| | OR | Press PROG START | System prompts FILE NAME |
| | | No Response | |
| | | Press PROG START | Job does not require external indicators. SWITCH will not be part of procedure. |
| FILE NAME | | File name of file used by program | Columns 7-14 of RPG File Description Specifications, or predefined filename for system programs. |
| | | Press PROG START | System prompts UNIT. |
| | OR | OR | |
| | | Press ENTER– | System prompts KEY LENGTH (see *Multi-Volume Files* in Appendix A). |
| | | ? (Delayed Response) | Forces operator to respond to FILE NAME during CALL cycle. |
| | OR | Press PROG START | System prompts UNIT |
| | | No Response | |
| | | Press PROG START | Either your job uses no files at all<br>**OR**<br>you have already described all the files the job uses. You want the system to prompt MODIFY |

**Keyword-Response Summary (Build Cycle) (Continued)**

| Keyword | Response | Considerations |
|---|---|---|
| UNIT | R1, R2, F1, or F2 | During a file creation run – location of disk where you want to write the file. During other runs – location of disk which contains the file to be processed. |
| | OR | |
| | Press PROG START | System prompts PACK. |
| | ? (Delayed Response) | Forces operator to respond to UNIT during CALL cycle. |
| | Press PROG START | System prompts PACK. |
| PACK | Disk Name | During a file creation run – the name which identifies the disk on which you want to write the file. During other runs – name which identifies the disk on which the file is located. |
| | OR | |
| | Press PROG START | System prompts LABEL. |
| | OR | |
| | Press ENTER– | System prompts FILE NAME for next file. |
| | ? (Delayed Response) | Forces operator to respond to PACK during CALL cycle. |
| | Press PROG START | System prompts LABEL. |
| | OR | |
| | Press ENTER– | System prompts FILE NAME. |
| LABEL | VTOC Filename | Required when VTOC Filename is different than response to FILE NAME. |
| | Press PROG START | System prompts RECORDS. |
| | OR | |
| | Press ENTER– | System prompts FILE NAME for next file. |
| | OR | |
| | ? (Delayed Response) | Forces operator to respond to LABEL during CALL cycle. |
| | Press PROG START | System prompts RECORDS. |
| | OR | |
| | Press ENTER– | System prompts FILE NAME. |
| | OR | |
| | No Response | |
| | Press PROG START | You don't want to respond to this keyword, you want the system to prompt RECORDS. |

Keyword-Response Summary (Build Cycle) (Continued)

| Keyword | Response | Considerations |
|---|---|---|
| RECORDS [1] | 1-999999 | Number of records in the file. |
| | Press PROG START | System prompts LOCATION. |
| | OR | |
| | Press ENTER– | System prompts FILE NAME for next file. |
| OR | ? (Delayed Response) | Forces operator to respond to RECORDS during CALL cycle. |
| | Press PROG START | System prompts LOCATION. |
| | OR | |
| | Press ENTER– | System prompts FILE NAME. |
| OR | No Response | |
| | Press PROG START | You don't want to respond to this keyword; you want system to prompt TRACKS. |
| TRACKS [1] | 1-398 | Number of tracks the file will occupy. |
| | Press PROG START | System prompts LOCATION. |
| | OR | |
| | Press ENTER– | System prompts FILE NAME for next file. |
| OR | ? (Delayed Response) | Forces operator to respond to TRACKS during CALL cycle. |
| | Press PROG START | System prompts LOCATION. |
| | OR | |
| | Press ENTER– | System prompts FILE NAME. |
| OR | No Response | |
| | Press PROG START | You don't want to respond to this keyword; you want to prompt LOCATION. |

[1] When a file is created, either the number of records or the number of tracks must be specified. If operator entered number of RECORDS, TRACKS will not be prompted.

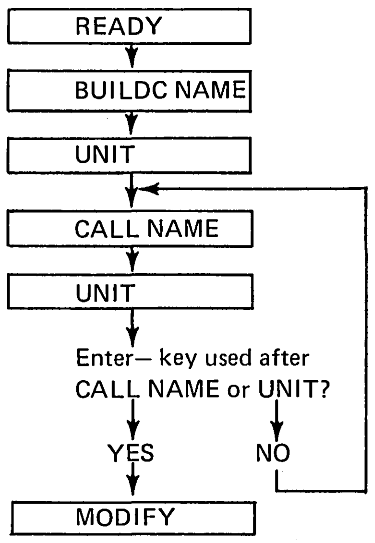
**Keyword-Response Summary (Build Cycle) (Continued)**

| Keyword | Response | Considerations |
|---|---|---|
| LOCATION | 8-405 | Use during file creation runs if you want to specify a beginning track location for the file. |
| | OR Press PROG START | System prompts RETAIN. |
| | OR | |
| | Press ENTER– | System prompts FILE NAME for next file. |
| | ? (Delayed Response) | Forces operator to respond to LOCATION during CALL cycle. |
| | OR Press PROG START | System prompts RETAIN. |
| | OR | |
| | Press ENTER– | System prompts FILE NAME. |
| | No Response | |
| | Press PROG START | You don't want to respond to this keyword; you want system to prompt RETAIN. |
| RETAIN | P, T, S, or A | P – permanent<br>T – temporary<br>S – scratch<br>A – activate scratch |
| | OR Press PROG START | System prompts DATE. |
| | OR | |
| | Press ENTER– | System prompts FILE NAME for next file. |
| | ? (Delayed Response) | Forces operator to respond to RETAIN during CALL cycle. |
| | OR Press PROG START | System prompts DATE. |
| | OR | |
| | Press ENTER– | System prompts FILE NAME. |
| | No Response | |
| | Press PROG START | If file is being created, file designation will be T. System prompts DATE. |

**Keyword-Response Summary (Build Cycle) (Continued)**

| Keyword | Response | Considerations |
|---|---|---|
| DATE | mmddyy or ddmmyy | Required when job uses a file whose name and label are the same as those of another file on the same disk. (Response must follow format established during system.) |
| | OR | |
| | Press PROG START | System prompts FILE NAME for next file. |
| | ? (Delayed Response) | Forces operator to respond to DATE during CALL cycle. |
| | OR Press PROG START | System prompts FILE NAME. |
| | No Response | |
| | Press PROG START | You don't have to respond to this keyword; you want system to prompt FILE NAME for next file. |
| MODIFY (Operator can use one, all, or a combination of the responses.) | LOG | Used only if CRT display or 22" printer on system (see Appendixes D and E). |
| | Press PROG START | System prompts LOG DEVICE. |
| | CANCEL | Cancel job. |
| | Press PROG START | System prompts READY or displays end-of-job halt. |
| | FORMS | Change lines per page printed output for system programs. |
| | Press PROG START | System prompts FORMS DEVICE. |
| | Asterisk (*) Followed by Comments | Enter comment. |
| | Press PROG START | System waits for next MODIFY response. |
| | Statement number and comma | To delete statement. |
| | Press PROG START | System waits for next MODIFY response. |
| | Statement number | To correct statement. |
| | Press PROG START | System waits for correct statement. |
| | INCLUDE | Add system program control statements to a procedure. |
| | Press PROG START | System prints 'ENTER INCLUDED STATEMENTS' and a 2-digit statement number. |
| | RUN | Tells system<br>a. The BUILD cycle is complete.<br>b. Run the job. |
| | Press PROG START | System runs the job. |

**Keyword Sequence for OCL BUILDC Cycle**



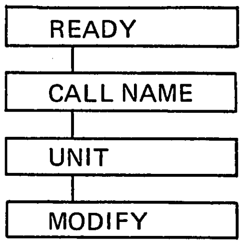
**Keyword-Response Summary (BUILDC Cycle)**

| Keyword | Response | Considerations |
|---|---|---|
| READY | BUILDC | None |
| | Press PROG START | System prompts BUILDC NAME. |
| BUILDC NAME | Master Procedure Name | Maximum of six alphanumeric characters. Must begin with alphabetic characters. (A-Z or #, @, $) Must not be DIR, SYSTEM, or ALL. Commas, blanks, quotes (apostrophes), and periods are not allowed. |
| | Press PROG START | System prompts UNIT. |
| UNIT | R1, R2, F1, or F2 | Location of the disk where you want to put procedure. (Procedure is placed in the source library of the disk which the operator specifies.) |
| | Press PROG START | System prompts CALL NAME. |

Keyword-Response Summary (BUILDC Cycle) (Continued)

| Keyword | Response | Considerations |
|---|---|---|
| CALL NAME | Name of Procedure | Name of a procedure in the source library. The procedure can be an IBM-supplied procedure (RPGB) or a procedure created by a BUILD or BUILDC cycle. |
| | Press PROG START | System prompts UNIT. |
| | **OR** | |
| | Press ENTER– | System prompts UNIT then MODIFY. |
| UNIT | R1, R2, F1, or F2 | Location of the disk whose source library contains the procedure. |
| | Press PROG START | System prompts CALL NAME (or MODIFY if ENTER– used after CALL NAME). |
| | **OR** | |
| | Press ENTER– | System prompts MODIFY. |
| MODIFY (Operator can use one, all, or a combination of the responses.) | LOG | Used only if CRT display or 22" printer is on system (see Appendixes D and E). |
| | Press PROG START | System prompts LOG DEVICE. |
| | CANCEL | Cancel job. |
| | Press PROG START | System prompts READY or displays end-of-job halt. |
| | FORMS | Change lines per page printed output for system programs. |
| | Press PROG START | System prompts FORMS DEVICE. |
| | Asterisk (*) Followed by Comments | Enter comment. |
| | Press PROG START | System waits for next MODIFY response. |
| | Statement number and comma | To delete statement. |
| | Press PROG START | System waits for next MODIFY response. |
| | Statement number | To correct statement. |
| | Press PROG START | System waits for correct statement. |
| | RUN | Tells system – a. The cycle is complete. b. Run the job. |
| | Press PROG START | System runs job. |

**Keyword Sequence for OCL Call Cycle**



Keywords that must be answered in every CALL cycle.

**Keyword-Response Summary (Call Cycle)**

| Keyword | Response | Considerations |
|---|---|---|
| READY | CALL | None |
| | Press PROG START | System prompts CALL NAME. |
| CALL NAME | Procedure Name | Procedure name from the source library directory **OR** RPG (the IBM-supplied RPG II compile procedure) |
| | Press PROG START | System prompts UNIT. |
| | **OR** Press ENTER– | System prompts UNIT, then runs the job. Does not display the procedure except for statements with delayed responses. Does not prompt MODIFY. |
| UNIT | R1, R2, F1, or F2 | Location of the disk whose source library contains the procedure. |
| | Press PROG START | Print procedure. |
| | **OR** Press ENTER– | System runs the job. Does not prompt procedure except for statements with delayed responses. Does not prompt MODIFY. |

**Keyword-Response Summary (Call Cycle) (Continued)**

| Keyword | Response | Considerations |
|---|---|---|
| PROCEDURE DISPLAYED ON SYSTEM PRINTER ❶ (unless ENTER– key was pressed after CALL NAME or UNIT prompts) | | |
| MODIFY (Operator can use one, all, or a combination of the responses.) | LOG | Used only if CRT or 22" printer on system (see Appendixes D and E). |
| | Press PROG START | System prompts LOG DEVICE. |
| | CANCEL | Cancel job. |
| | Press PROG START | System prompts READY or displays end-of-job halt. |
| | FORMS | Change lines per page of printed output for system programs. |
| | Press PROG START | System prompts FORMS DEVICE. |
| | Asterisk (*) Followed by Comment | Enter comment. |
| | Press PROG START | System waits for next MODIFY response. |
| | Statement number and comma | To delete statement in displayed procedure |
| | Press PROG START | System waits for next MODIFY response |
| | Statement number and corrected statement | To correct statement in displayed procedure (LOAD NAME cannot be changed). |
| | Press PROG START | System waits for correct statement. |
| | RUN | Tells system – a. The CALL cycle is complete. b. Run the job. |
| | Press PROG START | System runs job. |

❶ A. Procedures with INCLUDE Statements
When a procedure contains SORT source statements or utility control statements, the display part of the CALL cycle is more complex. See *Considerations During a CALL Cycle,* under *MODIFY; Including Control Statements* in Part I.

B. Procedures with Delayed Responses
The procedure is displayed statements by statement (unless the ENTER– key was pressed after responding to the CALL NAME or UNIT keywords). When the system reaches a statement which contains a delayed response, it will display the statement keyword and wait for the operator's response.

The IBM 5496 Data Recorder, Model 1, with System/3 Model 6 Attachment Feature or the IBM 129 Card Data Recorder with card input/output attachment feature provides card input/output capability for System/3 Model 6. The data recorder is selected as system input device during OCL prompting. Control is returned to the keyboard by entering a READER statement in the data recorder or by performing another program load procedure.

## ASSIGNING DATA RECORDER AS SYSTEM INPUT DEVICE

| | System Prompts | Operator Enters |
|---|---|---|
| At IPL time | DATE – | current date |
| | READER – | DATA96 |
| Between jobs | READY – | READER |
| | READER – | DATA96 |

Following the DATA96 response, all OCL must be entered in card form from the data recorder.

At the time the data recorder is selected as system input device the following switch settings must be:

| | | |
|---|---|---|
| Operator Console | – | DATA RCRDR switch to ON LINE |
| 5496 Data Recorder | – | 1. Power switch ON<br>2. AUTO REL switch ON<br>3. Print switch ON or OFF<br>4. All other switches OFF |
| 129 Data Recorder | – | 1. Power switch ON<br>2. PROGRAM MODE dial set to DATA READ<br>3. PUNCH-DIR PUNCH-VERIFY switch set to PUNCH<br>4. Print switch ON or OFF<br>5. REC ADV/CARD FEED switch set to AUTO |

**IBM 129 Programming Considerations**

The user should be aware of the following considerations when programming applications for the IBM 129:

1. System support for the 5496 also supports the 129.
2. Unique diagnostics for the 129 are not provided.
3. Object programs cannot be read or punched on the 129 (whereas the 5496 provides this function). Therefore, the system function LOAD* is not supported for the 129.
4. The OCL command READER-DATA96 is used for either the 5496 or the 129.

## RETURNING CONTROL TO KEYBOARD

The keyboard is reassigned as system input device by doing either of the following:

- Enter a /& statement followed by a // READER KEY statement from the Data Recorder. These statements must be placed after a // RUN statement and before a // LOAD or // CALL statement.
- Perform a program load from the operator console.

## CONTROL STATEMENTS IN PROCEDURES

OCL statements that control the entering of other OCL statements are invalid in procedures. These statements, HALT, NOHALT, LOG, READER, and PAUSE, are ignored when read from procedures during a CALL cycle and are not put in a procedure during a BUILD cycle.

## CARD FORMAT OF OCL STATEMENTS

The following OCL statements can be loaded from the data recorder. The parameters of the statements that are prompted in conversational mode are described elsewhere in this book. The statements that are allowed with card input are described in the notes following this list.

In statement formats, special characters such as //, and words written in capital letters are information that must be used exactly as shown. Words written in small letters, such as code, program-name, and unit, represent information that you must supply.

## OCL STATEMENTS

// LOAD Program-Name, Unit

// LOAD*

**Explanation:** An asterisk indicates that the object program will be loaded from the data recorder. Program-name unit parameters must not be included. The cards that contain the program must follow the RUN statement for the program and must be followed by /* to indicate the end of the object deck.

// CALL Procedure-Name, Unit

// RUN

// READER KEY

// SWITCH

// COMPILE OBJECT-unit, SOURCE-name, UNIT-unit

**Explanation:** OBJECT-unit must be the first parameter on the statement.

// FORMS DEVICE–PRIMARY LINES-number
// FORMS DEVICE–SECONDARY LINES-number

**Explanation:** The DEVICE parameter is optional. The LINES parameter must be present.

// LOG ON
// LOG OFF
// LOG CRT
// LOG PRIMARY
// LOG SECONDARY

**Explanation:** The log device must be on when the system is in conversational mode.

// FILE NAME-filename, UNIT-unit, PACK-name,
   // LABEL-filename, RECORDS-number, TRACKS-number,
   // LOCATION-track number, RETAIN-code, DATE-date

**Explanation:** LABEL, RECORDS or TRACKS, LOCATION, RETAIN, and DATA parameters are optional. NAME-filename must be the first parameter on the statement.

// NOHALT
// HALT

**Explanation:** During card input, the system halts each time a /* (end-of-job) or /& statement is read. The NOHALT statement allows the system to start the next job without a halt. The HALT statement is used to cancel a NO-HALT condition. If the HALT and NOHALT statements are placed in a procedure they are not displayed when the procedure is prompted.

// PAUSE

**Explanation:** A PAUSE statement entered from the data recorder causes the system to stop until the operator restarts it. PUASE statements are usually preceded by comments (*) instructing the operator to perform some function on the system. If PAUSE statements and comments are placed in a procedure the comments are displayed during prompting but the system does not stop.

*

/&

/*

**Explanation:** /* indicates end-of-job. /& is used as a delimiter and indicates that a new job is starting. If a 3 option (immediate cancel) was taken at a halt in the preceding job, the system looks for the next LOAD or CALL statement in the job stream. The /& statement changes this mode and tells the system to read the next // card regardless of what it is. In this manner a // READER KEY statement would be recognized, returning control to the keyboard.

## GENERAL CODING RULES

The rules for coding the OCL statements in cards are as follows:

1. // in positions 1 and 2.

2. One or more blanks between the // and the word that forms the statement identifier (LOAD, RUN, CALL, etc.).

3. One or more blanks between the statement identifier and the first parameter.

4. If you need more than one parameter, use a comma to separate them. No blanks are allowed in or between parameters. Anything following the first blank is considered a comment.

5. If you are writing keyword parameters (XXX-xxx), place the keyword first and use a hyphen to separate the keyword from the code or data.

6. If the parameter is not a keyword parameter, write the parameters in the order in which they are shown. Keyword parameters can be in any order except in the following cases:

| | |
|---|---|
| // COMPILE | OBJECT-unit must be the first parameter. |
| // FILE | NAME-filename must be the first parameter. |

7. All OCL statements except FILE must not exceed 96 characters. Because of the large number of parameters possible in a FILE statement, you can continue the FILE statement on additional cards. The rules are:

   - Place a comma after the last parameter in every card but the last. The comma followed by a blank indicates the statement is continued.
   - Begin each new card with // in positions 1 and 2.
   - Leave one or more blanks between the // and the first parameter.

8. Comments can be placed after the parameters on any OCL statement (except HIKEY parameters. See *Coding Multi-Volume File Parameters* in this appendix). Leave one or more blanks after the last parameter and then list the comment. Complete lines of comments are entered with the *comment statement. Place an * in column 1 and start the comments in column 2.

## STATEMENT ORDER

| | |
|---|---|
| /& | should be the first statement of a job. |
| // LOAD | statement must precede RUN statement in job stream. If LOAD*, the cards that contain the program must follow the RUN statement and be followed by a /* statement. |
| // CALL | statement must precede RUN statement in job stream. |
| // RUN | statement must be last statement within the set of statements required to run a program. |
| // READER | statement must precede a LOAD or CALL statement and follow a RUN statement. |
| // SWITCH | statement must follow a LOAD or CALL statement and must precede a RUN statement. |
| // COMPILE | statement must follow a LOAD or CALL statement and must precede a RUN statement. |
| // FORMS | can appear anywhere in the job stream. |
| // LOG | statement must follow a LOAD or CALL statement and precede a RUN statement. |
| // FILE | statements must follow a LOAD or CALL statement and precede a RUN statement. |
| // HALT | can appear anywhere in the job stream. |
| // NOHALT | can appear anywhere in the job stream. |
| // PAUSE | can appear anywhere in the job stream. |
| *comments | can appear anywhere in the job stream. |
| /* (end-of-job) | follows a program deck or data file entered from the Data Recorder. |

## CODING MULTI-VOLUME FILE PARAMETERS

When coding card OCL file statements for multi-volume files these rules must be followed:

1. Each parameter that requires multiple entries must begin and end with a single quote (') and have the entries separated by commas.

2. The HIKEY parameter must contain HIKEYs separated by commas. When continuation cards are needed for HIKEY parameters, comments are not allowed on the cards, and the data must start in column four of the continuation card.

3. An apostrophe within a HIKEY must be entered as a double apostrophe or it will be decoded as end of HIKEYs, and an error will occur.

4. When using only one volume of an indexed multi-volume file, the HIKEY parameter must be included with beginning and ending apostrophes. The other file parameters must not have apostrophes.

5. To indicate packed keys, HIKEY-P'xxxx, xxxx, xxxx,' must be coded. All characters in packed HIKEYs must be numeric and all packed HIKEYs must be the same length.

Key length is not a parameter for indexed files when OCL statements are entered on cards. Sample job 2 under *Multi-Volume Files* in Appendix A would have the following four OCL file statements if OCL were on cards:

```
// FILE NAME-INVMSTR,UNIT-'R1,R2',
PACK-'VOLI02,VOLI03,VOLI03,VOLI04,

// VOLI05',HIKEY-'175-200-233W1B2,
380-456-280W3R6,629-384-300W3F6,

// 949-475-849W8F8,999-999-999W9F9',
TRACKS-'100,193,150,193,80',

// LOCATION-'87,8,49,8,8',RETAIN-P
```

## Keyword Descriptions

### BUILD NAME

When the system prompts BUILD NAME, the operator responds with a name for the procedure that will be put in a source library at the end of the sequence. (The operator's response to UNIT determines what source library the procedure will be put in.) At the end of the BUILD cycle, the system enters the procedure in the source library and puts the procedure name in the source library directory as a permanent entry. Restrictions on naming a procedure are:

1. Name must not contain more than six alphanumeric characters. Blanks, commas, quotes (apostrophes), and periods are not allowed.

2. First character must be alphabetic (A-Z or #, @, $).

3. Name must not be DIR, SYSTEM, or ALL (these names are reserved for system use).

#### Duplicate Procedure Names

If the operator's response to BUILD NAME duplicates the name of a procedure already in the source library directory, the system prints a message and reprompts BUILD NAME.

The operator can:

1. Proceed – by typing a different name or the same name and a different unit.

2. Proceed – by typing the same name and unit again. The new procedure will then overlay the old procedure in the source library.

3. End the job – see description of error message options in *IBM System/3 Model 6 Operator's Guide,* GC21-7501.

#### Deleting a Source Library Procedure

The system gives a P (permanent) designation to all procedures entered into a source library during a BUILD cycle. Therefore, the only way to delete a procedure from a source library is to run the Library Maintenance Utility Program. (For information about the Library Maintenance Utility Program see Part II of this manual.)

### BUILDC NAME

The response to BUILDC NAME is the name of a master procedure you want to build. The rules and restrictions are the same as for the keyword BUILD.

### CALL NAME

The response to CALL NAME is the name of the procedure you want to run. This can be either:

- The name of a procedure entered in a source library after a BUILD or BUILDC cycle. (The operator's response to the keyword BUILD NAME, or BUILDC NAME determines the name of the procedure.)

- RPG (the IBM-supplied RPG II Compile Procedure).

If the operator does not know the procedure name, he can get a printout of the source library directory by running the Library Maintenance Utility Program. (See Part II of this manual for more information about this program.)

The operator can call a procedure without displaying all its OCL statements by pressing the ENTER– key after responding to CALL NAME or UNIT. The procedure is loaded and run. The only statements displayed are those with delayed responses. The system does not prompt MODIFY after either the OCL statements or the included control statements.

## COMPILE KEYWORDS

### COMPILE OBJECT Keyword

The keyword COMPILE OBJECT requires a response (R1, R2, F1, or F2) if the system has more than one object library and you do not want to put the compiled program in the same object library where the compiler resides.

If the operator does not respond to COMPILE OBJECT, but merely presses the PROG START key, the system places the compiled program in the object library where the compiler resides.

F1 refers to the fixed disk on drive one.
R1 refers to the removable disk on drive one.
F2 refers to the fixed disk on drive two.
R2 refers to the removable disk on drive two.

### SOURCE Keyword

*In a LOAD Cycle*

SOURCE is prompted only when the response to LOAD NAME is the name of a compiler (such as $RPG). The response to SOURCE is the name of the source program you want to compile. (This name must be the one you used when you put the program in a source library during a KSE or Library Maintenance Program run.)

For information about the KSE Program see the *IBM System/3 Model 6 Conversational Utility Programs Reference Manual,* SC21-7528. For information about the Library Maintenance Program see Part II of this manual.)

*In a BUILD Cycle*

There are two possible responses to SOURCE during a BUILD cycle: the name of a source program you want to compile or a delayed response (a question mark). Each response has a special significance to the system.

| **Response** | **Tells System** |
|---|---|
| Name of Source Program You Want to Compile | You're building a procedure that will compile a particular source program. (The program must be in a source library.) The program name you supply must be the one you used when you put the program in a source library during a KSE or Library Maintenance Program run. |
| ? (Delayed Response) | You're building a general compile procedure. You will supply the necessary source program information (name and location of the source program and where you want to put the compiled program) during the CALL cycle. |

### UNIT Keyword

The response to UNIT gives the location of the disk whose source library contains the source program being compiled. Possible responses are F1, R1, F2, and R2.
F1 refers to the fixed disk on drive one.
R1 refers to the removable disk on drive one.
F2 refers to the fixed disk on drive two.
R2 refers to the removable disk on drive two.

## DATE

This DATE keyword lets the operator change the system date for a particular job. (The system date is used in headings on program listings, in headings on printed output, and in labels for new files.)

The system date is established at IPL time. This date is used for every job unless the operator overrides it.

### Overriding the System Date

The operator can override the system date for any single job by typing in a new date when the system prompts the keyword DATE. The new system date is used only for the one job. When that job is finished, the system date automatically reverts to its IPL setting.

### Format of the DATE Statement

Although the operator can override the system date, he cannot change the date format. The system date format is established during sysgen as either:

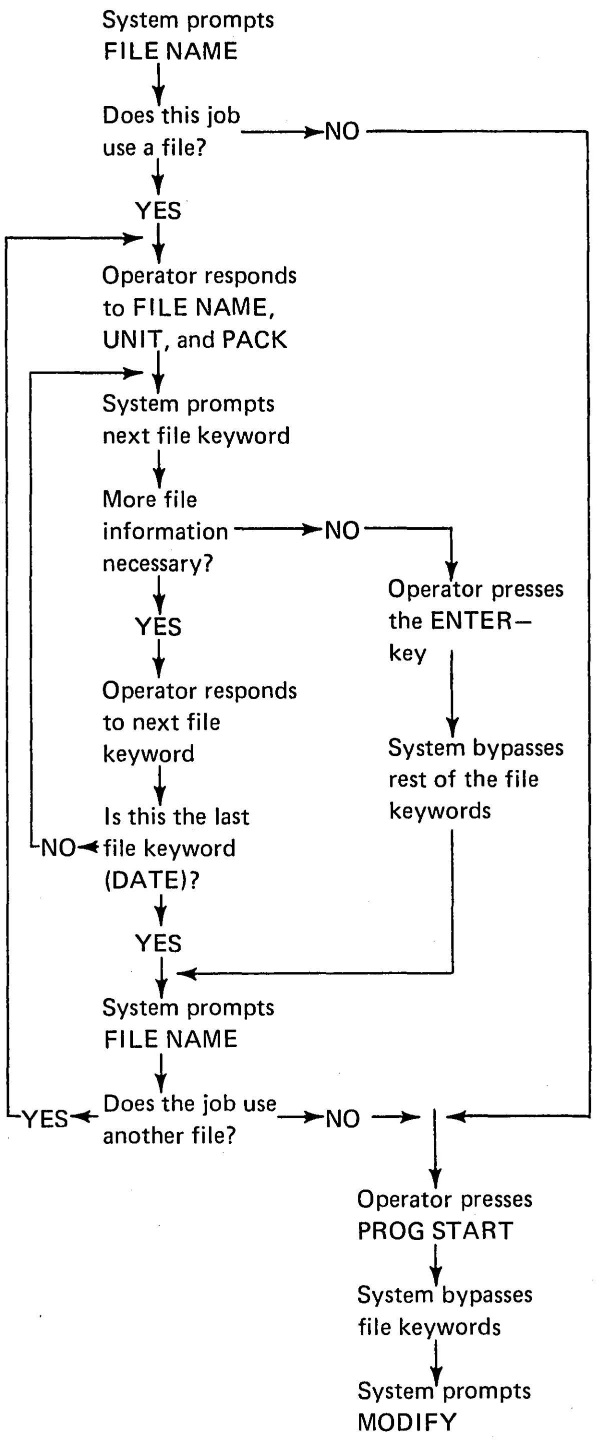
- mmddyy (month/day/year) -- For U.S. installations
- ddmmyy (day/month/year) – For World Trade installations.

The three elements (month/day/year) can be separated by any non-numeric symbol (except a comma, quotation mark, or blank) or run together without any separation.

In a system using the mmddyy format, for example, all of the following would be valid ways of typing May 12, 1971:

- 05/12/71
- 05-12-71
- 051271
- 5/12/71

### System-Operator Interaction During Prompting of File Keywords


System prompts FILE NAME
Does this job use a file?
NO
YES
Operator responds to FILE NAME, UNIT, and PACK
System prompts next file keyword
More file information necessary?
NO
Operator presses the ENTER–key
System bypasses rest of the file keywords
YES
Operator responds to next file keyword
Is this the last file keyword (DATE)?
NO
YES
System prompts FILE NAME
Does the job use another file?
YES
NO
Operator presses PROG START
System bypasses file keywords
System prompts MODIFY

For every file used in a job, you must respond to the following keywords:

| Keyword | Response |
|---|---|
| FILE NAME | FILENAME from the file specification at compile time **OR** Predetermined filename (for $RPG, $KDE, $DSORT, $COPY). |
| UNIT | R1 or F1 (Location of disk where you want to write a file during file creation run on system with one disk drive.) **OR** R1, F1, R2 or F2 (Location of disks which contain a file to be processed during other than a file creation run on systems with two disk drives. |
| PACK | Name assigned to a disk by the disks initialization program. This name can be one which identifies a disk on which you want to write a file during a file creation run or a name that identifies a disk on which a file is located. |

*File Name for Customer Programs*

For a file used in an RPG II compiled customer program, the operator's response to FILE NAME is the name in columns 7-14 of the RPG II File Description Specifications.

*File Name for $RPG, $DSORT, $COPY, $MICR, and $KDE*

For $RPG's predefined file names see *IBM System/3 Model 6 RPG II Reference Manual,* SC21-7517.

For $DSORT see *IBM System/3 Disk Sort Reference Manual,* SC21-7522.

For $COPY see Part II of this manual.

For $MICR see *IBM System/3 Model 6 Utility Program for the IBM 1255 Magnetic Character Reader Reference Manual,* SC21-7527.

For $KDE see *IBM System/3 Model 6 Conversational Utility Programs Reference Manual,* SC21-7528.

*Multiple Files*

A job often involves several files. When this is the case, the operator must respond to several series of file keywords. The first time the system prompts the file keywords, the operator responds with information about one file. After the operator responds to DATE, the system will again prompt FILE NAME. This time the operator responds with the name of the second file.

When he has responded to the file keywords for all the files that will be used in the job, the operator should respond to FILE NAME by pressing PROG START. The system then bypasses the rest of the file keywords and prompts MODIFY.

A maximum of 15 file statements can be used for each job.

### UNIT Keyword

Possible responses to the keyword UNIT are F1, R1, F2 and R2

F1 refers to the fixed disk on drive one.
R1 refers to the removable disk on drive one.
F2 refers to the fixed disk on drive two.
R2 refers to the removable disk on drive two.

### PACK Keyword

Whenever a job involves a disk file you must tell the system the name of the disk where the file is (or will be) located, so the system can make sure that disk is mounted before the job is begun. To tell the system the name of the disk the file is on, the operator responds to the keyword PACK with the name assigned to the disk during its initialization. (The Disk Initialization section of Part II of this manual explains the procedure for naming a new disk.)

Although most installations keep a record of the names and contents of each of their disk packs, the operator can always get the name of any disk by running the File and Volume Label Display Utility Program. The disk name is part of the output of this program.

### LABEL Keyword

When a file is created, the system enters a file name in the VTOC. The keyword LABEL refers to this VTOC file name. Unless the operator responds to LABEL, the name entered in the VTOC is the same as the operator's response to FILE NAME.

LABEL requires a response:

1. At file creation time, if you want the VTOC file name to be different from the operator's response to FILE NAME. (For example, if the RPG II file name is A but the disk already has an A file, a response to LABEL would be required, and the response would have to be something other than A.)

2. During a program run, if you are using a file whose VTOC file name is different from the operator's response to FILE NAME.

### RECORDS and TRACKS Keywords

When a file is created, the operator must tell the system how much disk space to allocate for the file. He does this by responding to either TRACKS or RECORDS. (If the operator responds to RECORDS, TRACKS will not be prompted.)

The following chart shows the possible responses to these keywords and how the system interprets the responses.

| Keyword | Operator Response | Tells System |
|---|---|---|
| TRACKS | 1-398 | Number of disk tracks needed for the file |
| RECORDS | 1-999999 | Number of records in the file |

#### *Responding to TRACKS*

The response to TRACKS is the number of disk tracks the records in a file will occupy. (Appendix B reviews how to convert the number of records in a sequential, direct, or indexed file into the number of tracks that would be required to contain the file records on a disk.)

#### *Responding to RECORDS*

If the operator does not want to convert record numbers into track requirements himself, the system will do it for him. The system determines the track requirements for a file when the operator responds to RECORDS.

### LOCATION Keyword

LOCATION requires a response during file creation if you want to control the placement of files on the disk. LOCATION is required when creating several versions of the same file of the same size (LOCATION is not required if the file size is different). LOCATION is also required when loading an offline multi-volume file to packs that contain other files. It can also be used to reference one of several files having the same name.

The response to LOCATION is the track where you want the file to begin. Possible responses are 8 through 405. (Tracks 0 through 7 are reserved for system use.)

If the operator does not respond to the keyword LOCATION when a new file is created, the system places the file in whatever available area it fits best.

### RETAIN Keyword

The keyword RETAIN applies to file designation. Files can be designated: P (permanent), T (temporary), or S (scratch).

The operator responds to RETAIN either:

1. At file creation, to give a designation to the file being created.

2. When accessing a file, to change the designation of a file from T to S or from S to T.

#### *File Creation*

A file designation (along with the file name, length, and other related information) is placed in the VTOC when a file is created. The operator controls file designation by his response to RETAIN. (If the operator does not respond to RETAIN, the system gives the file a T designation.)

*Permanent Files:* Because permanent files are protected against inadvertent overlaying or altering, give a P designation to all the files you want to keep. The only way to change a permanent file is to delete it by running the File Delete Utility program.

## FILE KEYWORDS (continued)

*Temporary Files:* Give a T designation to a file if you plan to use it several times within a couple of days and will not need it after that. The only way to overlay a temporary file is to load a new file over it. To do this, the operator's responses to all the file keywords must duplicate those of the present T file.

*Scratch Files:* Give an S designation to any file you plan to use only once. When a scratch (S) file is created, it is not entered in the Volume Table of Contents (VTOC). After the job that created the file is run, the file is lost. The way that an S retain type can appear in the VTOC is to change a T entry to an S by using RETAIN-S in the file statement, or change a T or P entry to S by using a $DELET SCRATCH statement. The system will overlay a scratch file if the disk pack is full and/or file space is needed by a new file or a system program.

*Changing File Designation of Existing File*

When the system prompts RETAIN, the operator can:

- Accept the current file designation. (By pressing PROG START).
- Change a temporary file to a scratch file (by typing an S). The VTOC will contain an S entry for the file.
- Change a scratch file listed in the VTOC to a temporary file by typing an A.

*Deleting Files*

The operator can delete any file by running the File Delete Utility Program, which changes the file designation in the VTOC to S. This effectively deletes the entire file, because the system will overlay the file area as soon as more file space is needed. When the file area is overlaid, the file name is erased from the VTOC.

### DATE Keyword

This keyword (prompted after the keyword RETAIN) refers to the system date in effect when a file was created. The system date is established at IPL. This date is used for every job unless the operator overrides it.

DATE requires a response only if the job being run uses a file whose name and label are duplicated by another file on the same disk. In this case, the operator responds to DATE by typing in the system date in effect when the file he wants to use was created. With this date, the system can distinguish one file from others on the same disk with the same VTOC file name and label.

If neither the date nor the location is given, the file having the latest date is the one automatically referenced.

If the operator does not know what the system date was when the file was created, he can get a printout of the creation dates for all files on a disk by running the File and Volume Label Display Utility Program. (Detailed information on this program is available in Part II of this manual.)

*Restriction During File Creation*

A response to DATE tells the system that this file already exists. If DATE is entered during a file creation run a FILE NOT FOUND error occurs.

## HALT

The operator can respond to the keyword READY with HALT. The system will then halt at the end of each job. HALT need only be entered to cancel the effect of a NOHALT statement.

## LOAD NAME

### For Customer Programs

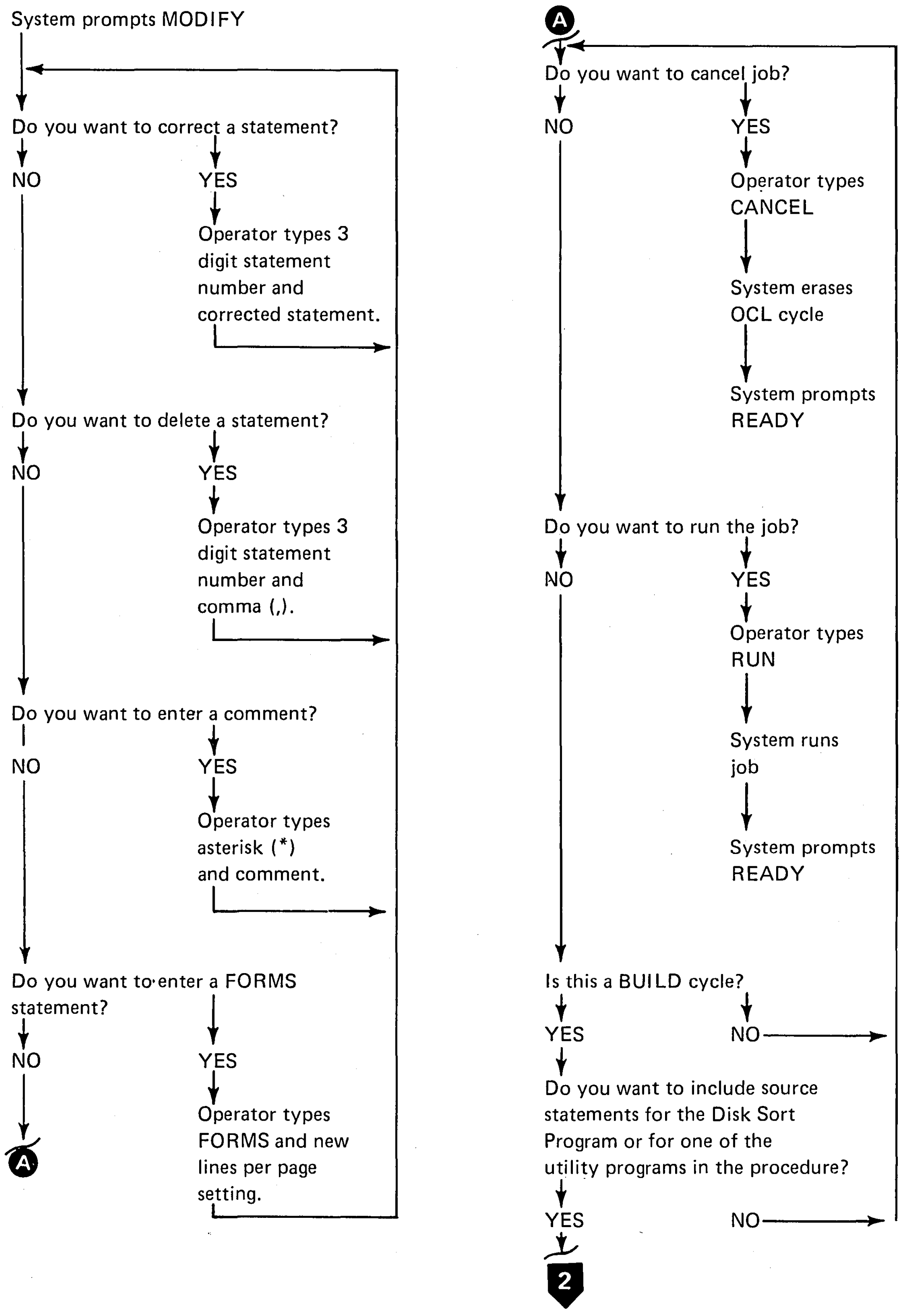
The response to LOAD NAME is the name of the customer's RPG II program.

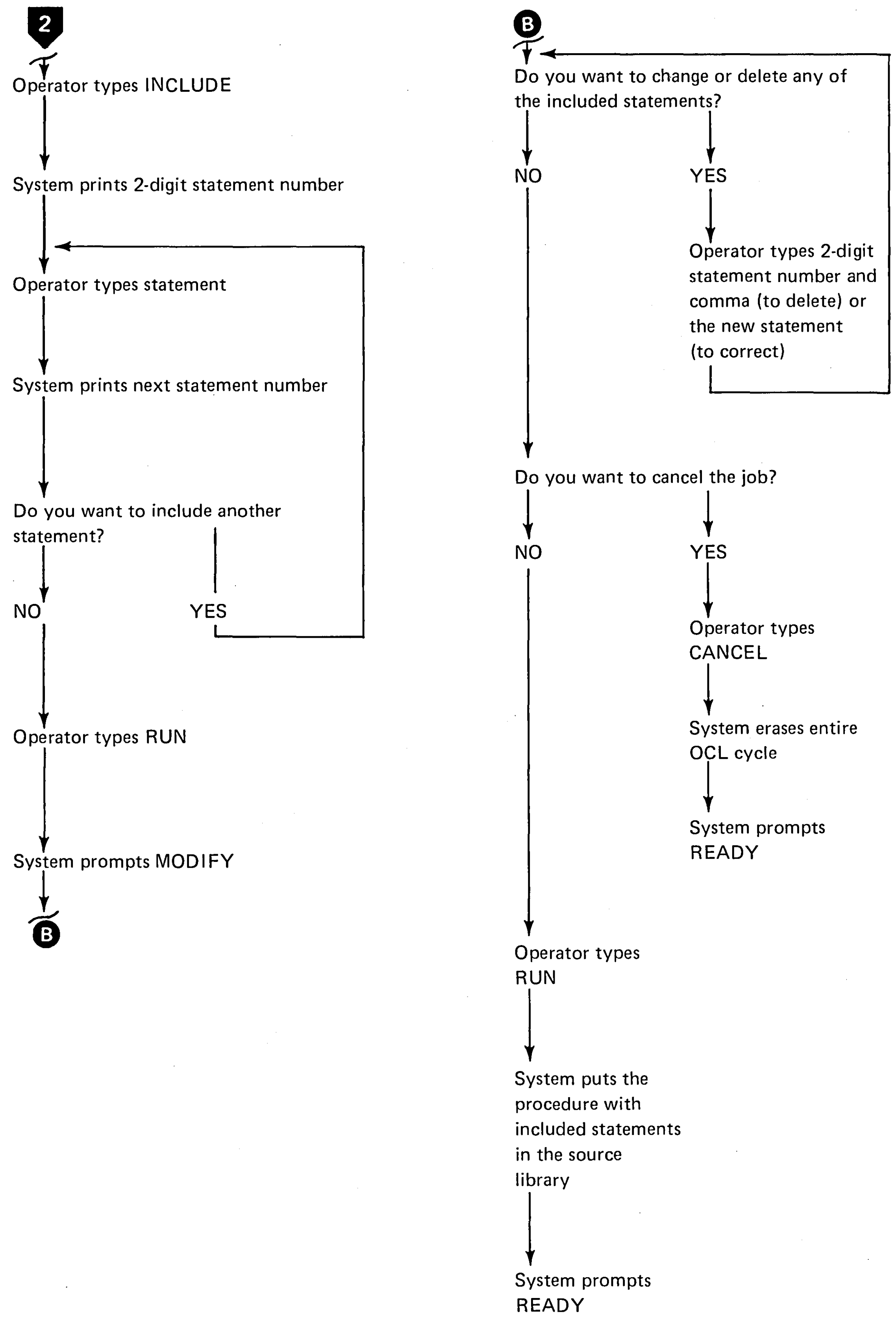
### For System Programs

The response to LOAD NAME is the name of the specific system program you want to run.

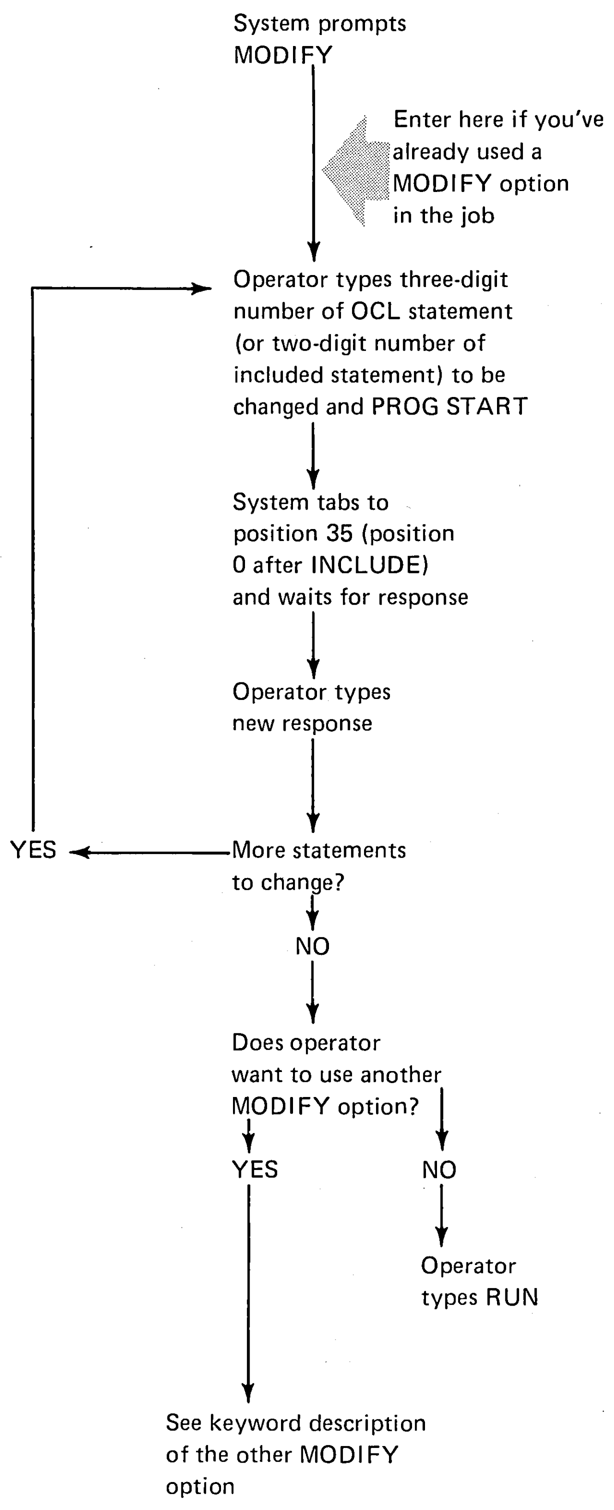
| Name | Program |
|---|---|
| $ALT | Alternate Track Assignment |
| $BUILD | Alternate Track Rebuild |
| $COPY | Disk Copy/Dump |
| $FORT | FORTRAN Compiler |
| $INIT | Disk Initialization |
| $LABEL | File and Volume Label Display |
| $DELET | File Delete |
| $MAINT | Library Maintenance |
| $KSE | Keyboard Source Entry |
| $KDE | Keyboard Data Entry |
| $DIU | Data Interchange |
| $MICR | 1255 Magnetic Character Reader Utility |
| $RPG | RPG II Compiler |
| $DSORT | Disk Sort |

## System-Operator Interaction During Modification

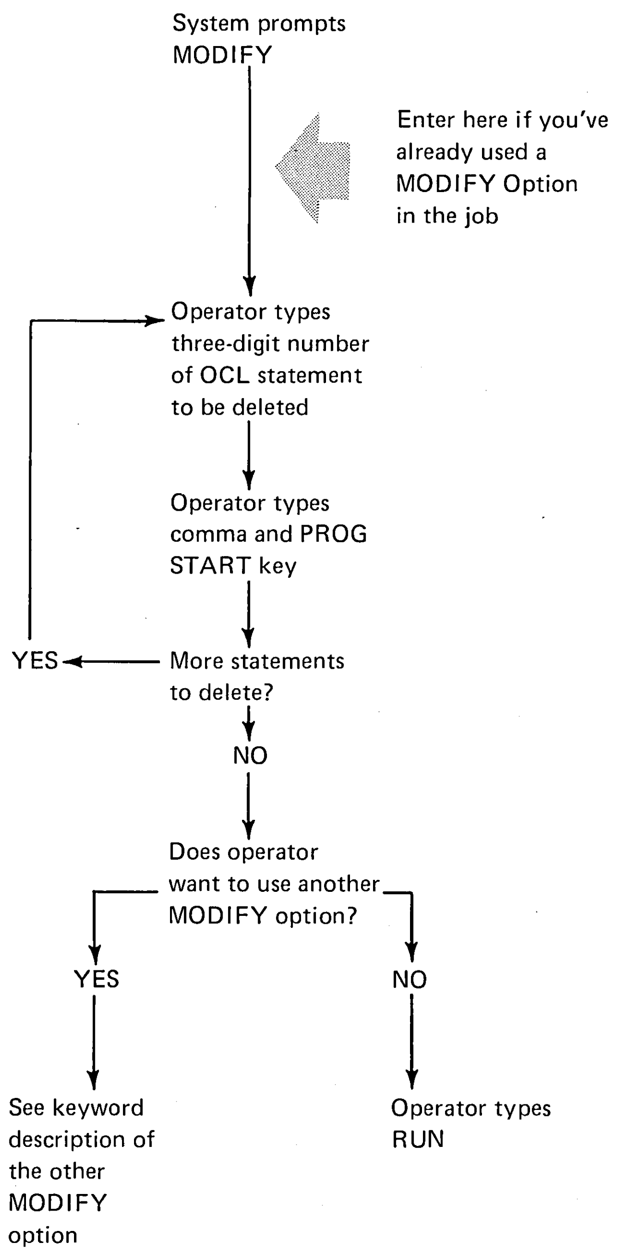

System prompts MODIFY
Do you want to correct a statement?
NO
YES
Operator types 3 digit statement number and corrected statement.
Do you want to delete a statement?
NO
YES
Operator types 3 digit statement number and comma (,).
Do you want to enter a comment?
NO
YES
Operator types asterisk (*) and comment.
Do you want to enter a FORMS statement?
NO
YES
Operator types FORMS and new lines per page setting.
A
A
Do you want to cancel job?
NO
YES
Operator types CANCEL
System erases OCL cycle
System prompts READY
Do you want to run the job?
NO
YES
Operator types RUN
System runs job
System prompts READY
Is this a BUILD cycle?
YES
NO
Do you want to include source statements for the Disk Sort Program or for one of the utility programs in the procedure?
YES
NO
2


2
Operator types INCLUDE
System prints 2-digit statement number
Operator types statement
System prints next statement number
Do you want to include another statement?
NO
YES
Operator types RUN
System prompts MODIFY
B
B
Do you want to change or delete any of the included statements?
NO
YES
Operator types 2-digit statement number and comma (to delete) or the new statement (to correct)
Do you want to cancel the job?
NO
YES
Operator types CANCEL
System erases entire OCL cycle
System prompts READY
Operator types RUN
System puts the procedure with included statements in the source library
System prompts READY

### Changing a Previous OCL Statement


System prompts MODIFY
Enter here if you've already used a MODIFY option in the job
Operator types three-digit number of OCL statement (or two-digit number of included statement) to be changed and PROG START
System tabs to position 35 (position 0 after INCLUDE) and waits for response
Operator types new response
More statements to change?
YES
NO
Does operator want to use another MODIFY option?
YES
NO
Operator types RUN
See keyword description of the other MODIFY option

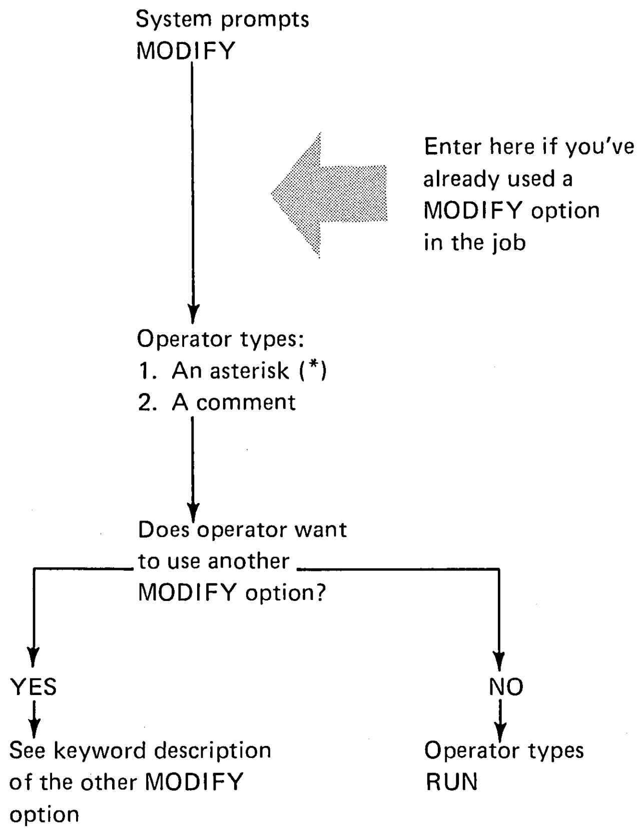
### Deleting a Previous OCL Statement


System prompts MODIFY
Enter here if you've already used a MODIFY Option in the job
Operator types three-digit number of OCL statement to be deleted
Operator types comma and PROG START key
More statements to delete?
YES
NO
Does operator want to use another MODIFY option?
YES
NO
See keyword description of the other MODIFY option
Operator types RUN

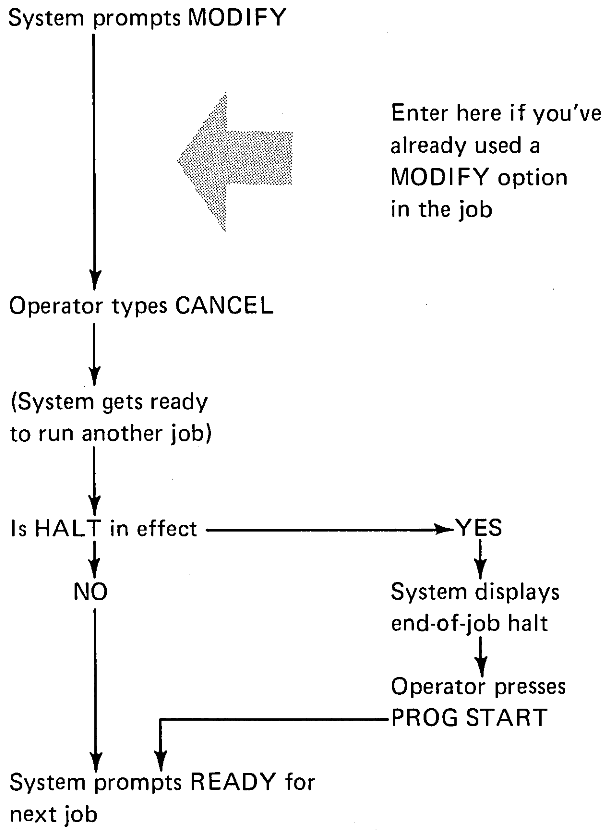
#### *Deleting Multiple Keywords*

When the OCL statement number for FILE NAME is deleted, all keywords for that file will be deleted from the cycle. For example, the LABEL or DATE keywords could be deleted from a file keyword statement without deleting the other keywords for that file. However, if FILE NAME is deleted, that entire file would be deleted from the cycle.

**Entering Comments**


System prompts MODIFY
Enter here if you've already used a MODIFY option in the job
Operator types:
1. An asterisk (*)
2. A comment
Does operator want to use another MODIFY option?
YES
NO
See keyword description of the other MODIFY option
Operator types RUN

**Cancelling Job**


System prompts MODIFY
Enter here if you've already used a MODIFY option in the job
Operator types CANCEL
(System gets ready to run another job)
Is HALT in effect
YES
NO
System displays end-of-job halt
Operator presses PROG START
System prompts READY for next job

*Points to Remember When Entering Comments*

- The usual purpose of a comment is to remind the operator of something he must do (mount a new disk pack, for example) or to document a problem during a program run.
- After the operator types a comment, it is immediately displayed on the system printer.
- Comments typed during a BUILD cycle become a permanent part of the procedure. They are entered into the Source Library along with OCL statements.
- Comments typed during a LOAD or CALL cycle do not become a permanent part of the job; their only purpose is to help document the program run.

*Effect of Entering CANCEL During a LOAD Cycle*

The entire LOAD cycle will be overlaid by the next OCL cycle.

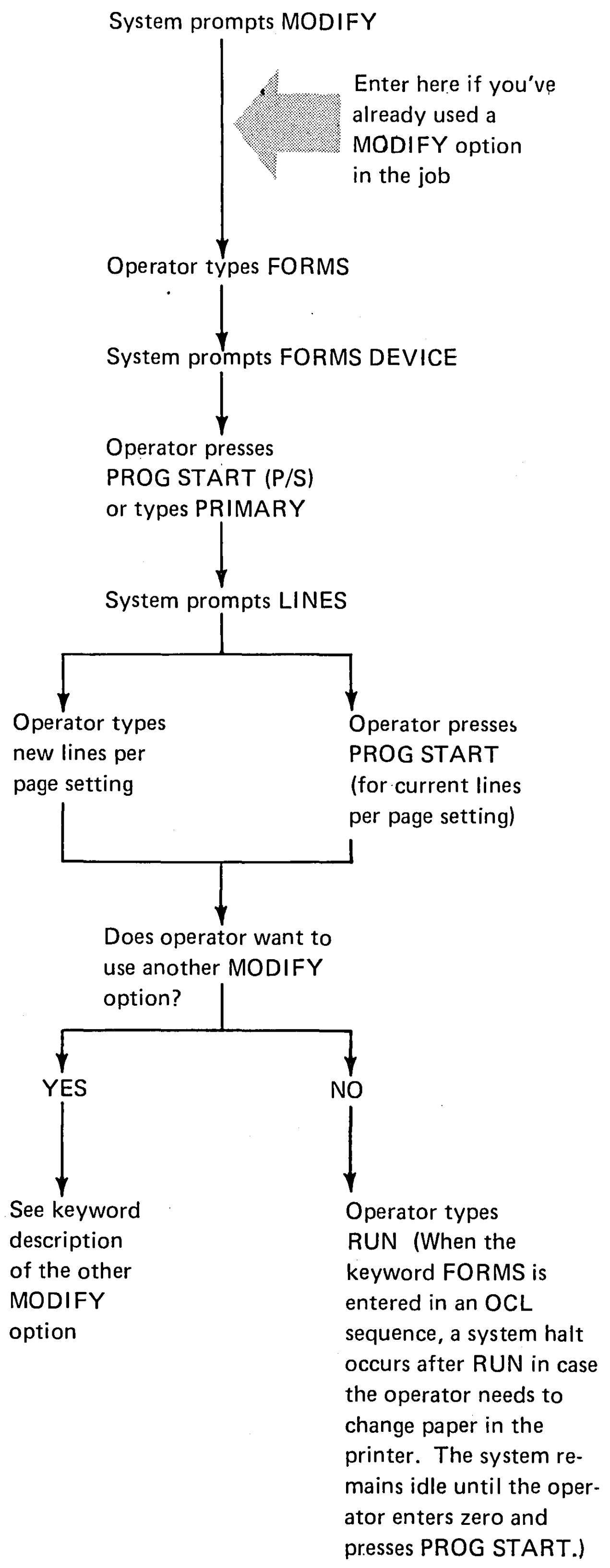
*Effect of Entering CANCEL During a BUILD Cycle*

The entire BUILD cycle will be overlaid by the next OCL cycle. (If a duplicate procedure is being built, and CANCEL entered, the original procedure remains in the source library. Except: if CANCEL is entered after INCLUDE, neither procedure will be in the library.)

*Effect of Entering CANCEL During a CALL Cycle*

The entire CALL cycle will be overlaid by the next OCL cycle. The original procedure will be unchanged.

**Changing Forms Length**


System prompts MODIFY
Enter here if you've already used a MODIFY option in the job
Operator types FORMS
System prompts FORMS DEVICE
Operator presses PROG START (P/S) or types PRIMARY
System prompts LINES
Operator types new lines per page setting
Operator presses PROG START (for current lines per page setting)
Does operator want to use another MODIFY option?
YES
NO
See keyword description of the other MODIFY option
Operator types RUN (When the keyword FORMS is entered in an OCL sequence, a system halt occurs after RUN in case the operator needs to change paper in the printer. The system remains idle until the operator enters zero and presses PROG START.)

*Purpose of FORMS*

Standard output for Model 6 printers is 66 lines per page. At IPL time, 66 lines per page is established as the forms length unless a different value was specified during system generation.

To change the lines per page of printed output for RPG II programs, you code line counter specifications. To change the lines per page of printed output for system programs (utilities, SORT, and the RPG Compiler), you type the keyword FORMS and an appropriate response.

If line counter specifications and an OCL FORMS statement are both used in one job, and if the specified lengths are different, the system will accept the RPG II line counter specifications and ignore the OCL FORMS statement.

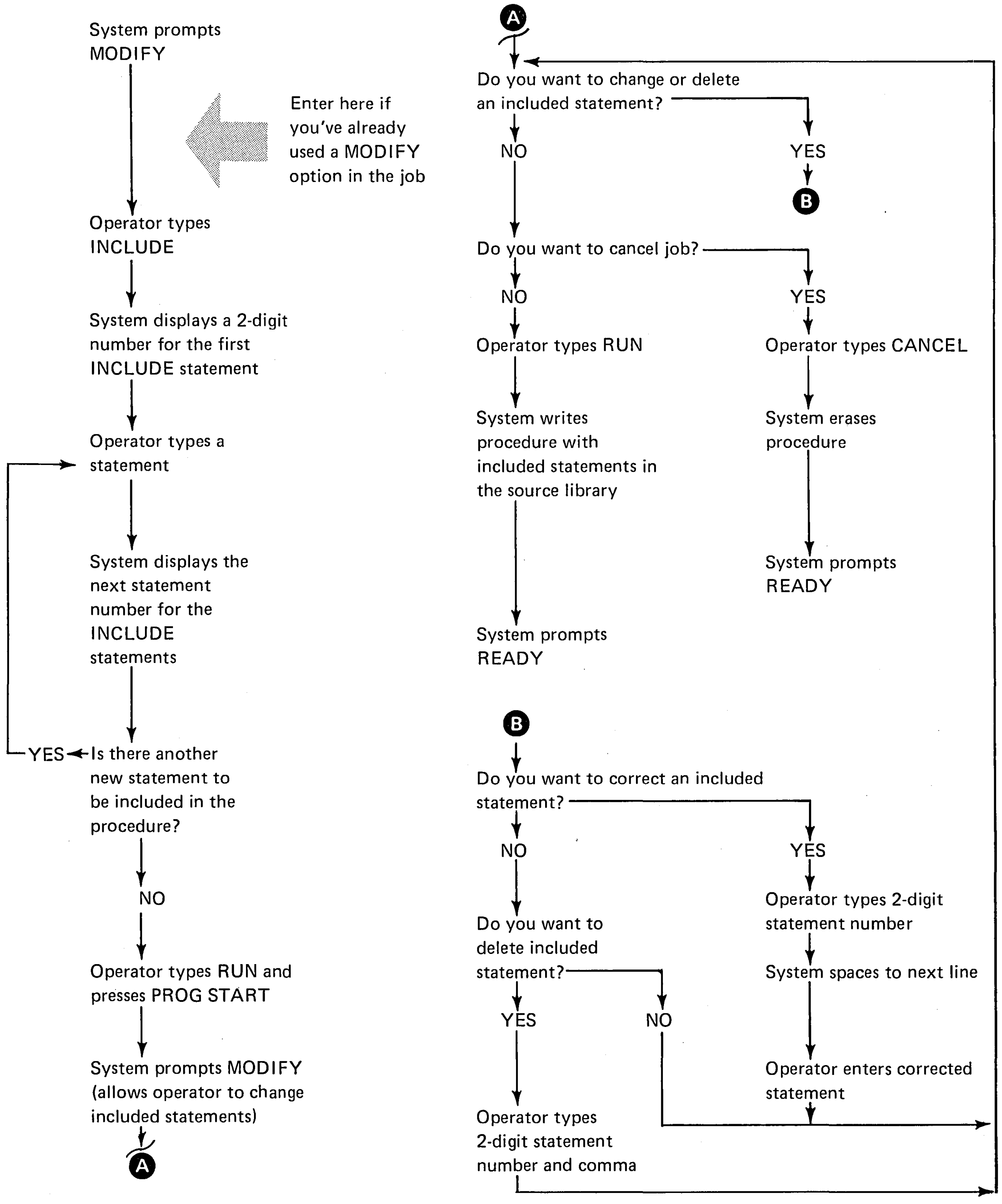
The new lines per page setting (from either an OCL FORMS statement or an RPG II line counter specification) remains effective until another OCL FORMS statement or RPG II line counter specification is read.

FORMS can be entered during the MODIFY phase of any OCL cycle. (The system never prompts FORMS.)

Whenever the operator types FORMS during an OCL cycle, a system halt follows RUN in case the operator needs to change the paper in the printer. Job processing does not resume until the operator enters a zero (option 0) and presses the PROG START key.

For additional operating information, including line counter considerations, related to the keyword FORMS, see the *IBM System/3 Model 6 Operator's Guide,* GC21-7501.

## Including Control Statements


System prompts MODIFY
Enter here if you've already used a MODIFY option in the job
Operator types INCLUDE
System displays a 2-digit number for the first INCLUDE statement
Operator types a statement
System displays the next statement number for the INCLUDE statements
Is there another new statement to be included in the procedure?
YES
NO
Operator types RUN and presses PROG START
System prompts MODIFY (allows operator to change included statements)
A
A
Do you want to change or delete an included statement?
NO
YES
B
Do you want to cancel job?
NO
YES
Operator types RUN
Operator types CANCEL
System writes procedure with included statements in the source library
System erases procedure
System prompts READY
System prompts READY
B
Do you want to correct an included statement?
NO
YES
Operator types 2-digit statement number
System spaces to next line
Operator enters corrected statement
Do you want to delete included statement?
YES
NO
Operator types 2-digit statement number and comma

## MODIFY (continued)

*Purpose of INCLUDE*

The keyword INCLUDE lets you add system program control statements to a procedure. INCLUDE tells the system that the next entry will be a set of control statements for one of the system programs. (As used here, control statements refer to both the control statements for the utility programs and the sequence specifications for the SORT program.) A maximum of 25 control statements can be included in each procedure.

*Restrictions After INCLUDE*

After including statements in a procedure, the procedure cannot be changed. MODIFY is prompted to allow changing included statements. If CANCEL is used after INCLUDE in a procedure that overlaid a duplicate procedure, neither the original nor the new procedure will be in the source library.

*Considerations During a CALL Cycle*

When the operator uses the CALL cycle to get the procedure out of the source library, the system displays the procedure in two separate steps: first the OCL statements, then the INCLUDE statements. The following shows details of the two display steps:

1. System displays OCL statements for the job.
   - System prompts MODIFY (to give operator a chance to correct any of the OCL statements).
   - Operator, after he has made any necessary corrections, types RUN.
2. System displays heading: INCLUDED STATEMENTS, then displays the INCLUDE statements.
   - System prompts MODIFY (to give operator a chance to correct any of the INCLUDE statements).
   - Operator, after he has made any necessary corrections, types RUN.
   - Model 6 runs the job.

If the operator presses the ENTER– key after responding to CALL NAME or UNIT, all steps except the last are omitted. The job is run without displaying the statements or prompting MODIFY. Statements with delayed responses are still displayed, to allow the operator to enter the response.

## NOHALT

Normally the system halts when a job ends. The operator can respond to the keyword READY with NOHALT. The system will then prompt READY for the next job when each job ends. The NOHALT will remain in effect until a HALT statement is entered or an IPL occurs.

## READY

When the system is ready to begin the OCL sequence for a new job, it prompts READY.

The operator responds by typing the name of one of the four OCL cycles: LOAD, BUILD, BUILDC, or CALL. The system then prompts the other keywords in the sequence.

(OCL cycles for the Model 6 are described in the *Summary of Conversational OCL* at the front of this manual.)

## RUN

RUN is the last entry in any OCL cycle. The operator types RUN when he is satisfied that the OCL cycle is complete and correct. The following table shows what happens when the operator types RUN during any of the three OCL cycles:

| **Sequence** | **Result** |
|---|---|
| LOAD | Job is run. |
| CALL | Job is run. |
| BUILD | The OCL statements are put in a source library. |

If INCLUDE statements are part of the procedure the BUILD and CALL cycles require two RUN entries. (See *Considerations During a CALL Cycle* under *MODIFY – Including Control Statements* in Part I.)

## RUN (continued)

After the operator types RUN, the system processes the job and end-of-job occurs. The system then prompts READY for the next job.

## SWITCH

The OCL SWITCH statement allows changing the eight external indicators used by RPG II programs.

(External indicators are discussed in the *IBM System/3 Model 6 RPG II Reference Manual;* SC21-7517.)

The operator-system interaction involved with the SWITCH statement is different for each OCL cycle as shown in the following charts.

### Indicator Settings

The indicator setting has eight positions, corresponding to the eight external indicators.

The three possible entries for each position are:

1 — sets corresponding indicator on.

0 — sets the corresponding indicator off.

X — leaves the corresponding indicator unchanged.

For example, if the operator keys in XXXX10XX:

Indicator five will be set on.

Indicator six will be set off.

Indicators one, two, three, four, seven, and eight will be unchanged.

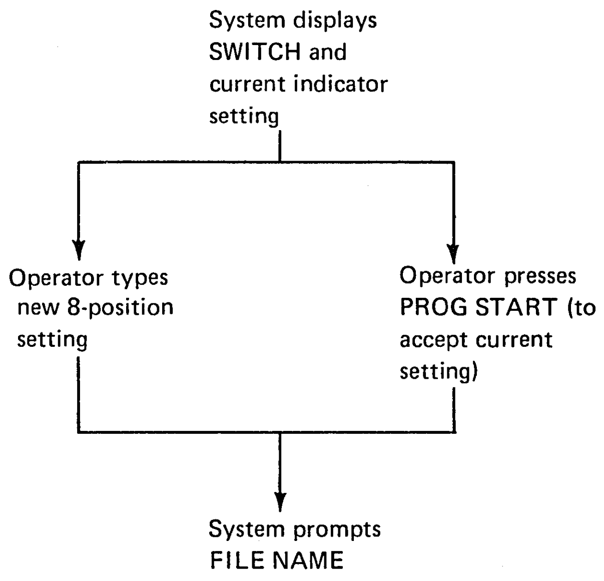
### IPL Considerations

All eight external indicators are set off at IPL. The only way to set an indicator on is by responding to the keyword SWITCH with a new eight-position response containing a 1 in the appropriate position.

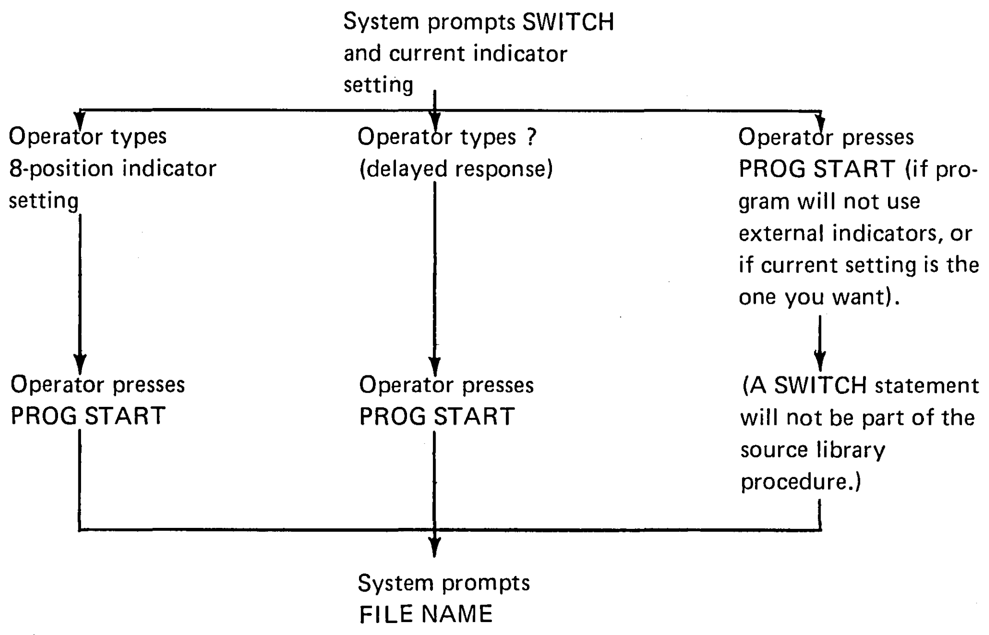
### Duration of SWITCH Setting

When an OCL SWITCH statement sets an indicator on, the indicator remains on until another SWITCH statement sets it off or the next IPL occurs.

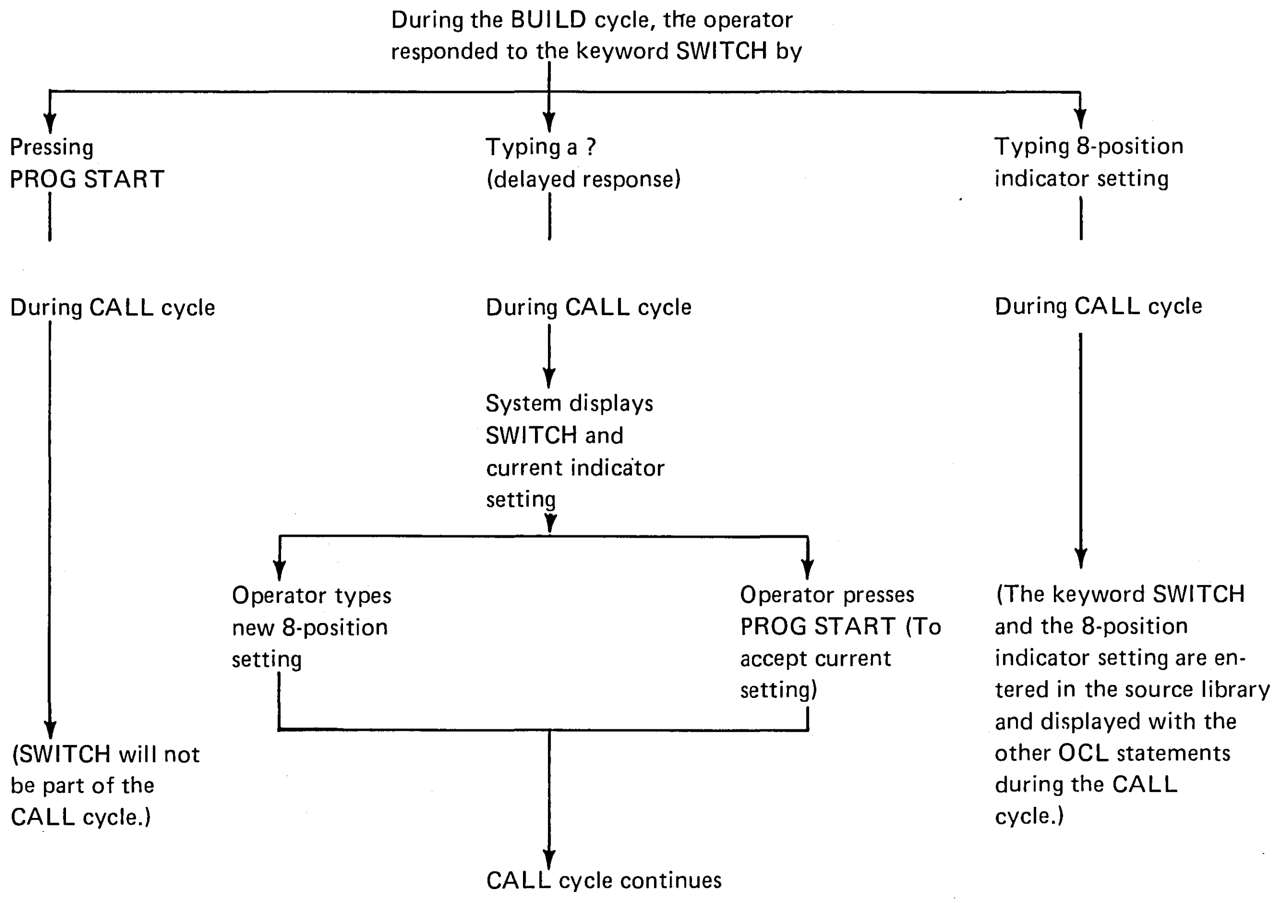
### Operator-System Interaction for SWITCH Statement (LOAD Cycle)


System displays SWITCH and current indicator setting
Operator types new 8-position setting
Operator presses PROG START (to accept current setting)
System prompts FILE NAME

**Operator-System Interaction For SWITCH Statement (BUILD Cycle)**


System prompts SWITCH and current indicator setting
Operator types 8-position indicator setting
Operator types ? (delayed response)
Operator presses PROG START (if program will not use external indicators, or if current setting is the one you want).
Operator presses PROG START
Operator presses PROG START
(A SWITCH statement will not be part of the source library procedure.)
System prompts FILE NAME

**Operator-System Interaction for SWITCH Statement (CALL Cycle)**


During the BUILD cycle, the operator responded to the keyword SWITCH by
Pressing PROG START
Typing a ? (delayed response)
Typing 8-position indicator setting
During CALL cycle
During CALL cycle
During CALL cycle
System displays SWITCH and current indicator setting
Operator types new 8-position setting
Operator presses PROG START (To accept current setting)
(SWITCH will not be part of the CALL cycle.)
(The keyword SWITCH and the 8-position indicator setting are entered in the source library and displayed with the other OCL statements during the CALL cycle.)
CALL cycle continues

## MULTI-VOLUME FILES

### File Statements for Multi-Volume Files

If a file is too large for one disk, you can continue it on one or more subsequent disks. Such files are called multi-volume files. (A volume is one disk.) Multi-volume files can be online or offline. A file is online if all volumes are mounted when the job begins. The UNIT and PACK parameters are equal. An offline file has fewer UNIT parameters (shares same unit).

*Creation*

The ways that you can create a multi-volume file depend on the type of file you are creating. For a consecutive and indexed file, the records are stored in consectuive locations on disk, in the order that they are read. One disk is filled at a time.

For consecutive files, each volume must be filled before the next volume is loaded. For indexed files, each volume need not be filled. Each indexed volume is loaded until a key-field is reached that is higher than the HIKEY for that volume, then the next volume is loaded. Indexed files must be loaded in keyfield sequence. A halt occurs if a volume is filled and there is not a record with a keyfield equal to the HIKEY for that volume. For example, suppose the HIKEY for a volume is 199. You load a record with the keyfield 195. It is less than the HIKEY, so it is loaded on the volume. Next, you load a record with the keyfield 200. Record 200 would be loaded on the next volume, and a halt would occur. The reason for the halt is that you did not load a keyfield record equal to 199 before you jumped to a new volume. This halt can be ignored. You can load the next volume and at some future time insert a keyfield record equal to the HIKEY. To insert a record after the loading sequence has passed, a random add must be done.

Indexed and consecutive files may be either online or offline.

If using removable disks when creating consecutive or indexed files you can mount a disk, wait until the system indicates it is filled. Then, mount the next disk. If you have two drives, you can mount the two disks, wait until the first one is filled, then replace it with the third while your program fills the second disk. In either case, you cannot use more than 40 disks per job.

Space can be allocated on all volumes of a multi-volume file if the volumes are online at the time of the allocation. Space can also be allocated for an offline file, other than the initial volume, but the packs must be empty packs or space (TRACKS and LOCATION) known to be available. You can use both fixed and removable disks with any online multi-volume file.

Space on a volume of a multi-volume file is reserved after one or more records are placed in that volume.

Direct files must be online. Direct files are created in a non-consecutive manner. When creating such files, you are required to mount all the disks on your disk unit at the same time. The maximum number of disks you could use, therefore, is two if you have only one drive, or three or four if you have two drives.

*Processing*

The ways in which you can process multi-volume files depend on the method your program uses to get records from the file. If records are read from a consecutive or indexed file, you can mount a disk, wait until all of the records have been read from the disk, then mount the next disk. If you have two drives, you can mount two disks, wait until all of the records have been read from the first disk, then replace that disk with the third while your program reads from the second disk. When you are processing files offline the disks must be removable. When online, any combination of fixed and removable disks is acceptable, but all must be mounted and must remain mounted.

### OCL Considerations

Multi-volume files, like other disk files, must be described in FILE statements. However, because a multi-volume file involves more than one disk, some FILE keywords require a list of data or codes to describe all of the disks containing the files. This section explains the considerations for using these lists. Each list must begin and end with apostrophes.

**List Requirements**

The PACK parameter requires a list. The UNIT parameter may require a list while LOCATION, TRACKS, HIKEY, and RECORDS require a list if they are stated. The considerations for using the lists in these parameters are included in the keyword discussions following.

KEY LENGTH: This keyword will be prompted if the response to FILE NAME indicated a multi-volume file (see *Enter Minus* under *End-of-Statement Keys* in Part I). If this is an indexed file, you must respond to KEY LENGTH with a two-digit number 01 through 29. If this is not an indexed file pressing the PROG START key will skip the HIKEY keyword.

HIKEY: This keyword must be answered for indexed files. The highest keyfield for each volume must be entered. All characters except commas are allowed as keys. The length of each HIKEY must equal the response to KEY LENGTH and a HIKEY must be entered for each volume. If a HIKEY with fewer characters is entered, blanks will be put into the remaining positions. If an apostrophe is used as part of a HIKEY, it must be entered as two apostrophes or it will be decoded at the end of HIKEY list and an error will occur. When using only one volume of an indexed multi-volume file, the HIKEY must be entered with beginning and ending apostrophes.

The keys in an indexed file can be packed numeric characters. To indicate that a file has packed keys, the operator responds to KEY LENGTH with nn,P where nn is 01-08. Only numeric characters (0-9) are allowed in packed HIKEYS. When responding to HIKEY, the number of characters entered per key is equal to 2nn—1. If the KEY LENGTH response is 07, the HIKEYS would be 13 characters long.

UNIT: The keyword UNIT must be followed by a code or codes indicating where the disks that contain the file will be located on the disk unit. No UNIT parameter may be repeated. The codes are as follows:

| **Code** | **Meaning** |
|---|---|
| R1 | Removable disk on drive one. |
| F1 | Fixed disk on drive one. |
| R2 | Removable disk on drive two. |
| F2 | Fixed disk on drive two. |

The order of codes in the UNIT parameter must correspond to the order of names in the PACK parameter.

When you are creating or processing a consecutive or indexed file, you can use the same drive for more than one of the disks; however, the units must then all be removable units. If they are, you must not repeat the code for the drive in the UNIT parameter. When the number of codes in the UNIT parameter is less than the number of names in the PACK parameter, the system uses the codes alternately.

If F1 or F2 is specified, the file must be online multi-volume.

PACK: The names of the disks that contain, or will contain, the multi-volume file must follow the keyword PACK. (PACK names must be unique for proper functioning.)

When a multi-volume file is created, the system writes a sequence number on the disks to indicate the order of the disks. The disks are numbered in the order in which you list their names in the PACK parameter.

When a multi-volume file is processed, the system provides two checks to ensure that the disks are used in the proper order:

1. It checks to ensure that the disks are used in the order that their names are listed in the PACK parameter.

2. It checks the sequence numbers of the disks used to ensure that they are consecutive and in ascending order (01, 02, and so on).

The system stops when it detects a disk that is out of sequence. The operator can do one of three things:

1. Mount the proper disk and restart the system.

2. Restart the system and process the disk that is mounted if the sequence is ascending (for consecutive input and update).

3. End the program.

Consecutive input or update sequence numbers are ignored if the file was not created as multi-volume. If the file is multi-volume and the sequence is ascending but not consecutive, a diagnostic halt is given which allows the proceed option.

TRACKS or RECORDS: The keyword TRACKS or RECORDS must be followed by numbers that indicate the amount of space needed on each of the disks that will contain the multi-volume file. TRACKS or RECORDS must be specified. Any multi-volume file load requires a TRACKS or RECORDS keyword whether the file previously existed or not. The order of these numbers must correspond to the order of the names in the PACK parameter.

LOCATION: The keyword LOCATION must be followed by the numbers of the tracks on which the file is to begin on each of the disks you use for the file. The order of the numbers must correspond to the order of the names in the PACK parameter. If you omit the LOCATION parameter, the system chooses the beginning track on each of the disks. If LOCATION is specified for one disk, it must be specified for all disks. If the multi-volume file exists, LOCATION must be given and must be identical to the LOCATION parameter specified when the file was created.

RETAIN: RETAIN-S must not be specified unless the file is online multi-volume. If RETAIN-S is used for online multi-volume, it cannot be changed to RETAIN-T unless also done online.

## File Statement Summary

| KEYWORDS | | SEQUENTIAL FILES | DIRECT FILES |
|---|---|---|---|
| UNIT | Maximum Number of Disks | 10 disks per file statement, 40 disks per job (number of HIKEYS plus number of packs cannot exceed 40) | Single Drive–2 disks<br>Two Drives–4 disks |
| UNIT | Location Requirements | R1 or R2 for offline files<br>No restriction for online files | No restriction |
| UNIT | Restrictions on Disk | At file creation time only:<br>● First disk can also contain programs, procedures, other files.<br>● Remaining disks must be used only for the one file. | All the disks used for the file can also contain programs, procedures, other files. |
| UNIT | Operating Considerations | Single Drive – Disks must be mounted one at a time.<br>Two Drives – Disks must be mounted in sequence specified in UNIT statement. | All disks must be on-line during processing. |
| UNIT | Relation to | One entry in the UNIT statement can correspond to more than one disk name in the PACK statement. | A one-to-one correspondence is required between the entries in the UNIT statement and the disk names in the PACK statement. |
| PACK | | When processing a file (or a subset of a file) the disk names must be in the same sequence as they were at file creation time. | |
| KEY LENGTH | | Length must be less than 30 (01–08 if packed keys). | Used only for Indexed-Sequential Files. For Consecutive-Sequential and Direct files, pressing PROG START will also bypass HIKEY prompt. |
| HIKEY | | HIKEY responses must correspond one-for-one with the disk names in the PACK statement. | Used only for Indexed-Sequential Files. For Consecutive-Sequential and Direct files, pressing PROG START will also bypass HIKEY prompt. |
| TRACK or RECORDS | | At file creation time:<br>● Number of tracks (or records) must be specified for each disk.<br>● Number in TRACKS (or RECORDS) statement must correspond one-for-one with the disk names in the PACK statement.<br>During subsequent runs: TRACKS (or RECORDS) statement can be included in the OCL sequence. (For greater detail see keyword descriptions of TRACKS/RECORDS.) | |
| LOCATION | | ● If specified:<br>Addresses must correspond, one-for-one with disk names in PACK statement.<br>● If not specified:<br>System will allocate space on each disk. | |

**Coding Multi-Volume File Statements**

1. The operator must begin and end each statement with an apostrophe.
2. The system displays information about each volume on a separate line.
3. The system assigns one statement number to the entire file statement.

**Changing Multi-Volume File Statements with MODIFY Keyword**

When using MODIFY keyword to change a multi-volume file statement (other than HIKEY), the entire response to the keyword must be re-entered on one line, separated by commas, with beginning and ending apostrophes.

*Example*

|  | UNIT Statement is | Should be |
|---|---|---|
| 041 | UNIT – 'F1 | UNIT – 'F1 |
|  | – R1 | – R1 |
|  | – R2 | – F2 |
|  | – F2' | – R2' |

To change at MODIFY time

```
MODIFY
041              – 'F1,R1,F2,R2'
RUN
```

## INCLUDING SORT SOURCE OR UTILITY CONTROL STATEMENTS IN A PROCEDURE

The INCLUDE option can be used during MODIFY time of a BUILD cycle to include sort source or utility control statements in a procedure. This is useful if the control statements are long or complex and the job is run frequently. A maximum of 25 control statements can be included in each procedure.

During the BUILD cycle, the INCLUDE option must be the last MODIFY option used. After the included statements are keyed in, the RUN entry then puts the procedure and included statements in the source library.

The CALL cycle will be different if the called procedure has included statements. After the OCL statements are printed, MODIFY will be prompted to allow changes to the OCL statements. After the operator types-RUN, the system will print INCLUDED STATEMENTS and then list the statements. MODIFY will now be prompted again, to allow changes to be made to the included statements. The operator types RUN to run the job.

For an example of Including Sort Source Statements in a procedure see the *IBM System/3 Disk Sort Reference Manual*, SC21-7522.

An example of including Utility Control statements in a procedure is shown in sample job 10 (see *Sample Jobs* at end of this part).

## INCREASING FILE SIZE OF THE RPG PROCEDURE

The IBM-supplied compile procedure can only compile RPG II programs with less than 400 statements. To compile larger programs, the file statements must be modified to increase their size above 10 tracks (see *Modify; Changing a Previous OCL Statement* in Part I). Using the MODIFY option will only increase the file size for one compile. The RPG II procedure will not be changed in the source library. To change the procedure in the source library you must either build a new procedure (see *BUILD NAME* in Part 1), use the Library Maintenance Modify function, or use the KSE utility program.

## PROCESSING LARGE INDEXED DISK FILES

When additions are made to a large indexed file, the amount of time needed to sort the keys of the index at end-of-job may become excessive. This sort time can be reduced by using a work file.

The work file is used to merge the added keys into the index and must be large enough to contain all of the keys added to the file. If the program adds records to more than one indexed file, the work file must be large enough to contain all the keys added to the file having the greatest number of additions.

The work file must be named $INDEX44 and should be located as close as possible to the index being sorted. To compute the number of tracks required for the work file, use the following formula:

$$\text{number of adds} \div \left(\frac{256}{\text{keylength}+3}\right) \div 24 = \text{tracks}$$

After dividing 256 by keylength+3, the remainder should be dropped. After the other divisions, round the quotient to the next higher whole number.

If the work file is not large enough to contain all the index keys, the keys are sorted in the normal manner without using a work file. If possible, the work file should be located on a different disk drive than the indexed file whose keys are being sorted. If this is not possible, the work file should be as close as possible to the beginning of the file whose keys are being sorted. This minimizes the disk seek time.

The work file can be used with multivolume files. However, it cannot be located on a pack that contains one of the offline volumes of a multivolume file. The pack containing the work file must remain online while running the job. The work file must be RETAIN-S. If RETAIN-T or RETAIN-P is specified, the system will default to RETAIN-S.

For small indexed files (10 tracks or less) where the sort time is negligible, the use of the work file will not improve performance and should not be used.

To use this performance option, no change is needed to your source program. Also, programs need not be recompiled to use this option. Only the additional OCL FILE statement is needed to use this option.

## ENTERING RPG II SOURCE STATEMENTS FROM THE KEYBOARD AT COMPILE TIME

The IBM-supplied compile procedure requires that the RGP II source statements be in the source library of a disk. By using the Keyboard Source Entry Utility ($KSE), source statements can be format checked as they are put on disk.

The source statements can, however, be entered from the keyboard at compile time. These statements are read by the compiler and checked for format errors. If any errors are found they cannot be corrected and the compile will not be successful. The compile job must be rerun and all source statements keyed in again.

To key in source statements from the keyboard, the IBM-supplied compile procedure RPG is used. This procedure does not prompt COMPILE OBJECT, SOURCE, or UNIT.

### Inquiry Interrupt

Certain programs can be interrupted while they are being processed. A request for interruption is called an inquiry request (made by depression of the inquiry key on the keyboard). Programs are usually interrupted to permit another program to run. Control is then given back to the first program.

The instructions given the compiler at compile time determine the inquiry type of a program.

The three types of programs include:

1. A program that cannot be interrupted (does not recognize an inquiry request).
2. A program that can be interrupted (does recognize an inquiry request). This is a B-type inquiry program.
3. An inquiry program that can only be executed when an inquiry request is made. This is an I-type program.

Usually I-type programs are read in only when a program is interrupted. In this case the inquiry program will not recognize an inquiry request. However, if an inquiry program is loaded in the normal manner (not because of a program interrupt), it can only be executed when an inquiry request is made. While this program is running, it will not recognize an inquiry request.

The inquiry interrupt involves these three steps:

1. When the program recognizes an inquiry request, a Roll-Out routine moves the interrupted program from main storage to disk.

2. The program for which the interrupt was requested must be loaded normally. The interrupting program may be any type. This interrupting program cannot be interrupted.

3. After the interrupting program is executed, the interrupted program moves back into main storage using a Roll-In routine. The interrupted program begins execution at the point of interruption and terminates in a normal manner.

The *IBM System/3 Model 6 RPG II Reference Manual*, SC21-7517, describes coding necessary to define inquirable programs.

### Restrictions During Inquiry

Inquiry always causes the conversational OCL scheduler to be used, even if the interrupted program was running under the card scheduler. The OCL statements cannot be read from cards during inquiry.

The Log device cannot be changed during inquiry.

## CHAINED PROCEDURES

A finished job usually requires that more than one program be run. Several customer programs with utility programs between them may be required to complete the finished report. This sequence of programs can be put in chained procedures.

By chaining procedures, several benefits can be realized, including:

- Programs are always run in the correct sequence.

- Operator intervention and, therefore, chance of operator error, is decreased.

- File space can be saved. Files used to pass data from job to job can be scratched after the last program.

- Files are less likely to be destroyed by running non-related programs between programs of a job.

To chain procedures, the operator first builds a master procedure to chain together other procedures. This is done by responding to READY with BUILDC. The system will then repetitively prompt CALL NAME and UNIT, allowing the operator to respond with the name and unit of the procedures that are to be chained. When all procedure names have been entered, the operator responds to CALL NAME or UNIT with the ENTER MINUS (ENTER–) key. The system then allows the operator to MODIFY the entries. When RUN is entered, the master procedure is put in the source library as a permanent entry.

Master procedures can call other master procedures up to 9 levels. The original master procedure called (level 1) can call another master procedure (level 2), which can call another master procedure (level 3), etc., on up to 9 levels. Care must be taken to avoid calling a master procedure that was already called earlier in the chain or an endless loop will result. A master procedure can contain only CALL and UNIT statements.

Delayed responses are not allowed in a BUILDC cycle. However, the called procedures can contain delayed responses.

To run the chained procedures, the operator initiates a CALL cycle and responds to CALL NAME with the name of the master procedure. Each procedure is then called by the master procedure and run.

When running chained procedures, the operator is never prompted MODIFY to make changes.

If the operator presses the ENTER–key after responding to CALL NAME or UNIT, only the CALL NAME and UNIT statements of each chained procedure will be displayed. All other OCL statements (except those with delayed responses) and included control statements are not displayed.

If HALT is specified, the system will not halt until the last job of a chain is complete.

## OCL FOR THE IBM 2222 PRINTER

The IBM 2222 printer provides the MODEL 6 system with the ability to print on two forms. Each form has its own forms tractor. The left tractor is called PRIMARY and the right tractor is SECONDARY.

### Using the FORMS Statement

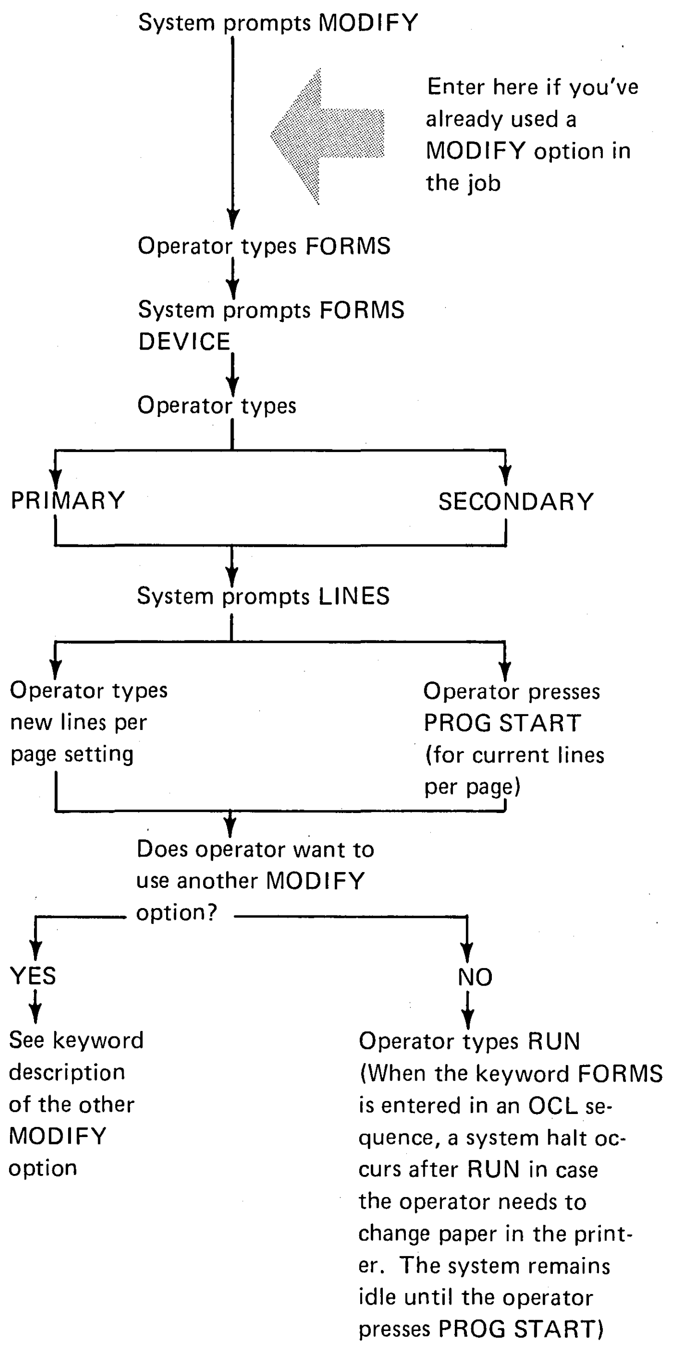
The lines per page setting of the PRIMARY and SECONDARY tractors can be different. (For example, the PRIMARY tractor could print 25 lines per page, while SECONDARY prints the standard 66 lines per page.) Separate settings are specified by entering different FORMS statements for each tractor during the MODIFY phase.

### Log Device

The log device is used to print OCL statements and error messages and codes. The PRIMARY tractor will be the log device at IPL time when the 2222 Printer is used. The secondary tractor can be assigned as the logging device by entering LOG at either READY or MODIFY time. If the secondary tractor is the logging device, logged data begins in print position 110. (See *READY-Entering LOG and MODIFY-Entering LOG*).

If the log device is used for normal program output, the error messages and codes are not printed.

### MODIFY – Entering the Keyword FORMS



System prompts MODIFY

Enter here if you've already used a MODIFY option in the job

Operator types FORMS

System prompts FORMS DEVICE

Operator types

PRIMARY

SECONDARY

System prompts LINES

Operator types new lines per page setting

Operator presses PROG START (for current lines per page)

Does operator want to use another MODIFY option?

YES

See keyword description of the other MODIFY option

NO

Operator types RUN (When the keyword FORMS is entered in an OCL sequence, a system halt occurs after RUN in case the operator needs to change paper in the printer. The system remains idle until the operator presses PROG START)

## OCL FOR THE IBM 2265-2 DISPLAY

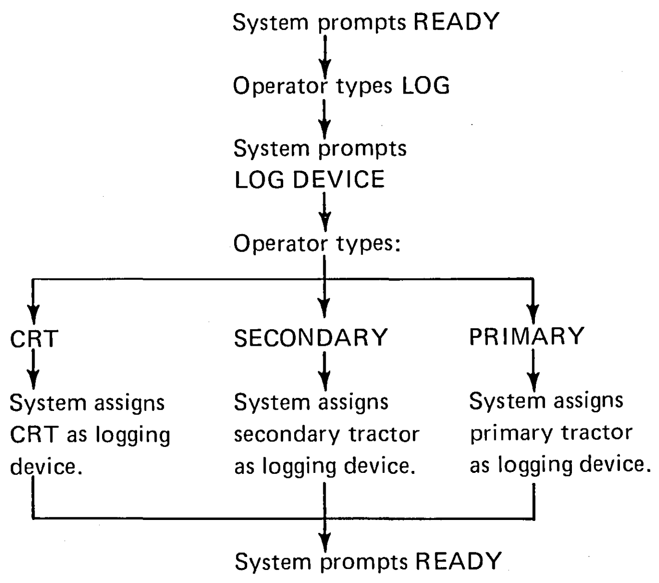
The IBM 2265-2 display unit can be used as the system logging device. The logging device displays conversational OCL statements, utility control statements, job comments, and error messages and codes. The log device can also be used for normal output from the job being run. Error messages and codes are not displayed if the 2265-2 is used for normal job output.

When the 2265-2 (CRT) is used as the logging device, an additional 1K of core storage is needed for the system, thus reducing the core available for the user program. This extra core is not needed if the user program specifies the CRT as an output device.

The operator can assign either the CRT display or the printer as the logging device. If the operator changes the logging device the change remains in effect until either:

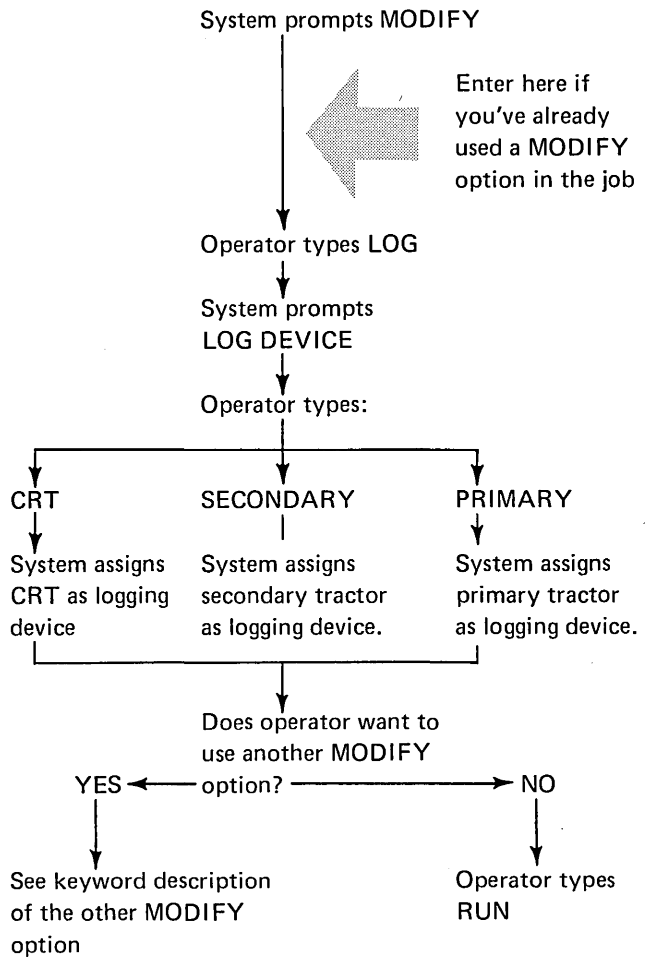
- The operator specifically overrides the change with another LOG statement.
- The next IPL procedure.

### READY – Entering LOG


System prompts READY
Operator types LOG
System prompts LOG DEVICE
Operator types:
CRT
SECONDARY
PRIMARY
System assigns CRT as logging device.
System assigns secondary tractor as logging device.
System assigns primary tractor as logging device.
System prompts READY

*Note:* The CPU usage meter will continue to run during halts (other than end-of-job halt in halt mode) when the CRT is used as the logging device or when it is used by the customer program. To stop the usage meter, the system START/STOP switch should be moved to the STOP position. This will blank the CRT display, but the halt will continue to be displayed in the halt code indicator lights on the system console. When halt ABCD12345 occurs (end-of-job in HALT mode), the CRT is blanked and the usage meter is stopped.

### MODIFY – Entering LOG


System prompts MODIFY
Enter here if you've already used a MODIFY option in the job
Operator types LOG
System prompts LOG DEVICE
Operator types:
CRT
SECONDARY
PRIMARY
System assigns CRT as logging device
System assigns secondary tractor as logging device.
System assigns primary tractor as logging device.
Does operator want to use another MODIFY option?
YES
NO
See keyword description of the other MODIFY option
Operator types RUN

## OCL ERROR MESSAGES

| Message | Explanation |
|---|---|
| MESSAGE #00 – NO PROGRAM NAME GIVEN | Response to LOAD NAME was blank. |
| MESSAGE #01 – NO UNIT GIVEN | Response to UNIT was blank. |
| MESSAGE #02 – INVALID PROGRAM NAME SPECIFIED | Response to LOAD NAME was invalid. |
| MESSAGE #03 – INVALID UNIT SPECIFIED | Response to UNIT was invalid. |
| MESSAGE #04 – PROGRAM NOT FOUND ON SPECIFIED UNIT | The program indicated by your response to LOAD NAME was not found in the object library of the unit specified. |
| MESSAGE #05 – NO PROCEDURE NAME GIVEN | Response to CALL NAME or BUILD NAME was blank. |
| MESSAGE #06 – SOURCE NOT FOUND ON SPECIFIED UNIT | The source module specified by your response to SOURCE was not found in the source library of the unit specified. |
| MESSAGE #07 – INVALID PROCEDURE NAME | Response to BUILD NAME or CALL NAME was invalid. |
| MESSAGE #08 – MULTIVOLUME FILE RESPONSES NOT IN 1-1 RATIO | The number of responses to file keywords PACK, HIKEY, LOCATION, TRACKS, or RECORDS were not equal. |
| MESSAGE #09 – PROCEDURE NOT FOUND ON SPECIFIED UNIT | Procedure specified by response to CALL NAME was not found in source library of the unit specified. |
| MESSAGE #10 – INVALID SWITCH SETTINGS | Response to SWITCH was other than eight positions of X, 1, or 0. |
| MESSAGE #11 – NO SOURCE NAME GIVEN | Response to SOURCE was blank. |
| MESSAGE #12 – INVALID SOURCE NAME SPECIFIED | Response to SOURCE was invalid. |
| MESSAGE #13 – INVALID DATE SPECIFIED | Response to DATE in file keywords was invalid. |
| MESSAGE #14 – TOO MANY RESPONSES TO A MULTIVOLUME FILE KEYWORD | Only ten volumes are allowed in each multivolume file. |
| MESSAGE #15 – NO FILE NAME GIVEN | Procedure contains file keywords but no FILE NAME response. |
| MESSAGE #16 – NO PACK GIVEN | Procedure contains file keywords but no PACK response. |
| MESSAGE #17 – INVALID FILE NAME SPECIFIED | Response to FILE NAME was invalid. |
| MESSAGE #18 – INVALID LABEL SPECIFIED | Response to LABEL was invalid. |
| MESSAGE #19 – INVALID PACK SPECIFIED | Response to PACK was invalid. |

| Message | Explanation |
|---|---|
| MESSAGE #20 – INVALID RETAIN DESIGNATION SPECIFIED | Response to RETAIN other than P, T, S, or A. |
| MESSAGE #21 – INVALID TRACKS SPECIFIED | |
| MESSAGE #22 – MAXIMUM FILE STATEMENTS ENTERED | No more than 15 FILE statements can be specified in a job. |
| MESSAGE #23 – BOTH TRACKS AND RECORDS SPECIFIED | Procedure contains responses to both TRACKS and RECORDS. |
| MESSAGE #24 – INVALID RECORDS SPECIFIED | |
| MESSAGE #25 – INVALID LOCATION SPECIFIED | Response to LOCATION must be 8 through 405. |
| MESSAGE #26 – DEVICE NOT SUPPORTED | CRT or data recorder was specified but is not on the system. |
| MESSAGE #27 – INVALID DEVICE | Response to DEVICE or READER invalid. |
| MESSAGE #28 – INVALID NUMBER OF LINES | Response to LINES not between 12 and 112. |
| MESSAGE #29 – INVALID REQUEST | Response to MODIFY was invalid. |
| MESSAGE #30 – INVALID STATEMENT NUMBER | Invalid statement number entered as response to modify. |
| MESSAGE #31 – TOO MANY UTILITY CONTROL STATEMENTS IN PROCEDURE–JOB CANCELED | |
| MESSAGE #32 – RUN OUT OF SPACE IN THE SCHEDULER WORK AREA | |
| MESSAGE #33 – RESPONSE REQUIRED–DELAYED RESPONSE IN CALLED PROCEDURE | |
| MESSAGE #34 – TOO MANY MULTIVOLUME FILE UNITS SPECIFIED | Number of units specified exceeds number of packs specified. |
| MESSAGE #35 – DELAYED RESPONSE (?) NOT ALLOWED | |
| MESSAGE #36 – JOB CANCELED | /* was entered or job was canceled because of errors. |
| MESSAGE #37 – MULTIVOLUME FILE NOT VALID THIS STATEMENT | Multiple responses not allowed for this keyword. |
| MESSAGE #38 – ENTER MINUS (-) NOT ALLOWED | The ENTER– key is only allowed for certain keywords in the BUILD cycle. |

| Message | Explanation |
|---|---|
| MESSAGE #39 – ERRORS IN PROCEDURE–JOB CANCELED | |
| MESSAGE #40 – ERRORS IN OCL STATEMENT | |
| MESSAGE #41 – ERRORS IN RESPONSE | |
| MESSAGE #42 – DUPLICATE PROCEDURE NAME IN LIBRARY | Response to BUILD NAME is already in source library of unit specified. |
| MESSAGE #43 – DUPLICATE PROCEDURE DELETED | New procedure being entered will overlay old procedure with same name. |
| MESSAGE #44 – INVALID KEYWORD | Keyword found in procedure is invalid, or response to READY is invalid. |
| MESSAGE #45 – TOO MANY UTILITY CONTROL STATEMENTS ENTERED | Only 25 utility control statements may be entered. |
| MESSAGE #46 – PERMANENT DISK ERROR | |
| MESSAGE #47 – RUN OUT OF SPACE IN PROCEDURE LIBRARY–JOB CANCELED | |
| MESSAGE #48 – INVALID SYSTEM DATE SPECIFIED | |
| MESSAGE #49 – DUPLICATE KEYWORD | A procedure contains a duplicate keyword. |
| MESSAGE #50 – RESPONSE REQUIRED | You must respond to this keyword; PROG START as the only response is not allowed. |
| MESSAGE #51 – TOO MANY PACKS, HIKEYS, OR BOTH SPECIFIED | The total number of PACK and HIKEY keywords cannot exceed 52. |
| MESSAGE #52 – DUPLICATE MULTIVOLUME FILE UNIT SPECIFIED | |
| MESSAGE #53 – INVALID RESPONSE DURING INQUIRY | Cannot change logging device or change to card OCL. |
| MESSAGE #54 – INVALID HIKEY SPECIFIED | Response (number) to HIKEY exceeds response (number) to KEY LENGTH |
| MESSAGE #55 – INVALID HIKEY LENGTH SPECIFIED | Response to KEY LENGTH is greater than 29, or is 00. |
| MESSAGE #56 – HIKEYS OUT OF SEQUENCE | Responses to HIKEY must be in ascending sequence. |

## CO-RESIDENT SYSTEMS

IBM System/3 Model 6 users who have co-resident systems (both disk system management and System/3 BASIC) can transfer control from disk system management to System/3 BASIC by responding to READY with ENTER BASIC.

## Sample Jobs

This section presents a typical sequence of jobs:

- Initialize a disk.
- Compile an RPG II source program.
- Run the compiled program.
- Copy a file from one disk to another.
- Build a procedure to run a multi-file job.
- Call and modify the procedure built in job 5.
- Update a multi-volume master file.
- Create a multi-volume indexed file.
- Maintain a multi-volume indexed file with packed keys.
- Include utility control statements in a procedure.
- Chain procedures.

Each sample job is orgainzed into three sections:

1. An introductory summary explaining the job.
2. The OCL statements (and—where applicable—the utility control statements) for the job.
3. Explanatory notes on individual statements in the job.

The examples shown are actual computer printouts. End-of-statement keys used are shown in parenthesis to indicate actual operator response. These are shown for example only and will not be printed on normal OCL printouts.

Any response without end-of-statement key indicated is printed by the system without operator intervention.

## SAMPLE JOB 1. INITIALIZE DISK

We're going to use the Disk Initialization Program (located on the fixed disk on drive one) to initialize the removable disk on drive one. We want to:

- Initialize the entire disk pack.
- Do surface analysis only once.

The name of the new disk will be 12345.

Here are the OCL and utility control statements for the job.

```
READY-                                            LOAD  (P/S)
**************************************************************************
010 LOAD                 NAME-                    $INIT  (P/S)
011                      UNIT-                    F1  (ENTER–)
**************************************************************************
MODIFY

RUN (P/S)
   ENTER '// ' CONTROL STATEMENT
// UIN UNIT-R1,TYPE-PRIMARY  (P/S)
   ENTER '// ' CONTROL STATEMENT
// VOL PACK-12345  (P/S)
   ENTER '// ' CONTROL STATEMENT
// END  (P/S)
```

**Explanation**

| | |
|---|---|
| • 010 LOAD NAME | – $INIT<br>$INIT is the system name for the Disk Initialization Program. |
| • 011 UNIT – F1 | The Disk Initialization Program is located on the fixed disk on drive one. Pressing ENTER– instead of PROG START to end response causes DATE, SWITCH, and File keywords to be bypassed. |
| • // UIN UNIT | – R1, TYPE-PRIMARY<br>1. Tells the system to initialize the removable disk on drive one.<br>2. Because no other parameters are entered in the UIN statement, the program will:<br>• Initialize the entire pack.<br>• Read and verify the test data on the pack one time. |
| • // VOL PACK–12345 | – $INIT will enter the disk name 12345 in the VTOC. Whenever a file from this disk is used in a job, the operator must type 12345 when the system prompts PACK. |
| • // END | |

## SAMPLE JOB 2. COMPILE AN RPG SOURCE PROGRAM

We're going to use the IBM-supplied procedure RPGB (located in the source library on the fixed disk on drive one) to compile a source program INVUPD (an inventory update) located on R1. The RPG II Compiler (the program to compile RPG II source programs) is also located on R1. We want to put the compiled program in the object library on R1. Here are the OCL statements for the job.

**Explanation**

- 000 CALL NAME – RPGB
  Tells the system you want to use the IBM-supplied Compile Procedure (RPGB).
- 010 LOAD NAME – $RPG
  Tells the system you want to use the RPG II Compiler (the program to compile RPG II source programs).
- 011 UNIT – R1
  The RPG II Compiler is located on R1.
- 020 COMPILE OBJECT – F1
  The object program will be put in the object library of the disk on F1.
- 021 SOURCE – INVUPD
  The SOURCE statement in the RPGB procedure requires a delayed response. When the system reaches the SOURCE statement in the display sequence, it prompts SOURCE and waits for the operator's response.
- 022 UNIT – R1
  The response tells the system that the program to be compiled (INVUPD) is located on R1.
- 020 MODIFY – R1
  1. System prompts MODIFY.
  2. Operator types 020, telling system he wants to change that statement. (He does not want the system to put the compiled program on F1.)
  3. System tabs to position 37 and waits for response.
  4. Operator types new response – R1. The system will put the compiled program on R1.

```
READY-                            CALL  (P/S)
000 CALL          NAME-           RPGB  (P/S)
001               UNIT-           F1   (P/S)
***********************************************
010 LOAD          NAME-$RPG
011               UNIT-R1
020 COMPILE     OBJECT-F1
021             SOURCE-            INVUPD  (P/S)
022               UNIT-R1
030 FILE          NAME-$WORK
031               UNIT-F1
032               PACK-F1F1F1
033             TRACKS-20
034             RETAIN-S
040 FILE          NAME-$SOURCE
041               UNIT-F1
042               PACK-F1F1F1
043             TRACKS-20
044             RETAIN-S
***********************************************
MODIFY

020  (P/S)                        R1  (P/S)

RUN  (P/S)
```

## SAMPLE JOB 3. PROCESS CUSTOMER PROGRAM "INVUPD"

We're going to run the customer program INVUPD, compiled in SAMPLE JOB 2 and located on the removable disk on drive one. The job uses one file, INV, located on R2. The name of the disk which contains the file INV is 123456. Here are the OCL statements for the job.

```
READY-                                   LOAD  (P/S)
******************************************************************************
010 LOAD              NAME-              INVUPD (P/S)
011                   UNIT-              R1  (P/S)
020 DATE    (12/08/70)   -               (P/S)
030 SWITCH  (00000000)   -               (P/S)
040 FILE              NAME-              INV  (P/S)
041                   UNIT-              R2  (P/S)
042                   PACK-              123456 (P/S)
043                  LABEL-              (ENTER-)
050 FILE              NAME-              (P/S)
******************************************************************************
MODIFY

RUN   (P/S)
```

**Explanation**

- 020 DATE – (12/08/70)
  We'll use the current system date for the job.

- 030 SWITCH – (00000000) – (P/S)
  The program doesn't use external indicators so the operator doesn't care about the switch setting and responds by pressing the PROG START key.

- 043 LABEL – Press the ENTER– key
  Responding to LABEL by pressing the ENTER– key tells the system to bypass the rest of the file keywords and prompt FILE NAME.

- 050 FILE NAME – (P/S)
  Responding to FILE NAME by pressing PROG START causes the system to bypass the rest of the file keywords and prompt MODIFY.

## SAMPLE JOB 4. COPY FILE DISK TO DISK

We're going to copy an employee master file from R1 to R2. The second file will serve as a back-up in case the original file is damaged in some way, such as track becoming defective or a portion of the file being overlaid. When the master file was created the programmer:

1. Responded to FILE NAME with EMASTFIL.
2. Responded to PACK with VOL06.
3. Responded to LABEL with EMPMAST.
4. Responded to TRACKS with 15.

These responses caused the system to put the name EMPMAST in the VTOC on VOL06.

Here are the OCL and utility control statements we will use to copy the master file from R1 to R2.

```
READY-                                                     LOAD   (P/S)
*************************************************************************************************
010 LOAD                        NAME-                      $COPY   (P/S)
011                             UNIT-                      F1   (P/S)
020 DATE    (12/08/70)    -                                (P/S)
030 SWITCH  (00000000)    -                                (P/S)
040 FILE                        NAME-                      COPYIN   (P/S)
041                             UNIT-                      R1  (P/S)
042                             PACK-                      VOL06  (P/S)
043                            LABEL-                      EMPMAST  (ENTER-)
050 FILE                        NAME-                      COPYO  (P/S)
051                             UNIT-                      R2  (P/S)
052                             PACK-                      VOL07  (P/S)
053                            LABEL-                      EMPMAST2   (P/S)
054                          RECORDS-                      (P/S)
055                           TRACKS-                      15   (P/S)
056                         LOCATION-                      (P/S)
057                           RETAIN-                      P  (ENTER-)
060 FILE                        NAME-
*************************************************************************************************
MODIFY

RUN   (P/S)
      ENTER '// ' CONTROL STATEMENT
// COPYFILE OUTPUT-DISK  (P/S)
      ENTER '// ' CONTROL STATEMENT
// END   (P/S)
```

**Explanation**

| | |
|---|---|
| • 010 LOAD NAME | – $COPY<br>$COPY is the system name for the Disk Copy/Dump Program. |
| • 011 UNIT | – F1<br>The Copy Disk Program is on F1. |
| • 020 DATE | – (12/08/70)<br>We'll use the current system date for the job. |
| • 030 SWITCH | – (00000000)<br>This program doesn't use external indicators, so operator doesn't care about the switch setting and responds by pressing PROG START. |
| • 040 FILE NAME | – COPYIN<br>COPYIN is the predefined file name you must use for the input file whenever you use Disk Copy/Dump Program. |
| • 043 LABEL | – EMPMAST<br>EMPMAST is the VTOC file name for the OCPYIN file. You must supply this name so the system knows which file to use for COPYIN. Pressing the ENTER– key causes the system to bypass the rest of the file keywords and prompt FILE NAME. |
| • 050 FILE NAME | – COPYO<br>COPYO is the predefined file name you must use for the output file whenever you use the Disk Copy/Dump Program. |
| • 053 LABEL | – EMPMAST2<br>The system enters EMPMAST2 in the VTOC on VOL07. EMPMAST2 is the name by which the system will identify the back-up file. |
| • 055 TRACKS | – 15<br>Because we are creating a new file we must respond to one of the space keywords (TRACKS and RECORDS). We specify 15 tracks because that's what we specified for the original file. |
| • 057 RETAIN | – P<br>The back-up file is to be permanent to protect it against inadvertent overlaying. Pressing the ENTER– key causes the system to bypass the rest of the file keywords and prompt FILE NAME. |
| • COPYFILE OUTPUT | – DISK<br>The COPYFILE statement tells the program to copy the designated file from R1 to R2. |

## SAMPLE JOB 5. MULTI-FILE BUILD

Each day the customer runs a daily transaction job which creates a daily transaction file. Each day's file has a different name and date. We are going to build a procedure to use these daily files to create a weekly transaction file (WKLYTR). The weekly transaction program is located in the object library of fixed disk 1.

```
READY-                                       BUILD  (P/S)
000 BUILD               NAME-                WTR  (P/S)
001                     UNIT-                R2  (P/S)
*************************************************************************************
010 LOAD                NAME-                WKYRUN  (P/S)
011                     UNIT-                F1  (P/S)
020 DATE                    -                (P/S)
030 SWITCH (00000000)       -                11111XXX  (P/S)
040 FILE                NAME-                MONTR             MONDAYS FILE  (P/S)
041                     UNIT-                F1  (P/S)
042                     PACK-                PACK08  (P/S)
043                    LABEL-                (P/S)
044                  RECORDS-                (P/S)
045                   TRACKS-                (P/S)
046                 LOCATION-                (P/S)
047                   RETAIN-                (P/S)
048                     DATE-                ?  (P/S)
050 FILE                NAME-                TUETR             TUESDAYS FILE  (P/S)
051                     UNIT-                F1  (P/S)
052                     PACK-                PACK08  (P/S)
053                    LABEL-                (P/S)
054                  RECORDS-                (P/S)
055                   TRACKS-                (P/S)
056                 LOCATION-                (P/S)
057                   RETAIN-                (P/S)
058                     DATE-                ?  (P/S)
060 FILE                NAME-                WEDTR             WEDNESDAYS FILE  (P/S)
061                     UNIT-                F1  (P/S)
062                     PACK-                PACK08  (P/S)
063                    LABEL-                (P/S)
064                  RECORDS-                (P/S)
065                   TRACKS-                (P/S)
066                 LOCATION-                (P/S)
067                   RETAIN-                (P/S)
068                     DATE-                ?  (P/S)
070 FILE                NAME-                THUTR             THURSDAYS FILE  (P/S)
071                     UNIT-                F1 (P/S)
072                     PACK-                PACK08 (P/S)
073                    LABEL-                (P/S)
074                  RECORDS-                (P/S)
075                   TRACKS-                (P/S)
076                 LOCATION-                (P/S)
077                   RETAIN-                (P/S)
078                     DATE-                ?  (P/S)
080 FILE                NAME-                FRITR             FRIDAYS FILE  (P/S)
081                     UNIT-                F1  (P/S)
082                     PACK-                PACK08  (P/S)
083                    LABEL-                (P/S)
084                  RECORDS-                (P/S)
```

```
085              TRACKS-         (P/S)
086            LOCATION-         (P/S)
087              RETAIN-         (P/S)
088                DATE-         ?  (P/S)
090 FILE           NAME-         WKLYTR   (P/S)
091                UNIT-         R1  (P/S)
092                PACK-         PACK04   (P/S)
093               LABEL-         (P/S)
094             RECORDS-         500   (P/S)
095            LOCATION-         (P/S)
096              RETAIN-         P  (ENTER-)
100 FILE           NAME-         (P/S)
****************************************************************
MODIFY

RUN  (P/S)
```

**Explanation**

| | |
|---|---|
| • 000 BUILD NAME | – WTR<br>The procedure name in the source library is WTR. |
| • 001 UNIT | – R2<br>The procedure is located on unit R2. |
| • 020 DATE | – (P/S)<br>The date statement is not part of the procedure. |
| • 030 SWITCH (00000000) | – 11111XXX (P/S)<br>The first five external indicators are used to tell the program which input files are to be used (Monday – Friday). |
| • 040 FILE NAME | – MONTR MONDAYS FILE<br>The file name for each day is different. The comment (MONDAYS FILE) will become part of the procedure. |
| • 048 DATE | – ? (P/S)<br>The date each file was created is supplied at CALL time, when the job is run. |
| • 090 FILE NAME | – WKLYTR (P/S)<br>The output file is called WKLYTR and put on PACK04 on unit R1. |
| • 094 RECORDS | – 500 (P/S)<br>Our output file contains up to 500 records. |
| • 096 RETAIN | – P (ENTER–)<br>We want to make this a permanent file. The ENTER– key caused DATE to be skipped and FILE NAME prompted. |
| • 100 FILE NAME | – (P/S)<br>We are finished with file statements, prompt MODIFY. |
| • RUN | – Put the procedure in the source library. |

## SAMPLE JOB 6. MULTI-FILE CALL

We are going to run the procedure we built in sample job 5. However, this week Thursday was a holiday so there are only four input files. We can still use the same procedure if we delete an input file at MODIFY time.

```
READY-                                        CALL  (P/S)
000 CALL                NAME-                 WTR  (P/S)
001                     UNIT-                 F1  (P/S)
*******************************************************************
010 LOAD                NAME-WKYRUN
011                     UNIT-F1
020 SWITCH                 -11111XXX
030 FILE                NAME-MONTR
031                     UNIT-F1
032                     PACK-PACK08
033                     DATE-                 4/5/71   (P/S)
040 FILE                NAME-TUETR
041                     UNIT-F1
042                     PACK-PACK08
043                     DATE-                 4/6/71   (P/S)
050 FILE                NAME-WEDTR
051                     UNIT-F1
052                     PACK-PACK08
053                     DATE-                 4/7/71   (P/S)
060 FILE                NAME-THUTR
061                     UNIT-F1
062                     PACK-PACK08
063                     DATE-                 4/8/71   (P/S)
070 FILE                NAME-FRITR
071                     UNIT-F1
072                     PACK-PACK08
073                     DATE-                 4/9/71   (P/S)
080 FILE                NAME-WKLYTR
081                     UNIT-R1
082                     PACK-PACK04
083                   RECORDS-500
084                    RETAIN-P
*******************************************************************
MODIFY

020  (P/S)                                    11101XXX  (P/S)

060,  (P/S)

*     THURSDAYS FILE NOT USED BECAUSE OF HOLIDAY, NO RUN THAT DAY  (P/S)

RUN  (P/S))
```

**Explanation**

| | |
|---|---|
| ● 633 DATE | – 4/5/71 |
| ● 043 DATE | – 4/6/71 |
| ● 053 DATE | – 4/7/71 |
| ● 063 DATE | – 4/8/71 |
| ● 073 DATE | – 4/9/71<br>We must supply the date for each day's input file because we gave a delayed response (?) at BUILD time. Thursday's date is entered even though we will delete the file later. A date should be entered to continue the cycle. |
| ● MODIFY 020 | – We set off switch four to indicate Thursday's file is missing. |
| ● MODIFY 060 | – We delete the entire file for Thursday and enter a comment to explain why. |
| ● RUN | – Start the job. |

## SAMPLE JOB 7. UPDATE MULTI-VOLUME MASTER FILE

Every Monday the XYZ Novdlty Company prepares customer invoices, updates their customer master file, and updates their inventory file. Because the company has a huge customer file they've had to put the file on two disks: customer names beginning with A-L on one disk and the remaining customer names on a second disk. When he created this multi-volume master file, XYZ's programmer assigned the following identifying information:

1. A-L customer names:
   FILE NAME – CMASTER
   PACK – VOL01

2. M-Z customer names:
   FILE NAME – CMASTER
   PACK – VOL02

Because the company often needs information on individual customers, the programmer designed the customer master file as a direct file. The program to update the customer master file is CMUPDA. Here are the OCL statements for the job.

```
READY-                                 LOAD  (P/S)
************************************************
010 LOAD                NAME-          CMUPDA  (P/S)
011                     UNIT-          F1  (P/S)
020 DATE    (12/08/70)     -           (P/S)
030 SWITCH  (00000000)     -           (P/S)
040 FILE                NAME-          CMASTER  (P/S)
041                     UNIT-          'F1  (P/S)
                           -           R1'  (P/S)
042                     PACK-          'VOL01  (P/S)
                           -           VOL02'  (P/S)
043                    LABEL-          (ENTER-)
050 FILE                NAME-          (P/S)
************************************************
MODIFY

RUN  (P/S)
```

**Explanation**

- 041 UNIT – 'F1
  R1'
  The single quotation marks tell the system the file CMASTER is a multi-volume file. F1, R1 tells the system the file is split between the fixed and removable disks on drive one.

- 042 PACK – 'VOL01
  VOL02'
  The single quotation marks tell the system the file is on more than one disk pack. VOL01, VOL02 tells the system the name of the disk packs containing the file. Pressing the ENTER– key causes the system to bypass the rest of the file keywords and prompt FILE NAME.

- 050 FILE NAME – Pressing the PROG START key causes the system to bypass all the file keywords and prompt MODIFY.

## SAMPLE JOB 8. CREATE A MULTI-VOLUME INDEXED FILE

We are creating an inventory file. The file is very large and requires five packs. It is an indexed file with a 15 position keyfield; the keyfield consists of part number and warehouse location. The file is divided among the five volumes as follows:

| | | | | |
|---|---|---|---|---|
| Volume I01 | Keyfields | 000-000-000W1B1 | to | 175-200-233W1B2 |
| I02 | | 175-200-233W1B3 | to | 380-456-280W3R6 |
| I03 | | 380-456-287W7B3 | to | 629-384-300W3F6 |
| I04 | | 629-384-301W7B6 | to | 949-475-849W8F8 |
| I05 | | 949-476-836W4F8 | to | 999-999-999W9F9 |

The processing starts with I01 on unit R1 and I02 on unit R2. After processing I01, the program processes I02 allowing the operator to remove I01 and mount I03 on unit R1. Likewise, I04 replaces I02 and I05 replaces I03.

```
READY-                                       LOAD (P/S)
****************************************************************************
010 LOAD              NAME-                  CRTINV  (P/S)
011                   UNIT-                  F1  (P/S)
020 DATE   (12/31/23)  -                     (P/S)
030 SWITCH (00000000)  -                     (P/S)
040 FILE              NAME-                  INVMSTR  (ENTER—)
               KEY LENGTH-                   15  (P/S)
04A                  HIKEY-                  '175-200-233W1B2  (P/S)
04B                  HIKEY-                  380-456-280W3R6  (P/S)
04C                  HIKEY-                  629-384-300W3F6  (P/S)
04D                  HIKEY-                  949-475-849W8F8  (P/S)
04E                  HIKEY-                  999-999-999W9F9'  (P/S)
041                   UNIT-                  'R1  (P/S)
                         -                   R2'  (P/S)
042                   PACK-                  'VOLI01  (P/S)
                         -                   VOLI02  (P/S)
                         -                   VOLI03  (P/S)
                         -                   VOLI04  (P/S)
                         -                   VOLI05'  (P/S)
043                  LABEL-                  (P/S)
044                RECORDS-                  (P/S)
045                 TRACKS-                  '100  (P/S)
                         -                   193  (P/S)
                         -                   150  (P/S)
                         -                   193  (P/S)
                         -                   80'  (P/S)
046               LOCATION-                  '87  (P/S)
                         -                   8  (P/S)
                         -                   49  (P/S)
                         -                   8  (P/S)
                         -                   8'  (P/S)
047                 RETAIN-                  P  (ENTER—)
050 FILE              NAME-                  (P/S)
****************************************************************************
MODIFY

RUN (P/S)
```

**Explanation**

| | |
|---|---|
| • KEY LENGTH: | All characters except commas are allowed as part of the HIKEY. If apostrophes are used as part of the key, two apostrophes must be entered for each one in the key. The number of characters entered for HIKEYs must equal KEY LENGTH.<br><br>No statement number is assigned KEY LENGTH. This keyword cannot be changed at MODIFY time. |
| • 045 TRACKS<br>046 LOCATION | The file need not occupy the entire volume if the number of tracks and the starting location are given. You must be sure these areas are available because the system cannot check offline packs. |

## SAMPLE JOB 9. MAINTAIN A MULTI-VOLUME INDEXED FILE WITH PACKED KEYS

We are maintaining a multi-volume indexed file. The file occupies four volumes. The keyfield is 15 characters long in packed format. The keyfield takes eight bytes in the record. The file is divided as follows:

| | | |
|---|---|---|
| Volume P01 | Keyfields | 000 000 000 000 000 through 000 025 000 000 000 |
| P02 | | 000 025 000 000 001 through 000 050 000 000 000 |
| P03 | | 000 050 000 000 001 through 000 075 000 000 000 |
| P04 | | 000 075 000 000 001 through 000 100 000 000 000 |

The OCL required to use this file is as follows:

```
READY-                                      LOAD (P/S)
********************************************************************************
010 LOAD                NAME-               PAYROL (P/S)
011                     UNIT-               F1 (P/S)
020 DATE    (07/09/71)  -                   (P/S)
030 SWITCH (00000000)   -                   (P/S)
040 FILE                NAME-               PAYROLL (ENTER-)
                  KEY LENGTH-               08,P (P/S)
04A                    HIKEY-               '000025000000000 (P/S)
04B                    HIKEY-               000050000000000 (P/S)
04C                    HIKEY-               000075000000000 (P/S)
04D                    HIKEY-               000100000000000' (P/S)
041                     UNIT-               'R1 (P/S)
                            -               R2' (P/S)
042                     PACK-               'VOLP01 (P/S)
                            -               VOLP02 (P/S)
                            -               VOLP03 (P/S)
                            -               VOLP04' (P/S)
043                    LABEL-               ACCONT (ENTER-)
050 FILE                NAME-               (P/S)
********************************************************************************
MODIFY

RUN (P/S)
```

## SAMPLE JOB 10. INCLUDE UTILITY CONTROL STATEMENTS IN A PROCEDURE

Sample job 1 showed an OCL LOAD cycle for initializing the removable disk on drive one. This sample job shows how to do the same job using BUILD and CALL cycles and including the Utility Control Statements in the procedure.

```
READY-                                          BUILD (P/S)
000 BUILD                NAME-                  INITRI  (P/S)
001                      UNIT-                  F1  (P/S)
**********************************************************************
010 LOAD                 NAME-                  $INIT  (P/S)
011                      UNIT-                  F1  (P/S)
020 DATE                     -                  (P/S)
030 SWITCH (00000000)        -                  (P/S)
040 FILE                 NAME-                  (P/S)
**********************************************************************
MODIFY

INCLUDE  (P/S)
**********************************************************************
ENTER UTILITY CONTROL STATEMENTS
00

// UIN UNIT-R1,TYPE-PRIMARY  (P/S)
01

// VOL PACK-12345  (P/S)
02

// END  (P/S)
03

RUN  (P/S)
**********************************************************************
MODIFY
READY-                                          CALL  (P/S)
000 CALL                 NAME-                  INITRI  (P/S)
001                      UNIT-                  F1  (P/S)
**********************************************************************
010 LOAD                 NAME-$INIT
011                      UNIT-F1
**********************************************************************
MODIFY

RUN  (P/S)
**********************************************************************
INCLUDED STATEMENTS
00 // UIN UNIT-R1,TYPE-PRIMARY
01 // VOL PACK-12345
02 // END
**********************************************************************
MODIFY

RUN  (P/S)
```

## SAMPLE JOB 11. CHAIN PROCEDURES

We're going to use the BUILDC cycle to chain two procedures created with the BUILD cycle. First, we use the BUILD cycle to build procedures to use the Conversational Utilities ($KSE and $KDE).

After the chained procedure is built, the CALL cycle is used to run the chained procedures.

```
READY-                                    BUILD (P/S)
000 BUILD              NAME-              KSE (P/S)
001                    UNIT-              F1 (P/S)
*****************************************************************
010 LOAD               NAME-              $KSE (P/S)
011                    UNIT-              F1 (P/S)
020 DATE                   -              (P/S)
030 SWITCH (00000000)      -              (P/S)
040 FILE               NAME-              (P/S)
*****************************************************************
MODIFY

RUN (P/S)


READY-                                    BUILD (P/S)
000 BUILD              NAME-              KDE (P/S)
001                    UNIT-              F1 (P/S)
*****************************************************************
010 LOAD               NAME-              $KDE (P/S)
011                    UNIT-              F1 (P/S)
020 DATE                   -              (P/S)
030 SWITCH (00000000)      -              (P/S)
040 FILE               NAME-              KDEFILE (P/S)
041                    UNIT-              F1 (P/S)
042                    PACK-              F1F1F1 (P/S)
043                   LABEL-              DRIV2 (P/S)
044                 RECORDS-              4 (P/S)
045                LOCATION-              (P/S)
046                  RETAIN-              T (P/S)
047                    DATE-              (P/S)
050 FILE               NAME-              (P/S)
*****************************************************************
MODIFY

RUN (P/S)


READY-                                    BUILDC (P/S)
000 BUILDC             NAME-              MASTER (P/S)
001                    UNIT-              F1 (P/S)
*****************************************************************
010 CALL               NAME-              KSE (P/S)
011                    UNIT-              F1 (P/S)
020 CALL               NAME-              KDE (P/S)_
021                    UNIT-              F1 (ENTER-)
*****************************************************************
MODIFY

RUN (P/S)



READY-                                    CALL (P/S)
000 CALL               NAME-              MASTER (P/S)
001                    UNIT-              F1 (P/S)
000 CALL               NAME-KSE
001                    UNIT-F1
*****************************************************************
010 LOAD               NAME-$KSE
011                    UNIT-F1
*****************************************************************

FORMAT DESCRIPTION ?                 YES (P/S)

FORMAT TYPE -                        KDE (P/S)
```

```
NEW SOURCE MODULE ?                 YES (P/S)

SOURCE MODULE NAME -                KDEFOR (P/S)

SOURCE MODULE UNIT -                F1 (P/S)

06672 NEW STATEMENTS MAY BE ADDED TO SOURCE ENTRY

00000      H01                      096 (P/S)

00010      A005 (P/S)

00020      A091 (P/S)

00030      H02                      (COMMAND KEY 06 PRESSED)

END OF JOB ?                        YES (P/S)

KSE END OF JOB



000 CALL              NAME-KDE
001                   UNIT-F1
****************************************************************
010 LOAD              NAME-$KDE
011                   UNIT-F1
020 FILE              NAME-KDEFILE
021                   UNIT-F1
022                   PACK-F1F1F1
023                  LABEL-DRIV2
024                RECORDS-4
025                 RETAIN-T
****************************************************************
FORMAT NAME -                       KDEFOR (P/S)

FORMAT UNIT -                       F1 (P/S)

DISPLAY FORMATS ?                   YES (P/S)

H01096

A005

A091

NEW KDE FILE ?                      YES (P/S)

KEY FIELD START -                   NO (P/S)

SELECT FORMAT NUMBER -              01 (P/S)
*     *

00000 THIS IS AN EXAMPLE OF CHAIN PROCEDURE ON THE MODEL 6 (P/S)

00010 KSE WAS THE FIRST JOB EXECUTED AND KDE WAS THE SECOND AND LAST JOB (P/S)

00020 THE CHAIN WAS INITIATED BY CALLING MASTER, WHICH WAS BUILT IN A BUILDC CYCLE (P/S)

00030  (COMMAND KEY 06 PRESSED)

*********************************************************************************************

BATCH ACCUMULATORS 00                 01                 02                 03                 04
                   0                  0                  0                  0                  0
                   05                 06                 07                 08                 09
                   0                  0                  0                  0                  0

FINAL ACCUMULATORS 00                 01                 02                 03                 04
                   0                  0                  0                  0                  0
                   05                 06                 07                 08                 09
                   0                  0                  0                  0                  0

*********************************************************************************************

END OF JOB ?                        YES (P/S)

KDE END OF JOB
```

# PART II
# DISK UTILITY PROGRAMS

## Introduction to Disk Utility Programs

Every method of data processing requires a certain amount of maintenance work to keep it in good running order. For example, you must make back-up copies of important files, and remove out-of-date files. The Disk Utility programs are a collection of maintenance programs to serve your data-processing system. The Disk Utility programs are:

Disk Initialization
Alternate Track Assignment
Alternate Track Rebuild
File and Volume Label Display
File Delete
Disk/Copy Dump
Library Maintenance

You might use one of the preceding utility programs to:

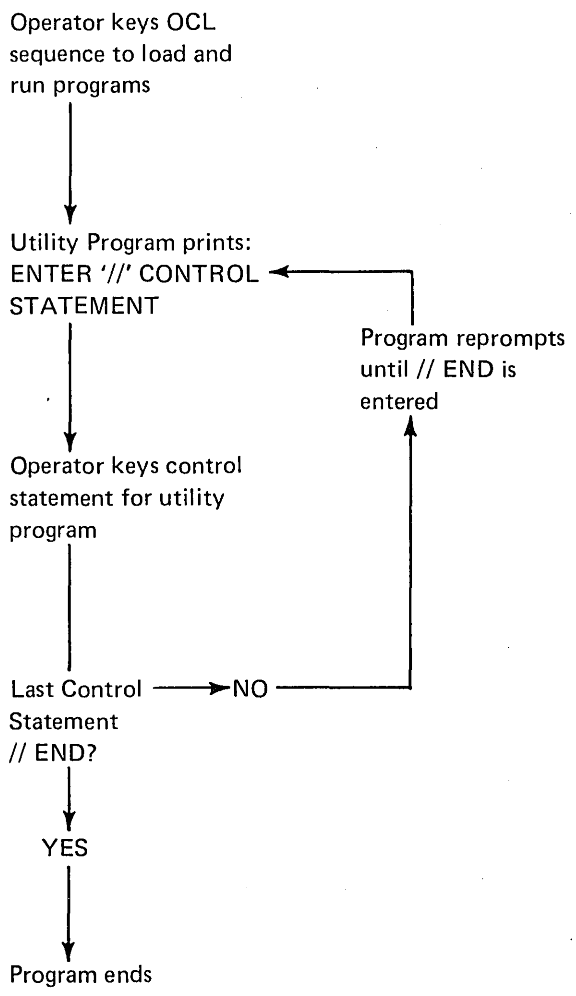
- Prepare disks for use.
- Replace defective tracks.
- Replace incorrect data on a track.
- Print VTOC (volume table of contents) information.
- Delete files from a disk.
- Copy or print files.
- Maintain system libraries.

### GENERAL PROGRAM OPERATION

The utility programs require control statements describing the jobs you want done. They read these statements from the system input device, or from procedures stored in a source library on disk. The system input device is normally the keyboard, but the operator can specify another device by his response to the OCL keyword READER during initial program loading (IPL).

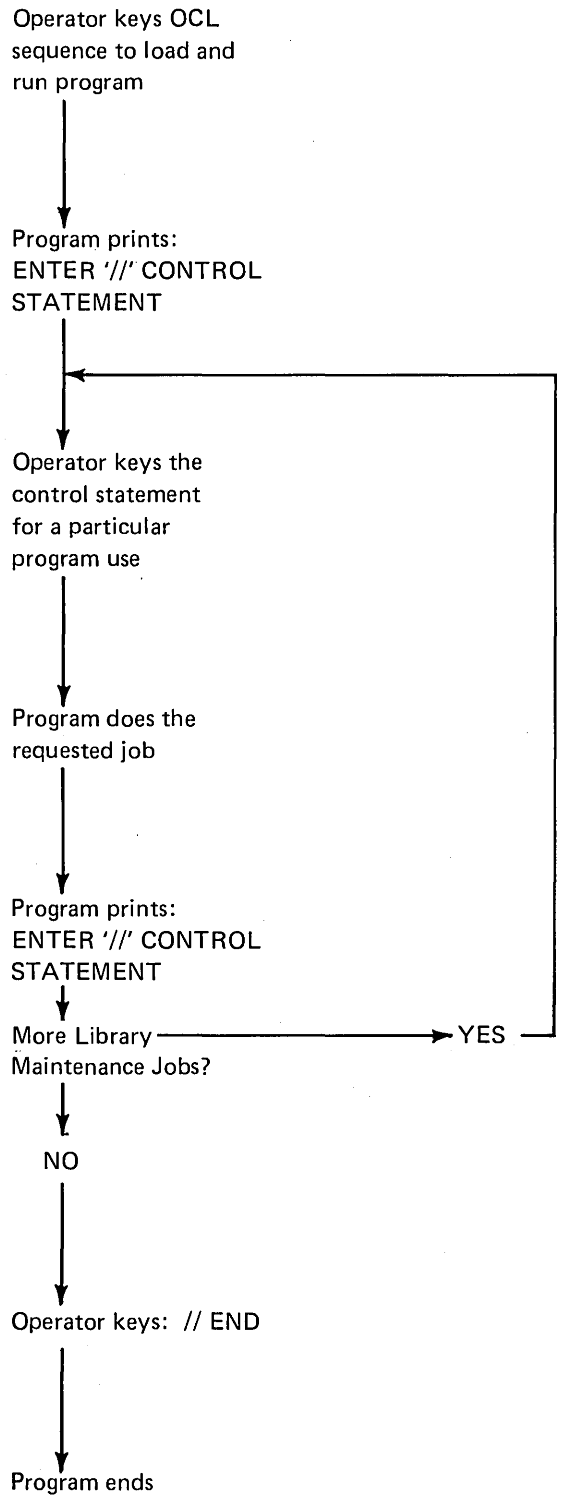
The following diagrams outline the general way the utility programs operate. Assume that the programs are reading control statements from the keyboard.

**All Programs Except Library Maintenance**



Operator keys OCL sequence to load and run programs
Utility Program prints: ENTER '//' CONTROL STATEMENT
Program reprompts until // END is entered
Operator keys control statement for utility program
Last Control Statement // END?
NO
YES
Program ends

**Library Maintenance Program**


Operator keys OCL sequence to load and run program
Program prints: ENTER '//' CONTROL STATEMENT
Operator keys the control statement for a particular program use
Program does the requested job
Program prints: ENTER '//' CONTROL STATEMENT
More Library Maintenance Jobs?
YES
NO
Operator keys: // END
Program ends

## USING DISK UTILITIES

To use utility programs, you must write utility control statements and operation control language (OCL) statements. In this manual, therefore, the information for every program is divided into five sections:

- Control statement summary
- Parameter summary
- Parameter descriptions
- OCL considerations
- Examples

The first three sections are to guide you in writing utility control statements. The OCL section is to guide you in writing OCL statements. The examples will help you in both.

### Control Statements

Every control statement is made up of an identifier and parameters. The identifier is a word that identifies the control statement. It is always the first word of the statement (following // blank in positions 1-3). Parameters are information you are supplying to the program. Every parameter consists of a keyword, which identifies the parameter, followed by the information you are supplying.

You may write utility control statements on whatever paper or preprinted forms you like. In writing the statements, use the manual in the following way:

1. Look at the CONTROL STATEMENT SUMMARY to determine which control statements and parameters apply to the program use you are interested in. (The program uses are stated in the text preceding the control statement summary.)

2. If you need information about the contents or meanings of particular parameters, look at the PARAMETER SUMMARY.

3. If you need more detailed information about parameters, read the PARAMETER DESCRIPTIONS following the parameter summary.

4. If you need examples of specific jobs, look at the EXAMPLE section. All examples show the OCL and utility control statements needed to load and run the utility programs for specific jobs. The statements are shown in the form they are printed on the system printer.

*Coding Rules*

The rules for writing control statements are as follows:

1. *//blank.* All control statements must have // blank in positions 1-3.

2. *Statement Identifier.* Begin in it position 4 or after of the statement. Do not use blanks within the identifier.

3. *Blanks.* Use one or more blanks between the identifier and the first parameter. Do not use them anywhere else in the statement.

4. *Statement parameters.* Parameters can be in any order. Use a comma to separate one parameter from another. Use a hyphen (-) within each parameter to separate the keyword from the information you supply. Do not use blanks within or between parameters.

5. *Statement parameters containing a list of data after the keyword.* Use apostrophes (') to enclose the items in the list. Use a comma to separate one item from another. For example: UNIT-'R1,R2' (R1 and R2 are the items in the list).

6. *Statement length.* Control statements must not exceed 96 characters.

The following example shows a control statement. The statement identifier is COPY. The parameter keywords are FROM, LIBRARY, NAME, and TO. The information you supply is F1, O, SYSTEM, and R1.

```
// COPY FROM-F1,LIBRARY-O,NAME-SYSTEM,TO-R1
```

*End-Control Statement*

The END statement is a special control statement that indicates the end of control statements. It consists of the letters // END in positions 1-6 and must always be the last control statement for the programs.

## WRITING OCL STATEMENTS

To write OCL statements to run a utility program, look at the OCL CONSIDERATIONS section for that program. There you will find a list of the required keywords and responses for LOAD and BUILD sequences. (Keywords not listed can be bypassed.) Should you need more general information about OCL, or more specific information about the keywords, see Part I of this manual.

*Note:* Capitalized words and letters, numbers, and special characters have special meanings in OCL and utility control statement descriptions in this manual.

### Utility Control Statements

In utility control statements, capitalized words and letters must be written as they appear in the statement description. Sometimes numbers appear with the capitalized information. These numbers must also be written as shown.

Words or letters that are not capitalized mean you must use a value that applies to the job you are doing. The values you can use are listed in the parameter summaries for the control statements.

Braces ( $\{\ \}$ ) sometimes appear in parameters shown in control statement summaries and parameter sumamries. They are not part of the parameters. They simply indicate that you must choose one of several values to complete the parameter. For example, RETAIN $\left\{\begin{matrix}T\\P\end{matrix}\right\}$ means you can use either RETAIN-T or RETAIN-P.

### OCL Statements

In OCL statements, keywords are capitalized. Responses that are shown in capital letters must be written as shown. If numbers or special characters are included with the capital letters, they must be written as part of the response. For example, $INIT is the name of the Disk Initialization program and must be written exactly as shown. Responses that are not capitalized mean you must use the value that applies to the job you are doing.

# Disk Initialization Program ($INIT)

Disks that are being used for the first time must be prepared for use. This process is called *initialization.* You can also use a disk that has been used before by reinitializing that disk (any data on the disk is destroyed). You use the Disk Initialization program to perform initialization.

Track and sector addresses are not written on disks when the disks are manufactured. You must do this before you use the disks. The Disk Initialization program does it for you.

## FUNCTIONS

Initializing a disk involves:

- Naming the disk.
- Writing track and sector addresses on the disk.
- Checking for defective tracks.
- Assigning alternate tracks to any defective tracks.

### Naming a Disk

You must name every disk you intend to use. The operator uses this name to ensure that the correct disks are being used for a job. He supplies the disk name in either OCL statements or program control statements. The system checks this name against the name stored as identification on the disk pack. If the names don't match, a halt occurs and a message is printed to the operator. The operator may then change disks. All this must happen before a Model 6 program can use a disk.

### Writing Track and Sector Addresses

A disk contains 200 or 400 tracks, each of which is divided into 24 sectors. An area at the beginning of every track and sector is set aside for an address. These addresses are necessary for locating data.

### Checking for Defective Tracks (Surface Analysis)

The Disk Initialization program checks the condition of tracks. It does this by writing data on the tracks, then reading and checking the data to ensure it was recorded properly. If the check shows that the data is incorrect, the track on which the data was written is considered defective. This process is called *surface analysis.*

### Assigning Alternate Tracks

If a defective track is found during surface analysis, an alternate track is assigned to it. The sole purpose of the alternate track is to act as a substitute for the defective track. Model 6 programs attempting to use the defective track will automatically use the alternate instead.

If either track 0 or 1 is defective, the program considers the disk unusable and stops initializing it. Tracks 0 and 1 are used only by the system and cannot have alternates assigned to them.

Every disk has six alternate tracks. Therefore, a maximum of six defective tracks may be assigned alternates on a disk. If there are more, the disk is considered unusable.

If tracks become defective after a disk is initialized, another program (Alternate Track Assignment) is used to assign alternate tracks. Disks need not be reinitialized to assign alternate tracks.

## OPTIONS

The Disk Initialization program allows you the following options:

- You may choose one of three types of initialization: primary, secondary, or clear.
- You may initialize up to three disks during the same program run.
- During primary initialization, you may decide whether to erase alternate track assignments already on the disk or leave them assigned.
- You may use up to ten characters, in addition to the disk name, to further identify a disk.
- You may specify the number of times you want the program to do surface analysis.

You specify the options you want in control statements (see *Control Statements* in this chapter).

### Type of Initialization

The program offers three types of initialization: primary, secondary, and clear. The type you choose determines the portion of the disk that will be initialized. The portions of a disk that can be initialized depend on the data-storage capacity of your disk drive.

Disk drives of differing storage capacities are available for your system. All drives use the same type of disks. The only difference is the number of tracks the drives can use. The larger the drive capacity, the more tracks the drive can use.

If you increase the capacity of your disk drives, more tracks on your disks become available for use. These additional tracks must be initialized before being used. The three types of initialization allow you the following options according to type.

- Primary or clear—initializing all tracks corresponding to the new capacity, including any that were previously initialized.
- Secondary—initializing only the additional tracks made available by the increased capacity.

#### *Primary Initialization*

Primary initialization applies to new disks, or disks you have used but want to initialize again. The program initialized all tracks corresponding to the capacity of the drives on which the disks are mounted. Tracks that were previously initialized are initialized again. Any data on the tracks is destroyed.

You can use primary initialization on a disk as often as you want. However, the program will not initialize disks containing libraries, temporary data files, or permanent data files. You must delete data files with the File Delete Program and libraries with the allocate function of the Library Maintenance Program.

#### *Secondary Initialization*

Secondary initialization applies to disks that were initialized on drives of less capacity than drives you are now using. When you increase the capacity of your drives, more tracks on your disks become available for use. You must initialize the additional tracks. Use secondary initialization if you do not want information destroyed on tracks already in use. The program initializes the additional tracks only. Tracks already in use are not disturbed.

The program will not do secondary initialization on new disks or disks that have already been initialized to the capacity of the drives on which they are mounted.

#### *Clear Initialization*

Clear initialization applies to new disks but only to those which cannot be used because of invalid pack labels or some other unrecoverable disk error. All tracks corresponding to the capacity of the drives on which the disks are mounted are initialized. Tracks that were previously initialized are reinitialized.

*Warning:* All libraries, temporary data files, or permanent data files are completely wiped out.

### Number of Disks

The Disk Initialization program can initialize a maximum of three disks during one program run. The type of initialization you specify for a program run applies to all disks being initialized during that run. The disks, however, must be mounted at the same time. You can't, for example, initialize more than one removable disk on a given drive during the same program run.

### Erasing Alternate Track Assignments

You can use primary or clear initialization to reinitialize disks that have been used. However, alternate track assignments could exist on such disks. The Disk Initialization program, therefore, gives you the option of:

- Erasing existing alternate track assignments and checking the condition of all tracks.
- Leaving existing alternate track assignments and checking only those tracks to which alternates are not assigned.

The option you choose applies to all disks being initialized during the program run.

### Additional Disk Identification

When you name a disk during primary or clear initialization, you can use up to ten characters, in addition to the disk name, to further identify the disk. The additional identification is strictly for your use. It is not used by the checking programs to ensure that the right disks are being used.

If you use the File and Volume Label Display program to print VTOC (volume table of contents) information from a disk, the additional identification is printed with the disk name.

### Surface Analysis Option

You can tell the Disk Initialization program to perform surface analysis from 1 to 255 times before judging whether or not tracks are defective. A track must successfully complete every check before being judged usable. If incorrect data is detected during surface analysis, the track on which the data was written is judged defective and an alternate is assigned to it.

The number of times you specify surface analysis to be performed applies to all disks being initialized during the program run. The time required for initialization is increased if you request surface analysis to be performed more than once.

## CONTROL STATEMENTS

You must supply the following control statements to specify the program options you want:

1. *UIN statement*—indicates the type of initialization, the number of disks being initialized, the number of times you want surface analysis performed, and whether or not you want previous alternate track assignments erased. One UIN statement is required per program run.
2. *VOL statement*—indicates the name you assign to the disk, plus any additional identification you want to give the disk. The VOL statement applies to primary and clear initialization only. One is required for every disk you initialize.
3. *END statement*—indicates the end of control statements.

## Control Statement Summary

| Type of Initialization | Control Statements ❶ |
|---|---|
| Primary ❷: | |
| New Disks | // UIN TYPE-PRIMARY,UNIT-$\begin{Bmatrix}\text{code}\\ \text{'codes'}\end{Bmatrix}$,VERIFY-number,CAP-$\begin{Bmatrix}\text{HALF}\\ \text{FULL}\end{Bmatrix}$<br>// VOL PACK-name,ID-characters<br>// END |
| Disk already in use (reinitialize) | // UIN TYPE-PRIMARY,UNIT-$\begin{Bmatrix}\text{code}\\ \text{'codes'}\end{Bmatrix}$,VERIFY-number,ERASE-$\begin{Bmatrix}\text{NO}\\ \text{YES}\end{Bmatrix}$,CAP-$\begin{Bmatrix}\text{HALF}\\ \text{FULL}\end{Bmatrix}$<br>// VOL PACK-name,ID-characters<br>// END |
| Secondary ❸: | |
| Disk already in use | // UIN TYPE-SECONDARY,UNIT-$\begin{Bmatrix}\text{code}\\ \text{'codes'}\end{Bmatrix}$,VERIFY-number<br>// END |
| Clear | // UIN TYPE-CLEAR,UNIT-$\begin{Bmatrix}\text{code}\\ \text{'codes'}\end{Bmatrix}$,VERIFY-number,CAP-$\begin{Bmatrix}\text{HALF}\\ \text{FULL}\end{Bmatrix}$<br>// VOL PACK-name,ID-characters<br>// END |

❶ Control statements are required in the order they are listed: UIN, VOL, END or UIN, END.

❷ For primary initialization, one VOL statement is required for each disk listed in the UNIT parameter of the UIN statement. The PACK parameter in the first VOL statement applies to the first disk listed in the UNIT parameter. The PACK parameter in the second VOL statement applies to the second disk listed in the UNIT parameter, and so on.

❸ VOL statements are not required for secondary initialization because the disks are already named.

## Parameter Summary

| UIN (Input Definition) Statement | | |
|---|---|---|
| TYPE-PRIMARY | Primary initialization. Initialize the disks to the capacity of the drives on which they are mounted. Tracks already initialized are re-initialized. The program will not initialize disks containing libraries, temporary data files, or permanent data files. | |
| TYPE-SECONDARY | Secondary initialization. Applies only to disks that were initialized on drives of less capacity than the drives you are now are using. It means initialize the uninitialized portions of the disks to the capacity of the drives on which the disks are mounted. Tracks already initialized are not distrubed. | |
| TYPE-CLEAR | Clear initailization. Initialize the disks to the capacity of the drives on which they are mounted. Tracks already initialized are re-initialized. Active files and library checking is bypassed and any data on the tracks is destroyed. Error logging areas on F1 are saved. | |
| UNIT-code | Disk location (one disk). | Possible codes: R1, F1, R2, F2 |
| UNIT-'code,code' | Disk location (two disks). | |
| UNIT-'code,code,code' | Disk location (three disks). | |
| VERIFY-number | Do surface analysis the number of times indicated (number can be 1-255). VERIFY-1 is assumed if you omit the parameter. | |
| ERASE-YES | Retest defective tracks. | Primary initialization only. ERASE-NO is assumed if you omit the parameter. |
| ERASE-NO | Do not retest defective tracks. | |
| CAP-HALF | Initialize a disk to half capacity even if on a full capacity drive. | The CAP Keyword forces ERASE-YES. Pack is initialized to capacity of the drive if this keyword is omitted. |
| CAP-FULL | Initialize a disk to full capacity. | |

| VOL (Volume) Statement | |
|---|---|
| PACK-name | Disk name. Can contain any of the standard System/3 characters except apostrophes, leading or embedded blanks, and embedded commas. Its length must not exceed six characters. |
| ID-characters | Additional identification. Can contain any of the standard System/3 characters except apostrophes, leading or embedded blanks, and embedded commas. Its length must not exceed ten characters. If you omit this parameter no additional identification is written on the disk. |

## PARAMETER DESCRIPTIONS

### TYPE Parameter (UIN)

The TYPE parameter indicates the type of initialization you want the program to do: primary, secondary, or clear. The type of initialization and the capacity of the disk drives on which the disks are mounted determine which disk tracks will be initialized. If this parameter is omitted, primary is assumed.

### UNIT Parameter (UIN)

The UNIT parameter (UNIT-code) tells the location of the disks you want to initialize. The program can initialize up to three disks during one program run.

The form of the UNIT parameter depends on the number of disks you are initializing:

1. For one disk, use UNIT-code
2. For two disks, use UNIT-'code,code'
3. For three disks, use UNIT-'code,code,code'

The codes indicate the locations of the disks:

| Code | Location |
|---|---|
| R1 | Removable disk on drive 1. |
| F1 | Fixed disk on drive 1. |
| R2 | Removable disk on drive 2. |
| F2 | Fixed disk on drive 2. |

For primary and clear initialization, the order of codes must correspond to the order of VOL control statements. If, for example, you had used the parameter UNIT-'R1,R2', the first VOL statement applies to the removable disk on drive 1 and the second VOL statement to the removable disk on drive 2. (No VOL statements are required for secondary initialization. The disk is already named.)

### VERIFY Parameter (UIN)

The VERIFY parameter (VERIFY-number) concerns surface analysis. It enables you to indicate the number of times you want the program to do surface analysis before judging whether or not tracks are defective. The number can be from 1-255. If this parameter is omitted, VERIFY-1 is assumed.

### ERASE Parameter (UIN)

The ERASE parameter concerns alternate track assignment. It applies only to disks that have already been initialized and used, but you are reinitializing using primary initialization.

The condition of tracks on such disks has been tested at least once before (during the previous initialization) and tracks that were found to be defective during surface analysis were assigned alternates. The ERASE parameter, therefore, enables you to indicate whether you want the program to (1) retest the tracks to which alternate tracks are already assigned or (2) leave the alternate tracks assigned without retesting the tracks.

The parameter ERASE-YES means to retest. If you tell the program to retest, it erases any existing alternate track assignments, and tests all tracks as though the disk were new.

The parameter ERASE-NO means not to retest. If you tell the program not to retest, it tests only those tracks to which no alternate tracks are assigned. Alternate tracks previously assigned remain assigned.

### CAP Parameter

The CAP parameter determines the size of the pack when it is initialized. The CAP-HALF parameter means to initialize the pack to half capacity even if it is on a full capacity drive. The CAP-FULL parameter means to initialize the pack to full capacity. CAP-FULL should not be used on a half capacity system. The use of the CAP keyword forces ERASE-YES.

#### *Disk Drive Capacity*

Disk Drives of different data-storage capacities are available for System/3 Model 6. All drives use the same type of disks. The only difference is the number of tracks the drives can use: the larger the drive capacity, the more tracks the drive can use. However, you must initialize the disk tracks before using them.

**PACK Parameter (VOL)**

The PACK parameter (PACK-name) applies to primary and clear initialization only. During primary and clear initialization, the Disk Initialization program writes a name on each disk. It uses the name you supply in the corresponding PACK parameter. (One VOL control statement containing a PACK parameter is required for each disk.)

The name can be any combination of standard System/3 characters except apostrophes (') and leading or embedded blanks (see Appendix J). Its length must not exceed six characters. The following are valid disk names: 0, F0001, 012, A1B9, ABC.

In general, disk names are used for checking purposes. Before a program uses a disk, the disk name is compared with a name you supply (either in OCL statements or control statements required by the program). If the names do not match, a message to the operator is printed. In this way, programs cannot use the wrong disks without the operator knowing about it.

**ID (Identification) Parameter (VOL)**

The ID parameter (ID-characters) applies to primary and clear initialization only. It enables you to include up to ten characters, in addition to the disk name, to further identify a disk. The information is strictly for your use. (It is not used for checking purposes by the system.) If you use the File and Volume Label Display program to print the disk name, it will also print the additional identification for you.

The additional identification can be any combination of standard System/3 characters except apostrophes (') and leading or embedded blanks. However, the maximum number is ten.

## OCL CONSIDERATIONS

### LOAD Sequence

| Keywords [1] | Responses [2] | Considerations |
|---|---|---|
| READY | LOAD | None |
| LOAD NAME | $INIT | Name of Disk Initialization program. |
| UNIT | R1, R2, F1, or F2 | Location of disk containing Disk Initialization program. |
| MODIFY | RUN | None |

[1] Only the keywords listed here are required. You can bypass the rest.

[2] You end every response by pressing PROG START.

### BUILD Sequence

| Keywords [1] | Responses [2] | Considerations |
|---|---|---|
| READY | BUILD | None |
| BUILD NAME | Procedure name | Name by which procedure will be identified in source library. |
| UNIT | R1, R2, F1, or F2 | Location of disk containing source library. |
| LOAD NAME | $INIT | Name of Disk Initialization program. |
| UNIT | R1, R2, F1, or F2 | Location of disk containing Disk Initialization program. |
| MODIFY | INCLUDE<br>utility control statements<br>RUN | Response when including control statements in procedure. |
|  | OR RUN | Response when not including control statements in procedure. |

[1] Only the keywords listed here are required. You can bypass the rest.

[2] You end every response by pressing PROG START.

## EXAMPLE

### Primary Initialization of Two Disks

*Statements*

```
READY                            -  LOAD
************************
010     LOAD    NAME             -  $INIT
011             UNIT             -  F1
020     DATE    (XX/XX/XX)       -
030     SWITCH  (00000000)       -
040     FILE    NAME             -
************************
MODIFY
RUN
```

**OCL LOAD Sequence.**

Boxed areas are operator responses.

Keywords for which no responses are shown are the ones bypassed. If you press ENTER— after responding to UNIT, the DATE, SWITCH, and FILE NAME keywords are not prompted.

RUN is the response to MODIFY even though the two words do not appear on the same line.

```
   ENTER '//' CONTROL STATEMENT
```

Message printed by Disk Initialization program.

```
// UIN UNIT-'F2,R2',TYPE-PRIMARY
```

Control statement supplied by operator.

```
   ENTER '//' CONTROL STATEMENT
// VOL PACK-2222
   ENTER '//' CONTROL STATEMENT
// VOL PACK-PAYROL,ID-010270
   ENTER '//' CONTROL STATEMENT
// END
```

Sequence repeats until operator enters END statement.

*Explanation*

- Disk Initialization program is loaded from the fixed disk on drive 1 (UNIT-F1 in OCL sequence).
- The two disks on drive 2 are being initialized (UNIT-'F2, R2' in UIN statement.
- The fixed disk (F2) will be given the name 2222 (PACK-2222 in first VOL statement).
- The removable disk (R2) will be given the name PAYROL (PACK-PAYROL in second VOL statement). Additional identifying information, 010270, will be written on the removable disk (ID-010270).

## MESSAGES FOR DISK INITIALIZATION

| Message | Meaning |
|---|---|
| INITIALIZATION ON XX COMPLETE | This message is printed when initialization of a disk is complete. XX indicates the unit (R1, R2, F1, or F2) on which the initialization is complete. |
| INITIALIZATION ON XX TERMINATED | This message is printed when initialization of a disk must be terminated for one of the following reasons:<br>1. Cylinder zero is defective.<br>2. More than six tracks are defective.<br>3. Possible disk hardware error exists.<br>4. The program attempted to initialize the disk ten times without success.<br>After this message is printed, halt A13 will occur. XX indicates the unit (R1, R2, F1, or F2) on which the initialization is terminated. |
| **ALTERNATE TRACKS ASSIGNED** | These two messages are printed when a primary track is defective and an alternate track is assigned to it. |
| PRIMARY TRACK XXX ALTERNATE TRACK XXX | XXX indicates the tracks involved. |
| UNRECOVERABLE ERROR; RE-INITIALIZING PACK | This message is printed when the Disk Initialization program determines that the disk has not been initialized properly. The program will again attempt to initialize the disk correctly with ERASE-YES forced. The maximum number of times that the program will attempt to initialize a disk is ten. After that number of times, halt A13 occurs. |

Sometimes a disk track causes a reading or writing error during a job and an alternate track must be assigned to replace the defective track. The process of assigning an alternate track is performed by the Alternate Track Assignment program.

## FUNCTIONS

The process of assigning an alternate track consists of:

- Writing track addresses on disk.
- Checking for defective tracks.
- Printing all track sectors that contain incorrect data.
- Assigning an alternate track.

### Writing Track Addresses

Any time a track causes reading or writing errors during a job, the system stops the program currently in operation and writes the track address in a special area on the disk. All disks contain such an area. The program can then locate a track by using the addresses stored in this area. As long as there are alternate tracks available for use, assignment can be done for all the tracks identified in this area.

### Checking For Defective Tracks

The Alternate Track Assignment program uses a procedure called *surface analysis* to test the condition of tracks. Surface analysis consists of writing test data on a track, then reading the data to ensure it was written properly.

Before doing surface analysis, the Alternate Track Assignment program transfers any data from the track to an alternate track. This is the alternate that will be assigned if the track proves to be defective.

In judging whether or not the track is defective, the program does surface analysis the number of times you specify in the VERIFY parameter. If you omit the parameter, the program does surface analysis once. If the track causes reading or writing errors any time during surface analysis, the program considers the track defective.

### Printing Sectors Containing Incorrect Data

If a track is defective, some of the data transferred to the alternate track could be incorrect. Therefore, when reading data from the defective track, the program logs all track sectors containing data that caused reading errors. For a hard-copy printout, the printer must be assigned as the logging device. Characters that have no print symbol are printed as two-digit hexadecimal numbers. The following is an example:

```
ABCDE GH123 45...
     B     A
     6     5
```

Appendix A lists the characters in the standard character set and their corresponding hexadecimal numbers.

To correct errors on the alternate track, use the Alternate Track Rebuild program.

### Assigning An Alternate Track

An alternate track is assigned if a track is defective. When the program assigns an alternate, it transfers the contents of the defective track to the alternate. The alternate track is then automatically used any time the program attempts to use the defective track.

There are six alternate tracks. The program will not do conditional assignment if all six are already in use.

## OPTIONS

The Alternate Track Assignment program gives you the following options:

- You may choose one of three types of assignment—conditional, unconditonal, or cancel prior.
- You may use up to six alternate tracks on every disk.
- You may specify the number of times you want the program to do surface analysis.

You specify the options you want in control statements (see *Control Statements* in this chapter).

### Type of Assignment

The program offers three types of assignment: conditional, unconditional, and cancel prior. The three types of assignment allow you the following options according to type.

- Conditional—testing the condition of a track and assigning an alternate if it is defective.
- Unconditional—assuming a track is defective and assigning an alternate.
- Cancel prior—canceling an alternate track assignemnt.

#### *Conditional Assignment*

Conditional Assignment consists of testing the condition of a track (surface analysis) and, if the track is defective, assigning an alternate track to replace it. It is the normal use of the Alternate Track Assignment program.

*Situation:* Conditional assignment applies to tracks that cause reading or writing errors during a job. Anytime a track causes such errors, the system does the following:

1. Stops the program currently in operation.
2. Writes the track address in a special area on the disk.

When you use the Alternate Track Assignment program to do conditional assignment, the program locates the tracks by using the addresses in the special area on disk. All disks, fixed and removable, have such an area. The program will do conditional assignment for all tracks identified in the area (one at a time), as long as there are alternate tracks available for assignment.

#### *Unconditional Assignment*

You have used the Alternate Track Assignment program to do conditional assignment. The test on the track indicated that the track was not defective (an alternate, therefore, was not assigned). But the track still causes reading or writing errors, and you want to assign an alternate to it. For this reason you should assign an alternate track using unconditional assignment. Alternate tracks are assigned without first testing the condition of the tracks suspected of being defective. (A conditional assignment is forced after an unconditional request to check any other tracks that previously caused errors.)

#### *Cancel Prior Assignment*

Cancel prior assignment is used when a defective track was found, but all alternates are in use. You want to free an alternate so you can recover the data from the defective track. Canceling an assignment involves transferring the data from an alternate track back to the track to which the alternate was assigned. Prior to transferring the data back to the original track, the Alternate Track Assignment program tests the condition of the original track. If the track is found defective, the program stops and one of three options is taken:

- You leave the assignment as it is but continue checking other assignments (if there are any), or the program ends.
- You cancel the assignment regardless of the condition of the original track. Before freeing the alternate, however, you would normally copy (to another disk) the file or library entry that uses the alternate. This saves the data that is already on the alternate.
- You test the track again.

You must run the File and Volume Label Display program to determine to what tracks alternates are assigned.

### Number of Alternate Tracks

There are six tracks on every disk that can be used as alternates. These tracks, in addition to tracks 0 and 1, can't be replaced; that is, they can't have an alternate assigned to them.

### Surface Analysis

You can tell the program to do surface analysis from 1 to 255 times before judging whether or not tracks are defective. A track is judged usable only after successfully completing every check. If at any time during surface analysis incorrect data is found, the track on which the data was written is judged defective, and an alternate is assigned to it.

## CONTROL STATEMENTS

You must supply the following control statements to specify the program options you want:

1. *ALT statement*—indicates the name and unit of the disk containing the defective track, the number of times you want surface analysis done, and the tracks to which you want to assign alternates or for which you wish to cancel assignment of an alternate track. There can be only 6 ALT statements per job.

2. *END statement*—indicates the end of control statements.

For each use, the program requires the statements in the order they are listed: ALT, END.

### Control Statement Summary

| Use | Control Statements |
|---|---|
| Conditional Assignment | // ALT PACK-name, UNIT-code,VERIFY-number<br>// END |
| Unconditional Assignment | // ALT PACK-name,UNIT-code,ASSIGN-$\left\{\begin{matrix}\text{track}\\ \text{'tracks'}\end{matrix}\right\}$,VERIFY-number<br>// END |
| Cancle Prior Assignment | // ALT PACK-name,UNIT-code,UNASSIGN-$\left\{\begin{matrix}\text{track}\\ \text{'tracks'}\end{matrix}\right\}$,VERIFY-number<br>// END |

### Parameter Summary: ALT (Alternate) Statement

| Parameter | Meaning | |
|---|---|---|
| PACK-name | Name of the disk. | |
| UNIT-code | Location of the disk. Possible codes are R1, F1, R2, F2, D1, D2. | |
| VERIFY-number | In testing the condition of a track, do surface analysis the number of times indicated (number can be 1-255). If VERIFY parameter is omitted, do surface analysis once. | |
| ASSIGN-track | Assign an alternate (unconditionally) to one track. | Use track numbers 8-405 to identify tracks. Tracks 0-7 are used by the system and cannot be assigned alternates. |
| ASSIGN-'track,track,...' | Assign one alternate (unconditionally) to each track (maximum is six). | |
| UNASSIGN-track | Cancel one alternate track assignment. | Use track numbers 8-405 to which alternates are assigned. |
| UNASSIGN-'track,track,...' | Cancel two or more alternate track assignments (maximum is six). | |

## PARAMETER DESCRIPTIONS

### PACK Parameter

The PACK parameter (PACK-name) tells the program the name of the disk containing the defective tracks. This is the name written on the disk by the Disk Initialization program.

The Alternate Track Assignment program compares the name in the PACK parameter with the name on the disk to ensure they match. In this way, the program ensures that it is using the right disk.

### UNIT Parameter

The UNIT parameter (UNIT-code) indicates the location of the disk containing defective tracks. Codes for the possible locations are as follows:

| Code | Location |
|---|---|
| R1 | Removable disk on drive 1. |
| F1 | Fixed disk on drive 1. |
| R2 | Removable disk on drive 2. |
| F2 | Fixed disk on drive 2. |

### VERIFY Parameter

The VERIFY parameter (VERIFY-number) enables you to indicate the number of times you want the program to do surface analysis before judging whether or not the track is defective. The number can be from 1-255. If you omit the parameter, the program does surface analysis once.

### ASSIGN Parameter

The ASSIGN parameter (ASSIGN-track) applies to unconditional assignment. It tells the program which tracks you want alternates assigned to.

You can assign alternates to any tracks except 0-7. Tracks 0-7 are for system use only.

The form of the ASSIGN parameter depends on the number of tracks you want to specify. For one track, use ASSIGN-track; for two tracks, use ASSIGN-'track,track'; and so on. You can specify up to six tracks.

Use the track numbers (8-405) to identify the tracks. For example, the parameter ASSIGN-'50, 301,353' causes the program to assign alternate tracks to tracks 50, 301, and 353.

### UNASSIGN Parameter

The UNASSIGN parameter (UNASSIGN-track) applies to cancelling alternate track assignments. It identifies tracks for which you want the program to cancel assignments.

You can cancel up to six assignments. The form of the UNASSIGN parameter depends on the number of assignments you want to cancel. For one assignment, use UNASSIGN-track; for two assignments, use UNASSIGN-'track,track'; and so on.

Use the track numbers (8-405) to identify the tracks. For example, the parameter UNASSIGN-'50,301,352' causes the program to cancel alternate-track assignments for tracks 50, 301, and 352.

## OCL CONSIDERATIONS

### LOAD Sequence

| Keywords [1] | Responses [2] | Considerations |
|---|---|---|
| READY | LOAD | None |
| LOAD NAME | $ALT | Name of Alternate Track Assignment program. |
| UNIT | R1, R2, F1, or F2 | Location of disk containing Alternate Track Assignment program. |
| MODIFY | RUN | None |

[1] Only the keywords listed here are required. You can bypass the rest.

[2] You end every response by pressing PROG START.

### BUILD Sequence

| Keywords [1] | Responses [2] | Considerations |
|---|---|---|
| READY | BUILD | None |
| BUILD NAME | procedure name | Name by which procedure will be identified in source library. |
| UNIT | R1, R2, F1, or F2 | Location of disk containing source library. |
| LOAD NAME | $ALT | Name of Alternate Track Assignment program. |
| UNIT | R1, R2, F1, or F2 | Location of disk containing Alternate Track Assignment program. |
| MODIFY | INCLUDE<br>utility control statements<br>RUN | Response when including control statements in procedure. |
|  | OR |  |
|  | RUN | Response when not including control statements in procedure. |

[1] Only the keywords listed here are required. You can bypass the rest.

[2] You end every response by pressing PROG START.

**EXAMPLE**

**Conditional Assignment**

*Situation*

Assume that during a job the system printed a message telling the operator it found a defective track on the removable disk on drive 1. (The name of the disk is BILLNG.) Before doing more jobs, the operator wants to use the Alternate Track Assignment program to check the condition of the track and assign an alternate to the track if it is defective.

*Statements*

```
READY                            - LOAD
*************************
010     LOAD    NAME             - $ALT
011             UNIT             - F1
020     DATE    (XX/XX/XX)       -
030     SWITCH  (00000000)       -
040     FILE    NAME             -
*************************
MODIFY
RUN
```

**OCL LOAD Sequence.**

Boxed areas are operator responses.

Keywords for which no responses are shown are the ones bypassed. If you press ENTER– after responding to UNIT, the DATE, SWITCH, and FILE NAME keywords are not prompted.

RUN is the response to MODIFY even though the two words do not appear on the same line.

```
  ENTER '//' CONTROL STATEMENT
```

Message printed by Alternate Track Assignment program.

```
// ALT PACK-BILLNG,UNIT-R1
```

Control statement supplied by operator.

```
  ENTER '//' CONTROL STATEMENT
// END
```

System reprompts. END statement terminates sequence.

*Explanation*

- Alternate Track Assignment program is loaded from the fixed disk on drive 1 (UNIT-F1 in OCL sequence).
- The name of the disk (BILLNG) and its location (removable disk on drive 1) are indicated by the PACK and UNIT parameters in the ALT statement.
- Because we omitted the VERIFY parameter from the ALT statement, the program does surface analysis once when it tests the condition of the track.

## MESSAGES FOR ALTERNATE TRACK ASSIGNMENT

| Message | Meaning |
|---|---|
| ALTERNATE TRACK ASSIGNED | This message is printed when an alternate track has been assigned to a defective track and the data has been transferred to the alternate track. |
| PRIMARY TRACK HAS BEEN TESTED OK | This message is printed when it is determined that a primary track is not defective. |
| PRIMARY TRACK STILL DEFECTIVE | This message is printed when the Alternate Track Assignment program determines that the track is still defective. |
| DATA TRANSFERRED BACK TO PRIMARY TRACK | This message is printed when the data is transferred back to the primary track. |
| **SECTOR WITH DATA ERROR** | This message is printed when the Alternate Track Assignment program found an error when transferring data. The sector that has the error is printed out. |
| PRIMARY TRACK xxx ALTERNATE TRACK yyy, UNIT-zz | This message is printed after ALTERNATE TRACK ASSIGNED and DATA TRANSFERRED BACK TO PRIMARY TRACK. xxx is the primary track number, yyy is the alternate track number, and zz is the unit involved. |

## Alternate Track Rebuild Program ($BUILD)

An alternate track assigned by the Alternate Track Assignment program may contain some incorrect data. In order to correct this data, you must use the Alternate Track Rebuild program.

### FUNCTIONS

The process of correcting data consists of:

- Locating incorrect data.
- Replacing incorrect data.

#### Locating Incorrect Data

The Alternate Track Assignment program prints a listing of all track sectors that may contain incorrect data. You will find, on the listing, the name of the disk, the track and sector numbers of the area suspected of containing incorrect data, and the data from these sectors.

#### Replacing Incorrect Data

The Alternate Track Rebuild program will replace the number of characters you indicate in the positions you indicate. You must key the new characters in hexadecimal form. These characters are called *substitute data.*

### OPTIONS

The Alternate Track Rebuild program gives you the following options:

- You may correct as many characters as you wish on one track.
- You may correct data on more than one track.

You specify the options you want in control statements (see *Control Statements* in this chapter).

#### Control Statement Summary

**Control Statement**

```
// REBUILD PACK-name,UNIT-code,TRACK-location,LENGTH-
number,DISP-position
   {
Substitute Data
   }
// END
```

To replace characters 1-12 and 75-78 of a sector, you can use either of the following:

1. Use one REBUILD statement to replace all the characters with a LENGTH parameter of 78.
2. Use one REBUILD statement for every set of positions you correct.

The data you want to substitute must follow the REBUILD statements to which it applies. The order of the statements and data in the preceding example would be:

```
// REBUILD statement          for positions 1-78
data
// END

// REBUILD statement          for positions 1-12
data
// REBUILD statement          for positions 75-78
data
// END
```

#### Parameter and Substitute Data Summary

| **REBUILD Statement** | **Meaning** |
|---|---|
| PACK-name | Name of the disk. |
| UNIT-code | Location of the disk. Possible codes are R1, F1, R2, F2. |
| TRACK-location | Number of track and sector containing incorrect data. Number is printed by Alternate Track Assignment program. Track number must be three digits; sector number must be two digits. (TRACK-01109 means track 11 sector 0). |
| LENGTH-number | Number of characters being replaced. Number can be 2-256 and must be a multiple of 2 (2, 4, 6, etc.). |
| DISP-position | Position of the first character being replaced in the sector. Position can be 1-255. |

{
Substitute Data
}

Code each character in hexadecimal form. Follow every second character, except the last, with a comma. EXAMPLE: The numbers 123456 would be coded as F1F2,F3F4,F5F6. (Appendix A lists the hexadecimal codes for System/3 characters.)

**Number of Characters**

You may replace from 2 to 256 characters on one track in one run. You can do this by replacing all the characters (including correct data) or just groups of incorrect data.

**Number of Tracks**

The Alternate Track Assignment program prints the track and sector numbers for those areas that contain incorrect data. You can correct one or more of these tracks in one program run. The possible tracks you can correct are 8 through 405 and the sectors are 0 through 23. Tracks 0 through 7 can't be corrected.

## PARAMETER AND SUBSTITUTE DATA DESCRIPTIONS

### PACK Parameter

The PACK parameter (PACK-name) tells the program the name of the disk that contains the alternate track being corrected. This name is the one written on the disk by the Disk Initialization program.

The Alternate Track Rebuild program compares the name in the PACK parameter with the name on the disk to ensure they match. In this way, the program ensures that the program is using the right disk.

### UNIT Parameter

The UNIT parameter (UNIT-code) indicates the location of the disk that contains the alternate track being corrected. Codes for the possible locations are as follows:

| Code | Location |
|---|---|
| R1 | Removable disk on drive 1. |
| F1 | Fixed disk on drive 1. |
| R2 | Removable disk on drive 2. |
| F2 | Fixed disk on drive 2. |

### TRACK Parameter

The TRACK parameter (TRACK-location) identifies the track and sector that contains the data being corrected. The defective track, not the alternate track, is the one you refer to. Referencing the defective track is the same as referencing the alternate track.

Use the track and sector numbers in the TRACK parameter. The possible track numbers are 008-405. Always use three digits. The possible sector numbers are 00-23. Always use two digits. The track number must precede the sector number. For example, the parameter TRACK-11019 means track 110, sector 19.

Track and sector numbers are printed by the Alternate Track Assignment program when it prints data from sectors that contain incorrect data.

### LENGTH Parameter

The LENGTH parameter (LENGTH-number) tells the program how many characters you are replacing in the sector. You must replace characters in multiples of 2 (2, 4, 6, and so on). The maximum is 256, which is the capacity of a sector.

Length applies to characters that occupy consecutive positions in the sector. If the characters you want to replace do not occupy consecutive positions, you must either replace more characters or use more than one REBUILD statement. For example, to replace characters 10-11 and 24-25 in a sector, you can do either of the following:

1. Use one REBUILD statement to replace characters 10-25 (LENGTH-16).
2. Use two REBUILD statements to replace characters 10-11 (LENGTH-2) and 24-25 (LENGTH-2).

### DISP (Displacement) Parameter

The DISP parameter (DISP-position) indicates the position of the first character being replaced in the sector. The position of the first character in the sector is 1; the position of the second character is 2; and so on. The maximum position is 255.

Beginning at the position you indicate, the Alternate Track Rebuild program replaces the number of characters you indicate in the LENGTH parameter.

### Substitute Data

After each REBUILD statement, you must key the substitute characters that apply to that statement. The characters must be in hexadecimal form. Appendix J shows the hexadecimal forms of the characters in the standard character set.

Include a comma after every second character. For example, the data F1F2,F3F4,F5F6 represents 123456. F1 is the hexadecimal form of 1; F2 is the hexadecimal form of 2; and so on.

Key only the number of characters you indicated in the LENGTH parameter in the REBUILD statement.

## OCL CONSIDERATIONS

### LOAD Sequence

| Keywords (1) | Responses (2) | Considerations |
|---|---|---|
| READY | LOAD | None |
| LOAD NAME | $BUILD | Name of Alternate Track Rebuild program. |
| UNIT | R1, R2, F1, or F2 | Location of disk containing Alternate Track Rebuild program. |
| MODIFY | RUN | None |

(1) Only the keywords listed here are required. You can bypass the rest.

(2) You end every response by pressing PROG START.

### BUILD Sequence

| Keywords (1) | Responses (2) | Considerations |
|---|---|---|
| READY | BUILD | None |
| BUILD NAME | procedure name | Name by which procedure will be identified in source library. |
| UNIT | R1, R2, F1 or F2 | Location of disk containing source library. |
| LOAD NAME | $BUILD | Name of Alternate Track Rebuild program. |
| UNIT | R1, R2, F1, or F2 | Location of disk containing Alternate Track Rebuild program. |
| MODIFY | RUN* | Response when not including control statements in procedure. |

(1) Only the keywords listed here are required. You can bypass the rest.

(2) You end every response by pressing PROG START.

*BUILD does not allow utility control statements in the procedure.

**EXAMPLE**

**Correcting Characters on an Alternate Track**

*Situation*

Assume that the Alternate Track Assignment program printed the following information:

```
PACK-R1

TRACK AND SECTOR BAD-05020

ABCDEF GH1 34567890... )  (Assume the entire contents of the sector
       B   A           }  was printed.)
       6   5           )
```

It means that errors were detected in sector 20 of track 50 on the removable disk on drive 1. (Assume the name of the disk is BILLNG.)

In checking the characters printed by the program, you found that the seventh and eleventh characters in the sector are incorrect and you want the operator to run the Alternate Track Rebuild program to correct them.

*Statements*

```
READY                        -  LOAD
************************
010    LOAD    NAME          -  $BUILD
011            UNIT          -  F1
020    DATE    (XX/XX/XX)    -
030    SWITCH  (00000000)    -
040    FILE    NAME          -
************************
MODIFY
RUN
```

**OCL LOAD Sequence.**

Boxed areas are operator responses.

Keywords for which no responses are shown are the ones bypassed. If you press ENTER– after responding to UNIT, the DATE, SWITCH, and FILE NAME keywords are not prompted.

RUN is the response to MODIFY even though the two words do not appear on the same line.

```
   ENTER '//' CONTROL STATEMENT
// REBUILD PACK-BILLNG ,UNIT-R1,TRACK-05020,LENGTH-6,DISP-7
   ENTER HEX DATA STATEMENT
C6C7,C8F1,F2F3
   ENTER '//' CONTROL STATEMENT
// END
```

Message printed by Alternate Track Rebuild program. (ENTER '//' CONTROL STATEMENT)

Message printed by Alternate Track Rebuild program. (ENTER HEX DATA STATEMENT)

Message printed by Alternate Track Rebuild program. (ENTER '//' CONTROL STATEMENT)

Control statements and substitute data supplied by the operator

*Explanation*

- Alternate Track Rebuild program is loaded from the fixed disk on drive 1 (UNIT-F1 in OCL sequence).
- The name of the removable disk (BILLNG) and its location (drive 1) are indicated in the PACK and UNIT parameters in the REBUILD statement.
- The sector containing the incorrect characters is sector 20 of the alternate track assigned to track 50 (TRACK-05020). The seventh character in the sector is the first character being replaced (DISP-7).
- The seventh through twelfth characters in sector 20 are being replaced (LENGTH-6). We included the twelfth character because the number of characters being replaced must be a multiple of 2. By also replacing the characters between the incorrect ones, we needed only one REBUILD statement.
- The substitute characters follow the REBUILD statement. They are F (C6), G (C7), H (C8), 1 (F1), 2 (F2), and 3 (F3).

# File and Volume Label Display Program ($LABEL)

You may need to obtain specific information about a file; find space available for libraries or new files; or check the contents of a disk for libraries, temporary data files, or permanent data files. In order to do any of these, you need information contained in the volume table of contents (VTOC). To obtain this information you must use the File and Volume Label Display program.

## FUNCTIONS

This program allows you to:

- Print VTOC information.
- Print headings for file information.

### Print VTOC Information

The VTOC is an area on disk that contains information about the contents of the disk. Every disk contains a VTOC. The File and Volume Label Display program allows you to print this information.

The printed VTOC information is a readable, up-to-date record of the contents of the disk. There can be any number of reasons why you might need the information. Some of the more common ones are as follows:

1. Before re-initializing a disk, you might want to check its contents to ensure that it contains no libraries, permanent data files, or temporary data files.
2. You want to find out what disk areas are available for libraries or new files.
3. You want specific file information, such as the file name, designation (permanent, temporary, scratch), or the space reserved for the file.

### Print Headings

If the file information you requested from the VTOC overflows onto another page, the program prints the headings for the information at the top of the next page. It will do this for each succeeding new page.

## OPTIONS

The File and Volume Label Display program gives you the following options:

1. Print the entire Volume Table of Contents (VTOC) from a disk.
2. Print only the VTOC information for certain data files. You may specify up to 20 file names in one run.

In both cases, the program also prints the name of the disk.

### Entire Contents of VTOC

There are many reasons why you may want to print the entire VTOC. You may want to check which tracks are assigned alternates or how many alternate tracks are still available for use. You may also want to check the boundaries of libraries or check for permanent or temporary data files.

### File Information Only

You may request information for specific files. You may want this information to find out file names, file designations, or disk areas reserved for files. You may also use it to determine the relationship of multivolume files.

### Number of File Names

When you specify a file name, you must use the name that identifies the file in the VTOC. You are allowed to specify up to 20 file names in one program run.

## CONTROL STATEMENTS

You must supply the following control statements to specify the program options you want:

1. *DISPLAY statement* — indicates whether you want the entire VTOC or specific file information from the VTOC. It also indicates the unit of the disk containing VTOC information.

2. *END statement* — indicates the end of control statements.

### Control Statement Summary

| Uses | Control Statement ❶ |
|---|---|
| Print entire VTOC: | // DISPLAY UNIT-code,LABEL-VTOC<br>// END |
| Print only file information from VTOC: | // DISPLAY UNIT-code,LABEL-{filename / 'filenames'} ❷<br>// END |

❶ For each use, the program requires the statements in the order they are listed: DISPLAY, END.

❷ The number of filenames you list for a program run may not exceed 20. (VTOC is considered as one filename.)

### Parameter Summary (Display Statement)

| Parameter | Meaning |
|---|---|
| UNIT-code | Location of the disk containing the VTOC information being printed. Possible codes are R1, F1, R2, F2, D1, D2. |
| LABEL-VTOC | Print entire contents of VTOC. |
| LABEL-filename | Print VTOC information for one file. |
| LABEL-'filename,filename,...' | Print VTOC information for more than one file. The number of filenames you list for a program run may not exceed 20. (VTOC is considered as one filename.) |

## PARAMETER DESCRIPTIONS

### UNIT Parameter

The UNIT parameter (UNIT-code) indicates the location of the disk containing the VTOC information being printed. Codes for the possible locations are as follows:

| Code | Location |
|---|---|
| R1 | Removable disk on drive 1. |
| F1 | Fixed disk on drive 1. |
| R2 | Removable disk on drive 2. |
| F2 | Fixed disk on drive 2. |

### LABEL Parameter

The LABEL parameter indicates the information you wanted printed: the entire contents of the VTOC or only the information for certain files. The VTOC is an area on disk that contains information about the contents of the disk. Every disk, fixed and removable, contains a VTOC. The meaning of the VTOC information is as follows:

| Heading | Meaning |
|---|---|
| PACK-name | Name of the disk. |
| ID-characters | Additional disk identification (if any). |
| NUMBER OF ALTERNATE TRACKS AVAILABLE-number | Number of alternate tracks available for assignment. |
| TRACKS WITH ALTERNATE ASSIGNED | Numbers of primary tracks that have been assigned an alternate. |
| DEFECTIVE ALTERNATE TRACKS | Numbers of the alternate tracks that are defective. |
| DEVICE CAPACITY-number | Disk drive capacity (number of tracks). |
| LIBRARY EXTENT | Boundary of libraries on the disk. (If the disk contains no libraries, these headings are not printed.) |
| START | Track on which library begins. } If the disk contains both source and object library, START refers to beginning of source library and END refers to end of object library. |
| END | Track on which library ends. |
| EXTENDED END | Object library only. Track on which extension to library ends. When object library is full, temporary entries can be placed in space following end of library, provided that space is available. |
| AVAILABLE SPACE ON PACK | Available disk areas. |
| LOCATION | First track in available area. |
| TRACKS | Number of tracks available. |
| PACK-name | Name of the disk. |
| UNIT-code | Location of the disk containing the VTOC information. |
| DATE-xx/xx/xx | Current system date. |
| FILE NAME | Name that identifies file in VTOC. |
| FILE DATE | Date given the file when file was placed on disk. |
| KEEP TYPE | File designation:<br>P = permanent<br>T = temporary<br>S = scratch |
| FILE TYPE | File type:<br>I = indexed<br>C = consecutive<br>D = direct<br>B = basic file |

| Heading | Meaning |
|---|---|
| REC LEN | Number of characters in each record in file. |
| KEY LEN | Indexed files only. Number of characters in each record key. |
| KEY LOC | Indexed files only. Position in record occupied by last character of record key. |
| NEXT AVAIL RECORD | Beginning location of next available record in file. Location is track, sector, and position within sector. EXAMPLE: 099/18/006 = track 99, sector 18, position 6. If the first byte of the next available record occurs in the next track after the end track of DATA START END then this field will contain **** |
| NEXT AVAIL KEY | Indexed files only. Beginning location of next available record key in index portion of file. Location is track, sector, and position within sector. EXAMPLE: 090/10/006 = track 90, sector 10, position 6. If the first byte of the next available key occurs in the next track after the end track of INDEX START END, then this field will contain **** |
| INDEX START END | Indexed files only. Tracks on which index starts (START) and ends (END). |
| DATA START END | Disk area reserved for the file. START is the first track of the area. END is the last track. For indexed files, this refers to the data portion of the file. |
| VOL SEQ | VOL SEQ applies to multivolume files only. It indicates the order of this disk as it relates to the other disks containing the remaining portion of the file. |

## OCL CONSIDERATIONS

### LOAD Sequence

| Keywords ❶ | Responses ❷ | Considerations |
|---|---|---|
| READY | LOAD | None |
| LOAD NAME | $LABEL | Name of File and Volume Label Display program. |
| UNIT | R1, R2, F1, or F2 | Location of disk containing File and Volume Label Display program. |
| MODIFY | RUN | None |

❶ Only the keywords listed here are required. You can bypass the rest.

❷ You end every response by pressing PROG START.

### BUILD Sequence

| Keywords ❶ | Responses ❷ | Considerations |
|---|---|---|
| READY | BUILD | None |
| BUILD NAME | procedure name | Name by which procedure will be identified in source library. |
| UNIT | R1, R2, F1, or F2 | Location of disk containing source library. |
| LOAD NAME | $LABEL | Name of File and Volume Label Display program. |
| UNIT | R1, R2, F1, or F2 | Location of disk containing File and Volume Label Display program. |
| MODIFY | INCLUDE<br>utility control statements<br>RUN | Response when including control statements in procedure. |
| | OR | |
| | RUN | Response when not including control statements in procedure. |

❶ Only the keywords listed here are required. You can bypass the rest.

❷ You end every response by pressing PROG START.

**EXAMPLE**

**Printing VTOC Information for Two Files**

*Statements*

```
READY                           - LOAD
************************
010    LOAD    NAME             - $LABEL
011            UNIT             - F1
020    DATE    (XX/XX/XX)       -
030    SWITCH  (00000000)       -
040    FILE    NAME             -
************************
MODIFY
RUN
```

**OCL LOAD Sequence.**

Boxed areas are operator responses.

Keywords for which no responses are shown are the ones bypassed. If you press ENTER- after responding to UNIT, the DATE, SWITCH, and FILENAME keywords are not prompted.

RUN is the response to MODIFY even though the two words are not on the same line.

```
   ENTER '//' CONTROL STATEMENT
// DISPLAY UNIT-R1,LABEL-'BILLNG,INV01'
   ENTER '//' CONTROL STATEMENT
// DISPLAY UNIT-F2,LABEL-VTOC
   ENTER '//' CONTROL STATEMENT
// END
```

Message printed by File and Volume Label Display program.

Control statement supplied by operator.

Sequence repeats until operator enters END statement.

*Explanation*

- The File and Volume Label Display program is loaded from the fixed disk on drive 1 (UNIT-F1 in OCL sequence).
- The files for which information is printed are named BILLNG and INVO1 (LABEL-'BILLNG,INVO1' in first DISPLAY statement). They are located on the removable disk on drive 1 (UNIT-R1).
- Information from the entire VTOC on F2 is printed.

You may find that you no longer need the information in a file. You can free the space in a file for use by new files by using the File Delete program.

The program may be used on temporary, scratch and permanent files. To delete permanent files, you must use the File Delete program. You can scratch temporary files by using the File Delete program or by changing the file designation from temporary to scratch (using the OCL keyword RETAIN) when you use the file.

## FUNCTIONS

This program allows you to:

- Eliminate file references in the VTOC.
- Erase information in a file.

### VTOC File References

The File Delete program allows you to remove the VTOC references to a file by removing the reference (SCRATCH statement). However, the file reference is not physically removed from the VTOC until normal end of job has occurred.

### Erase File Information

You may erase a file from the disk as well as removing the file reference in the VTOC (REMOVE statement). This involves erasing the information contained in the file. Its space is then made available for any new files.

## OPTIONS

The File Delete program gives you the following options:

- You may choose to delete files in one of two ways: remove or scratch.
- You may delete some or all files from a disk.
- You may specify up to 52 file names in one job.

You specify the options you want in control statements (see *Control Statements* in this chapter).

### Deleting a File

If you wish to delete a permanent file, you must use the File Delete program. If you delete a temporary file, you may use either the File Delete program or change the file designation when you use the file. You may either remove or scratch a file. No file is physically scratched or removed from the VTOC until end of job has occurred.

*Removing a File*

When you remove a file from a disk (REMOVE statement), you are removing the file reference from the VTOC. You may also erase the file from the disk, leaving its area available for use by other files.

*Scratching a File*

The File Delete program allows you to scratch a file if you find you may need to reference it later. The SCRATCH statement does not erase files from the disk. It changes their designation to scratch (S) in the Volume Table of Contents (VTOC). By doing this, the program makes the areas that contain the files available for other files or for system programs. You can use the file until a permanent file is created in its place.

A halt will occur if an attempt is made to create a new multivolume file that will have the same label on disk as an existing single volume file, or an attempt is made to create a single volume file bearing the same label as an existing multivolume file. The halt will occur even though the existing file is a scratch file.

### Number of Files

You may remove some or all files on a disk. If a file name applies to more than one file, all the files with that name are deleted. You can keep this from happening by identifying the files with both name and date.

### Number of File Names

You may specify as many file names as the control statement will allow. If you specify more, you must use more than one statement. However, you are only allowed to specify 52 file names in one job.

## CONTROL STATEMENTS

1. *REMOVE statement*—indicates the name and unit of the disk, what files are to be removed, and whether or not you are erasing the data for the file.
2. *SCRATCH statement*—indicates the name and unit of the disk and what files you wish to scratch.
3. *END statement*—indicates the end of control statements.

### Control Statement Summary

| Use | Control Statements ❶ |
|---|---|
| Scratch all files in the VTOC: | // SCRATCH PACK-name, UNIT-code, LABEL-VTOC<br>// END |
| Scratch only one file in the VTOC: | // SCRATCH PACK-name, UNIT-code, LABEL-filename, DATE-date ❷ |
| Scratch multiple files in the VTOC: | // SCRATCH PACK-name, UNIT-code, LABEL- $\left\{\begin{matrix}\text{filename}\\ \text{'filenames'}\end{matrix}\right\}$ |
| Remove all files from disk: | // REMOVE PACK-name, UNIT-code, LABEL-VTOC, DATA- $\left\{\begin{matrix}\text{NO}\\ \text{or}\\ \text{YES}\end{matrix}\right\}$ |
| Remove only the files named from disk: | // END<br>// REMOVE PACK-name, UNIT-code, LABEL- $\left\{\begin{matrix}\text{filename}\\ \text{'filenames'}\end{matrix}\right\}$ DATE-date, DATA- $\left\{\begin{matrix}\text{NO}\\ \text{or}\\ \text{YES}\end{matrix}\right\}$<br>// END |

❶ For each use, the program requires the statements in the order they are listed: SCRATCH, END, or REMOVE, END.

❷ Use this form of the SCRATCH or REMOVE statement when two or more files have the same name and you want to delete one of them.

**Parameter Summary**

| Parameter | Meaning |
|---|---|
| PACK-name | Name of the disk. |
| UNIT-code | Location of the disk. Possible codes are R1, F1, R2, F2, D1, D2. |
| LABEL-VTOC | Scratch or remove all files from the VTOC. |
| LABEL-filename | Scratch or remove only the file named in the VTOC. } Use names that identify files in VTOC. (These are the names you gave the files when you placed them on disk.) |
| LABEL-'filename,filename,... | Scratch or remove only the files named in the VTOC. |
| DATE-date | Date of the file being deleted. Date must be a 6-digit number. EXAMPLE: DATE-062070 means June 20, 1970. |
| DATA- { NO or YES } | Delete files from VTOC and/or disk. |

## PARAMETER DESCRIPTIONS

### Pack Parameter

The PACK parameter (PACK-name) tells the program the name of the disk that contains the files being deleted. The name you supply in this parameter is the one written on the disk by the Disk Initialization program.

The File Delete program compares the name in the PACK parameter with the name on the disk to ensure they match. In this way, the program ensures that it is using the right disk.

### Unit Parameter

The UNIT parameter (UNIT-code) tells the program the location of the disk containing the files being deleted. Codes for the possible locations are as follows:

| Code | Location |
|---|---|
| R1 | Removable disk on drive 1. |
| F1 | Fixed disk on drive 1. |
| R2 | Removable disk on drive 2. |
| F2 | Fixed disk on drive 2. |

### Label Parameter

The LABEL parameter identifies the files you want to delete from the disk. Its form depends on the files you are deleting:

| Form | Files Deleted |
|---|---|
| LABEL-VTOC | All of them. |
| LABEL-filename | Only the file that is named. The name can apply to more than one file. If it does, all of those files are deleted unless you use a DATE parameter to identify a particular one. |
| LABEL-'filename, filename,...' | Only the files that are named. A name can apply to more than one file. If it does, all of those files are deleted. (You can list as many filenames as the statement can hold; the statement length, however, is restricted to 96 characters. Additional REMOVE or SCRATCH statements may be used for additional filenames. The maximum number of files that can be deleted in one run is 52.) |

## Date Parameter

The DATE parameter (DATE-date) applies to two or more files that have the same name. It tells the program the date of the one you want to delete.

Every file on disk has a date, which is given to the file at the time it is created. When two or more files have the same name, the dates are used to tell one file from another.

The date is a six-digit number: two digits for day, two for month, and two for year. Day, month, and year can be in one of two orders: (1) month, day, year and (2) day, month, year. For example 061870 and 180670 both mean June 18, 1970.

In the DATE parameter, be sure to specify day, month, and year in the same order as when you placed the file on disk.

## Data Parameter (REMOVE Statement Only)

The DATA parameter lets you delete the files specified directly from the disk as well as from the VTOC.

If YES is coded in this parameter then the file specified will be removed from the disk and any reference to it in the VTOC will be removed. In addition, a message will be printed on the system logging device for each file removed from the disk in this format:

```
'DATA REMOVED FOR FILE XXXXXX
DATE 000000'
```

DATA-YES should be used only if file security is required. The time needed to remove the data is much greater than the time needed to remove the VTOC entry.

If NO is coded in this parameter, then the file specified will not be removed from the disk. However, any reference to it in the VTOC will be removed. If this parameter is not used, DATA-NO is assumed.

## OCL CONSIDERATIONS

### LOAD Sequence

| Keywords [1] | Responses [2] | Considerations |
|---|---|---|
| READY | LOAD | None |
| LOAD NAME | $DELET | Name of File Delete program. |
| UNIT | R1, R2, F1, or F2 | Location of disk containing File Delete program. |
| MODIFY | RUN | None |

[1] Only the keywords listed here are required. You can bypass the rest.

[2] You end every response by pressing PROG START.

### BUILD Sequence

| Keywords [1] | Responses [2] | Considerations |
|---|---|---|
| READY | BUILD | None |
| BUILD NAME | procedure name | Name by which procedure will be identified in source library. |
| UNIT | R1, R2, F1, or F2 | Location of disk containing source library. |
| LOAD NAME | $DELET | Name of File Delete program. |
| UNIT | R1, R2, F1, or F2 | Location of disk containing File Delete program. |
| MODIFY | INCLUDE utility control statements RUN | Response when including control statements in procedure. |
| | OR | |
| | RUN | Response when not including control statements in procedure. |

[1] Only the keywords listed here are required. You can bypass the rest.

[2] You end every response by pressing PROG START.

**EXAMPLE**

**Deleting One of Several Files Having The Same Name**

*Situation*

Assume that three files on a removable disk have the same name: INV01. The dates of these files are 6/16/70, 8/18/70, and 11/15/70. You want to delete the 6/16/70 version.

*Statements*

```
READY                              - LOAD
*************************
010     LOAD    NAME               - $DELET
011             UNIT               - F1
020     DATE    (XX/XX/XX)         -
030     SWITCH  (00000000)         -
040     FILE    NAME               -
*************************
MODIFY
RUN
```

**OCL Load Sequence**

Boxed areas are operator responses.

Keywords for which no responses are shown are the ones bypassed. If you press ENTER– after responding to UNIT, the DATE, SWITCH, and FILE NAME keywords are not prompted.

RUN is the response to MODIFY even though the two words do not appear on the same line.

```
   ENTER '//' CONTROL STATEMENT
// SCRATCH PACK-00001,LABEL-INV01,UNIT-R1,DATE-061670
   ENTER '//' CONTROL STATEMENT
// END
```

Message printed by File Delete program.

Control statement supplied by operator.

Sequence repeats until operator enters END statement.

*Explanation*

- File Delete program is loaded from the fixed disk on drive 1 (UNIT-F1 in OCL sequence).
- Disk that contains the file being deleted is named 00001 (PACK-00001 in SCRATCH statement).
- Because two other files have the name INV01, the date (061670) is needed to complete the identification of the file you want to delete (LABEL-INV01 and DATE-061670).
- The removable disk containing the file to be deleted is on drive 1 (UNIT-R1).

**Removing One File**

*Situation*

You want to remove a file named INV02 from the pack mounted on R1.

*Statements*

```
READY                              - LOAD
*************************
010    LOAD    NAME                - $DELET
011            UNIT                - F1
020    DATE    (XX/XX/XX)  -
030    SWITCH  (00000000)  -
040    FILE    NAME                -
*************************
MODIFY
RUN
```

**OCL Load Sequence.**

Boxed areas are operator responses.

Keywords for which no responses are shown are the ones bypassed. If you press ENTER– after responding to UNIT, the DATE, SWITCH, and FILE NAME keywords are not prompted.

RUN is the response to MODIFY even though the two words do not appear on the same line.

```
   ENTER '//' CONTROL STATEMENT
```

Message printed by File Delete program.

```
// REMOVE PACK-00001,LABEL-INV02,UNIT-R1,DATA-YES
```

Control statement supplied by operator.

```
'DATA REMOVED FOR FILE xxxxxx DATE 000000'
```

Printed by File Delete.

```
   ENTER '//' CONTROL STATEMENT
// END
```

Sequence repeats until operator enters END statement.

*Explantion*

- File Delete program is loaded from the fixed disk on drive 1 (UNIT-F1 in OCL sequence).
- Disk that contains the file being removed is named 00001 (PACK-00001 in REMOVE statement).
- The removable disk containing the file to be removed is on drive 1 (UNIT-R1).
- DATA-YES indicates that the file data as well as the file VTOC reference is to be removed.

## Disk Copy/Dump Program ($COPY)

You may need to check records in a file for errors. In order to do this you need to *print a copy* of the file. It is important to provide a reserve disk or file for disks containing libraries or permanent data files in case something happens to the original disk or file. You can *copy* the disk or file using the Disk Copy/Dump program.

### FUNCTIONS

Copying a disk or file involves:

- Identifying disk or file locations.
- Using a work area.

Printing a file involves:

- Identifying the portion to be printed.
- Printing record key or relative record numbers.

#### Disk or File Location

In order to copy a disk or file, you must specify the unit on which the disk or file is located and the unit to which it is to be copied. You can copy from one disk to another or from one area to another on the same disk (the latter applies to only part of a disk or file).

#### Using a Work Area

When you are copying a disk or file to another disk but have only one disk drive, you must use available space on the fixed disk on drive one. The disk you copy from must be a removable disk. The information from the disk you are copying is transferred to the available space on the fixed disk where it remains until another removable disk is mounted. This is the removable disk to which the information is copied.

If you are copying a file from one area on a removable disk to another area on the same disk you needn't use a work area on the fixed disk.

#### Printing a Portion of a File

You can print all or part of a file.

#### Record Keys and Relative Record Numbers

For indexed files the Disk Copy/Dump program will print each record key (used to access the record) followed by the contents of the record. The records are printed either in the order their keys appear in the index portion of the file or as they appear in the file itself. For sequential and direct files, a record is printed with its relative record number (used to access the record) preceding the record. The records are printed in the order they appear in the file.

### OPTIONS

The Disk Copy/Dump program allows you the following options:

- You may copy an entire disk or a file.
- You may print part or all of a file.
- You may delete records from a file.
- You may reorganize a file.

You specify the options you want in control statements (see *Control Statements* in this chapter).

#### Copying and Printing

You may specify any of the following copy or print combinations:

- Copy an entire disk.
- Copy a data file.
- Copy and print a data file.
- Copy a data file, but print only part of the file.
- Print an entire data file.
- Print only a part of a data file.

On a Model 6 with 8K of main storage, a halt may occur if all options on a COPYFILE are specified for files with large records (256-bytes). This halt (A234) occurs because not enough main storage is available. To avoid this halt, consider the following changes to the COPYFILE statement:

1. Specify OUTPUT-DISK instead of OUTPUT-BOTH.

2. Specify REORG-NO instead of REORG-YES.

3. Specify OMIT- instead of DELETE-.

*Copying Entire Disk*

When copying a disk, the Disk Copy/Dump program transfers the contents of the disk to another disk. The contents of the two disks will be the same, except for the disk names and alternate track information, which may be different.

The disk you are copying can contain libraries or data files or both. The disk that is to contain the copy must not have libraries, temporary data files, or permanent data files.

The program can copy the contents of one removable disk to another using one disk drive. The drive, however, must be drive 1. To do this, the program uses available space on the fixed disk on drive 1. It fills the available space with information from the disk you are copying. Then it prints a message telling the operator to mount the other removable disk (the one to contain the copy) on drive 1. After transferring the information from the fixed disk to the removable disk, the program prints another message telling the operator to remount the disk you are copying. The program repeats this procedure until all information has been transferred.

Until the contents of the disk are completely copied on the new disk, three addressing portions of the new disk are changed to prevent accidental usage of a partially filled disk. Therefore, if the copying process is stopped before it is completed, the pack is unusable. You can restart the copying process by reloading the copy program or you can restore the disk by reinitializing.

After a successful copy the copy program prints a message:

```
COPYPACK IS COMPLETE
```

*Copying Files*

The Disk Copy/Dump program can copy a file from one disk to another or from one area to another on the same disk.

Your responses to the OCL keywords prompted for the Disk Copy/Dump program indicate (1) the name and location of the file being copied and (2) the name and location of the copy being created. See *OCL Considerations* in this section.

The program can copy a file from one removable disk to another using one disk drive. The drive, however, must be drive 1. (See *WORK Parameter* in this section for more information.)

In copying a file, the program can omit records. (See *DELETE Parameter* in this section for more information.)

In copying an indexed file, the program can reorganize records in the data portion such that they are in the same order as their keys are listed in the index. (See *REORG Parameter* in this section for more information.)

*Printing Files*

The program can print all or part of the data file. To print only part, the program needs a SELECT control statement. (See *SELECT KEY and PKY Parameters and SELECT RECORD Parameters* in this section.) If you do not use a SELECT statement, the entire file is printed.

If you use SELECT or REORG, records from indexed files are printed in the order their keys appear in the index portion of the file; otherwise, they are printed as they appear in the file. For each record, the program prints the record key followed by the contents of the record.

Records from sequential and direct files are printed in the order they appear in the file. For each record, the program prints the relative record number followed by the contents of the record.

The program uses as many lines as it needs to print the contents of a record. If OUTPUT- is specified, only printable characters are printed. If OUTPTX- is specified, all characters are printed with their 2-digit hexadecimal value. Appendix J lists the hexadecimal values for characters in the standard character set.

The following is an example of the way the program prints a 20-character record when OUTPUT- is specified.

```
ABCDE GHI J 12345
```

If OUTPTX- is specified, the same record would be printed:

```
ABCDE GHI J 12345
CCCCCBCCCDFFFFF44444
1234567891 1234500000
```

After printing the last record, the printer triple spaces and prints the following message:

```
(number) RECORDS PRINTED
```

### Deleting Records

If you wish to delete records from a file while copying or printing, you must indicate the type of record you wish to omit. To do this, you must specify the identifying character (any of the standard System/3 character set except commas, apostrophes, and blanks) and the position of the character in the records (maximum position 999). The records that are deleted are printed. When the records of a file are being printed, the deleted records are indicated.

### Reorganizing a File

When you are copying an indexed file you can reorganize it. The records in the data portion are put in the same order as their index keys leaving the original of the file you are copying unaffected. If you are both copying and printing an indexed file, you must specify reorganization.

## COPYING MULTI-VOLUME FILES

When copying multi-volume files the first volume of the input file has to be online when the job is initiated. The output file must be a new file. If neither condition is satisfied a halt occurs.

### Maintaining Proper Volume Sequence Numbers

To maintain proper volume sequence numbers when copying a multi-volume file, you must either copy all the volumes of the file in one run or copy only one volume for each run of $COPY. For example, if you copy a 3-volume file one volume at a time: folume 1 in the first run, volume 2 in the second run, and volume 3 in the third run; the volumes will retain their original sequence numbers in the output file. Or if you copy all the volumes (1, 2, and 3) in the same run, the volume sequence numbers in the new file will be same as in the original file. However, if you copy only volumes 2 and 3 in one run, their volume sequence numbers will be changed to 1 and 2 in the output file.

$COPY will insure that all volumes of a multivolume file have the same date in the following manner. If only one volume of a multivolume file is copied for each run of $COPY, the new file will assume the same date as the input file. If all volumes, or as in the example above, volumes 2 and 3 of a 3-volume file, are copied in a single run, the new file will assume the current system date.

### Maintaining Correct Relative Record Numbers

To maintain correct relative record numbers when copying one volume of a multi-volume direct file, the size of the output volume must be the same size of the input volume. (If you want to increase the size of a file, you must copy the entire file.) If, for example, you copy the first volume of a 2-volume file and increase the number of records on that volume, you are also increasing relative record numbers of all the records on the next volume. Therefore, output and input volume extents must be equal if you are copying only one volume of a multi-volume direct file.

*Note:* You can not use the copy program to copy a single volume file to a multi-volume file. End of extents will probably occur after the first volume of output. If the output file is a new file, the copy program will not create it as a multi-volume file.

**Direct File Attributes**

If you copy a whole multi-volume direct file in one run, the output file will be given consecutive attributes in the Volume Table of Contents (VTOC). However, this does not affect file processing. A file with either consecutive or direct attrubutes can be accessed by a consecutive or direct access method. If only one volume is copied, the direct attribute will be maintained.

**Copying Multi-Volume Index Files**

If you want to copy a multi-volume indexed file, REORG-YES must be given. Since an unordered multi-volume indexed load is not permitted, a REORG-NO will cause a halt if an out-of-sequence record is found. If you would prefer not to reorganize the file, each volume of the file must be copied as a single volume file. When copying each volume separately, it can be either ordered or unordered. When copying one volume of a multi-volume indexed file, either REORG-YES or REORG-NO may be specified. HIKEY parameter(s) of the output file must be the same as the highest key(s) of each input volume.

## CONTROL STATEMENTS

You must supply the following control statements to specify the program options you want:

1. *COPYPACK statement*—indicates that an entire disk is to be copied. It contains the unit of the disk to be copied and the disk to which the copying is being done.

2. *COPYFILE statement*—indicates that all or part of a data file is being copied or printed or both, whether the file is to be reorganized, and whether any records are to be deleted. It also allows you to specify if you want a work area.

3. *SELECT KEY statement*—indicates, according to record keys, which part of an indexed file you want printed.

4. *SELECT RECORD statement*—indicates, according to relative record numbers, which part of a file you want printed.

5. *END statement*—indicates the end of control statements.

## Control Statement Summary

```
Uses ❶                Control Statements ❷

                      { // COPYPACK FROM-code, TO-code
Copy an Entire Disk:  {
                      { // END
                                  { OUTPTX- }       { DELETE- }                            ❸         { NO  } ❹          { NO  } ❺
                      { // COPYFILE {   -or-    } DISK, {   -or-   } 'position,character',  REORG- { -or- } , WORK- { -or- }
Copy a Data File:     {           { OUTPUT- }       { OMIT-   }                                      { YES }            { YES }
                      { // END
                                  { OUTPTX- }       { DELETE }                             ❸            ❹           { NO  } ❺
                      { // COPYFILE {   -or-    } BOTH, {   -or-   } 'position,character',  REORG-YES,  WORK- { -or- }
Copy and Print        {           { OUTPUT- }       { OMIT-  }                                                      { YES }
a Data File:          { // END
                                  { OUTPTX- }       { DELETE- }                            ❸            ❹           { NO  } ❺
                      { // COPYFILE {   -or-    } BOTH, {   -or-   } 'position,character',  REORG-YES,  WORK- { -or- }
                      {           { OUTPUT- }       { OMIT-   }                                                     { YES }
                      { // SELECT KEY,FROM-'key' ❹                       }
                      {          -or-                                     }
                      { // SELECT KEY,FROM-'key',TO-'key' ❹              }
                      {          -or-                                     }
Copy a Data File,     { // SELECT RECORD,FROM-number                      }
But Print Only a      {          -or-                                     }  One of these ❻
Part of the File:     { // SELECT RECORD,FROM-number,TO-number            }
                      {                                                   }
                      { // SELECT PKY,FROM-'key' ❼                       }
                      {          -or-                                     }
                      { // SELECT PKY,FROM-'key',TO-'key' ❼              }
                      {
                      { // END
                                  { OUTPTX- }
                      { // COPYFILE {   -or-    } PRINT
Print an Entire Data  {           { OUTPUT- }
File:                 { // END
                                    OUTPTX- }
                      { // COPYFILE   -or-    } PRINT
                      {             OUTPUT- }
                      { // SELECT KEY,FROM-'key' ❹                       }
                      {          -or-                                     }
                      { // SELECT KEY,FROM-'key',TO-'key' ❹              }
                      {          -or-                                     }
                      { // SELECT RECORD,FROM-number                      }  One of these ❻
Print Only a Part     {          -or-                                     }
of a Data File:       { // SELECT RECORD,FROM-number,TO-number            }
                      {                                                   }
                      { // SELECT PKY,FROM-'key' ❼                       }
                      {          -or-                                     }
                      { // SELECT PKY,FROM-'key',TO-'key' ❼              }
                      {
                      { // END
```

❶ The program uses include the possible combinations of copying and printing files.

❷ For each use, the program requires the control statements in the order they are listed: COPYPACK, END; COPYFILE, END; and COPYFILE,SELECT,END.

❸ Needed only if you want to delete a certain type of record. DELETE cannot be used with direct files.

❹ Applies only to indexed files. When OUTPUT-BOTH is specified, REORG-YES is required.

❺ WORK-YES applies if you are copying the file from one removable disk to another using the same disk drive (drive one). WORK-NO applies if you are copying the file from one area to another on the removable disk on drive one.

❻ Identifies the portion you want to print.

❼ Index files with packed keys.

## Parameter Summary

| COPYPACK Statement Parameters | Meaning |
|---|---|
| FROM-code | Location of disk to be copied. Possible codes are R1, F1, R2, F2, D1, D2. |
| TO-code | Location of disk to contain the copy. Possible codes are R1, F1, R2, F2, D1, D2. |
| **COPYFILE Statement Parameters** | |
| OUTPUT-DISK | Copy the file from one disk to another, or from one area to another on the same disk. ❶ |
| OUTPUT-PRINT | Print the entire file or only part of the file. ❶ |
| OUTPUT-BOTH | Copy the file from one disk to another, or from one area to another on the same disk. ❶ Also print the entire file or only part of it. |
| OUTPTX-$\left\{\begin{array}{l}\text{DISK}\\\text{PRINT}\\\text{BOTH}\end{array}\right\}$ | Printed output will be displayed in hexadecimal values. |
| DELETE-'position,character' -or- OMIT-'position,character' | These parameters are optional. It means that all records with the specified character in the specified record position are deleted. DELETE causes deleted records to be printed. OMIT causes deleted records not be printed. Position can be any position in the record (the first position is 1, second 2, and so on). The maximum position is 9999. |
| REORG-NO ❷ | Indexed files only. Copy records in the same way as they are organized in the original file (the file from which the records are copied). |
| REORG-YES ❷ | Indexed files only. Re-organize the records so that the records in the data portion of the file are in the same order as their keys are listed in the index. |
| WORK-NO ❸ | Required for copying a file from one area to another on a removable disk on drive one (R1 or D1). It means: do not use a work area. |
| WORK-YES ❸ | Required for copying a file from one removable disk on drive one to another removable disk on that drive. It means: use a work area on the fixed disk on drive one or on the removable disk on drive one if the file being copied is on the 5445. R1 must have a minimum of 198 contiguous unused tracks. |

| SELECT Statement Parameters | Meaning |
|---|---|
| $\left\{\begin{array}{l}\text{KEY}\\\text{PKY}\end{array}\right\}$,FROM-'key' | Indexed files only. Print only the part of the file from the record key that is specified in the FROM parameter to the end of the file. |
| $\left\{\begin{array}{l}\text{KEY}\\\text{PKY}\end{array}\right\}$,FROM-'key',TO-'key' | Indexed files only. Print only the part of the file between the two record keys that are specified in the FROM and TO parameters (including the records indicated by the parameters). To print only one record, make the FROM and TO record keys the same. |
| RECORD,FROM-number | Print only the part of the file from the relative record number specified in the FROM parameter to the end of the file. |
| RECORD,FROM-number, TO-number | Print only the part of the file between the relative record numbers indicated by the parameters (including the records indicated by the parameter). To print only one record, the FROM and TO record keys should be the same. |

❶ In the OCL load sequence, the operator indicates which file is to be copied or printed. For files being copied, he must also indicate whether the file is being copied from one disk to another or from one location to another on the same disk.

❷ REORG-NO is assumed if you omit the REORG parameter. When OUTPUT-BOTH is used for indexed files, REORG-YES is required.

❸ WORK-NO is assumed if you omit the WORK parameter.

## PARAMETER DESCRIPTIONS

### FROM and TO Parameters (COPYPACK Statement)

The COPYPACK statement is used to copy the contents of one disk to another. It has two parameters: FROM and TO. They tell the program the locations of the two disks on the disk units.

The FROM parameter (FROM-code) indicates the location of the disk you are copying. The TO parameter (TO-code) indicates the location of the disk that is to contain the copy.

Codes for the possible locations are as follows:

| Code | Location |
|---|---|
| R1 | Removable disk on drive 1. |
| F1 | Fixed disk on drive 1. |
| R2 | Removable disk on drive 2. |
| F2 | Fixed disk on drive 2. |

### OUTPUT Parameter (COPYFILE Statement)

The OUTPUT parameter is used when copying and printing data files. It indicates whether you want the program to copy, print, or copy and print a file.

The parameter OUTPUT-DISK means to copy the file; OUTPUT-PRINT means to print the file; and OUTPUT-BOTH means to copy and print the file.

OUTPTX can be used instead of OUTPUT to display the printed output with its hexadecimal values.

### DELETE Parameter (COPYFILE Statement)

In copying a data file, the Disk Copy/Dump program can omit records of one type. The DELETE parameter identifies the type of records. Use of the DELETE parameter is optional. If you do not use it, no records are deleted.

The form of the parameter is DELETE-'position, character'. *Position* is the position of the character in the record. It can be any position in the record (the first position is 1, the second 2, and so on) up to the maximum position of 9999. *Character* is the character, except for apostrophes, blanks, or commas, that identifies the record. For example, with the parameter DELETE-'100,R', all records with an R in position 100 are deleted. By specifying the hexadecimal code for the character, any character (including apostrophes, blanks, commas, and packed data) can be used to identify the record to be deleted. For example, with the parameter DELETE-'100,X40', all records with a blank (hexadecimal 40) in position 100 are deleted.

Deleted records are always printed. If you are both copying and printing a data file, deleted records are printed with the other records that are printed. The deleted records are preceded by the word DELETED.

The OMIT keyword can be used instead of DELETE. The deleted records are not printed if OMIT is used.

### REORG (Reorganize) Parameter (COPYFILE Statement)

In copying an indexed file, the program can reorganize the file, such that the records in the data portion are in the same order as their keys in the file index. The REORG parameter tells the program whether or not to reorganize the file.

REORG-YES means to reorganize. REORG-NO means not to reorganize. REORG-NO is assumed if you omit the keyword.

If you tell the program to reorganize the file, the reorganization applies to the copy of the file rather than the original file. The original file is not affected.

Reorganization (REORG-YES) is required any time you are both copying and printing an indexed file (OUTPUT-BOTH).

### WORK Parameter (COPYFILE Statement)

The WORK parameter applies to copying a data file from one removable disk to another using the same disk drive (drive 1). It tells the program whether or not to use a work area on the fixed disk on drive 1.

The parameter WORK-YES means to use a work area. WORK-NO means not to use a work area.

#### *Work Area*

If you have only one disk drive, a common use of the Disk Copy/Dump program might be to copy a file from one removable disk to another. To do this, the program must use a work area on the fixed disk. The output file must be a new file.

In copying the file, the program fills the work area with records from the file you are copying. Then it prints a message telling the operator to mount the other removable disk (the one to contain the copy) on drive 1. After transferring the records from the work area to the removable disk, the program prints another message telling the operator to remount the disk containing the file you are copying. The program repeats this procedure until all records have been transferred.

If you have two disk drives, you can also use the same drive to copy a file from one removable disk to another. The drive, however, must be drive 1.

You can copy a file from one area to another on the same disk. If you do, and the disk is a removable disk that you plan to mount on drive 1, use the WORK-NO parameter (WORK-NO is assumed if the WORK keyword is not used). This keeps the program from using a work area on the fixed disk when it transfers the file from one area to the other.

When using WORK-YES, the input and output files must have different labels, locations, or pack names. It is a good practice to have different pack names on all packs in an installation.

### SELECT KEY and PKY Parameters (SELECT Statement)

The SELECT KEY and SELECT PKY parameters apply to printing part of an indexed file. The parameters are FROM and TO.

The FROM parameter (FROM-'key') gives the key of the first record to be printed. The TO parameter (TO-'key') gives the key of the last record to be printed. The record key between those two in the file index ıdentify the remaining records to be printed. If you want to print only one record, use the same record key in both the FROM and TO parameters

For example, the parameters FROM-'000100' and TO-'000199' mean that records identified by keys 000100 through 000199 are to be printed.

If the file index does not contain the key you indicate in a FROM parameter, the program uses the next higher key in the index.

You can omit the TO parameter. If you do, the program assumes that the last key in the index is the TO key.

With the SELECT KEY parameter (but not PKY) you can use less characters in the FROM or TO parameter than are contained in the actual keys. If you do, the program ignores the remaining characters in the key. The number of characters used in the FROM and TO parameters need not be the same.

For example, assume that the following are consecutive record keys in an index: 99999, A1000, A1119, A1275, A1900, A1995, and A2075. The parameters FROM-'A1' and TO-'A199' refer to record keys A1000 through A1995.

If none of the keys in the file index begin with the characters you indicate in a FROM parameter, the program uses the key beginning with the next higher characters.

For example, assume that four consecutive record keys in an index begin with these characters: A1,A2,A8, and B1. The parameters FROM-'A3' and TO-'A9' would refer to the key beginning with the character A8.

### SELECT RECORD Parameters (SELECT Statement)

The SELECT RECORD parameters can apply to any file, but are normally used for sequential and direct files. These parameters use relative record numbers to identify the records to be printed.

Relative record numbers identify a record's location with respect to other records in the file. The relative record number of the first record is 1, the number of the second record is 2, and so on.

The SELECT RECORD parameters are FROM and TO. The FROM parameter (FROM-number) gives the relative record number of the first record to be printed. The TO parameter (TO-number) gives the number of the last record to be printed. Records between those two records in the file are also printed. If you want to print only one record, use the same record number in the FROM and TO parameters.

For example, the parameters FROM-1 and TO-30 mean that the first thirty records (1-30) in the file will be printed.

You can omit the TO parameter. If you do, the program assumes that the number of the last record in the file is the TO number.

## OCL CONSIDERATIONS

### LOAD Sequence for Copying an Entire Disk

| Keywords ❶ | Responses ❷ | Considerations |
|---|---|---|
| READY | LOAD | None |
| LOAD NAME | $COPY | Name of Disk Copy/Dump program. |
| UNIT | R1, R2, F1, or F2 | Location of disk containing Disk Copy/Dump program. |
| MODIFY | RUN | None |

❶ Only the keywords listed here are required. You can bypass the rest.

❷ You end every response by pressing PROG START.

### BUILD Sequence for Copying an Entire Disk

| Keywords ❶ | Responses ❷ | Considerations |
|---|---|---|
| READY | BUILD | None |
| BUILD NAME | procedure name | Name by which procedure will be identified in source library. |
| UNIT | R1, R2, F1, or F2 | Location of disk containing source library. |
| LOAD NAME | $COPY | Name of Disk Copy/Dump program. |
| UNIT | R1, R2, F1, or F2 | Location of disk containing Disk Copy/Dump program. |
| MODIFY | INCLUDE utility control statements RUN | Response when including control statements in procedure. |
|  | OR RUN | Response when not including control statements in procedure |

❶ Only the keywords listed here are required. You can bypass the rest.

❷ You end every response by pressing PROG START.

## LOAD Sequence for Copying or Printing Files

| Keywords [1] | Responses [2] | Considerations |
|---|---|---|
| READY | LOAD | None |
| LOAD NAME | $COPY | Name of Disk Copy/Dump program. |
| UNIT | R1, R2, F1, or F2 | Location of disk containing Disk Copy/Dump program. |
| FILE NAME | COPYIN | Name Disk Copy/Dump program uses to refer to file to be copied (input file). |
| UNIT | R1, R2, F1, or F2 | Location of disk containing file to be copied. |
| PACK | disk name | Name of disk containing file to be copied. |
| LABEL | file name | Name by which file to be copied is identified on disk. |
| FILE NAME | COPYO | Name Disk Copy/Dump program uses to refer to output file being created. |
|  | OR |  |
|  | Press PROG START | If you are only printing records from a file, press PROG START instead of typing COPYO. The next keyword prompted will be MODIFY. |
| UNIT | R1, R2, F1, or F2 | Location of disk on which output file is to be created. |
| PACK | disk name | Name of disk on which output file is to be identified on disk. |
| LABEL | file name | Name by which output file is to be identified on disk. |
| RECORDS or TRACKS | number | Size of output file expressed either as number of records (RECORDS) or number of disk tracks (TRACKS). |
| RETAIN | T, P, or S | Designation (temporary, permanent, or scratch) of output file. |
| MODIFY | RUN | None |

[1] Only the keywords listed here are required. You can bypass the rest.

[2] You end every response by pressing PROG START.

**BUILD Sequence for Copying or Printing Files**

| Keywords [1] | Responses [2] | Considerations |
|---|---|---|
| READY | BUILD | None |
| BUILD NAME | procedure name | Name by which procedure will be identified in source library. |
| UNIT | R1, R2, F1, or F2 | Location of disk containing source library. |
| LOAD NAME | $COPY | Name of Disk Copy/Dump program. |
| UNIT | R1, R2, F1, or F2 | Location of disk containing Disk Copy/Dump program. |
| FILE NAME | COPYIN | Name Disk Copy/Dump program uses to refer to file to be copied (input file). |
| UNIT | R1, F1, R2, or F2 | Location of disk containing file to be copied. |
| PACK | disk name | Name of disk containing file to be copied. |
| LABEL | file name | Name by which file to be copied is identified on disk. |
| FILE NAME | COPYO | Name Disk Copy/Dump program uses to refer to output file being created. |
| | OR | |
| | Press PROG START | If you are only printing records from a file, press PROG START instead of typing COPYO. The next keyword prompted will be MODIFY. |
| UNIT | R1, R2, F1, or F2 | Location of disk on which output file is to be created. |
| PACK | disk name | Name of disk on which output file is to be created. |
| LABEL | file name | Name by which output file is to be identified on disk. |
| RECORDS or TRACKS | number | Size of output file expressed either as number of records (RECORDS) or number of disk tracks (TRACKS). |
| RETAIN | T, P, or S | Designation (temporary, permanent, or scratch) of output file. |
| MODIFY | INCLUDE<br>utility control statements<br>RUN | Response when including control statements in procedure. |
| | OR | |
| | RUN | Response when not including control statements in procedure. |

[1] Only the keywords listed here are required. You can bypass the rest.

[2] You end every response by pressing PROG START.

## EXAMPLES

### Copying an Entire Disk

*Statements*

```
READY                        - LOAD
************************
010     LOAD    NAME         - $COPY
011             UNIT         - F1
020     DATE    (XX/XX/XX)   -
030     SWITCH  (00000000)   -
040     FILE    NAME         -
************************
MODIFY
RUN
```

**OCL LOAD Sequence.**
Boxed areas are operator responses.

Keywords for which no responses are shown are the ones bypassed. If you press ENTER— after responding to UNIT, the DATE, SWITCH, and FILE NAME keywords are not prompted.

RUN is the response to MODIFY even though the two words do not appear on the same line.

```
   ENTER '//' CONTROL STATEMENT    }   Message printed by Disk Copy/Dump program.
// COPYPACK FROM-F2,TO-R2          }   Control statement supplied by operator.
   ENTER '//' CONTROL STATEMENT    {   System reprompts. END statement
// END                             {   terminates sequence.
COPYPACK IS COMPLETE               }   Message printed by Disk Copy/Dump program
                                       to indicate successful copy.
```

*Explanation*

- The Disk Copy/Dump program is loaded from the fixed disk on drive 1 (UNIT-F1 in OCL sequence).
- The contents of the fixed disk on drive 2 (FROM-F2 in COPYPACK statement) is copied onto the removable disk on drive (TO-R2).

## Copying a File From One Disk to Another

*Statements*

```
READY                          - LOAD
*************************
010    LOAD    NAME            - $COPY
011            UNIT            - F1
020    DATE                    -
030    SWITCH                  -
040    FILE    NAME            - COPYIN
041            UNIT            - F1
042            PACK            - A1
043            LABEL           - MASTER
050    FILE    NAME            - COPYO
051            UNIT            - R1
052            PACK            - B2
053            LABEL           - BACKUP
054            RECORDS         -
055            TRACKS          - 50
056            LOCATION        -
057            RETAIN          - P
*************************
MODIFY
RUN
```

**File to be copied (input file)** (040–043)

**File being created (output file)** (050–057)

**OCL LOAD Sequence.**

Boxed areas are operator responses.

Keywords for which no responses are shown are the ones bypassed.

RUN is the response to MODIFY even though the two words do not appear on the same line.

```
   ENTER '//' CONTROL STATEMENT
// COPYFILE OUTPUT-DISK
// END
```

ENTER '//' CONTROL STATEMENT: Message printed by Disk Copy/ Dump program.

// COPYFILE OUTPUT-DISK: Control statement supplied by operator.

// END: System reprompts. END statement terminates sequence.

*Explanation*

- Disk Copy/Dump program is loaded from fixed disk on drive 1 (UNIT-F1 in OCL sequence).
- Input file (OCL sequence):
  1. Name that identifies file on disk is MASTER (LABEL-MASTER).
  2. Disk that contains the file is the fixed disk on drive 1 (UNIT-F1). Its name is A1 (PACK-A1).
- Output file (OCL sequence):
  1. Name to be written on disk to identify the file is BACKUP (LABEL-BACKUP).
  2. Disk that is to contain the file is the removable disk on drive 1 (UNIT-R1). Its name is B2 (PACK-B2).
  3. The file is to be permanent (RETAIN-P).
  4. The length of the file is 50 tracks (TRACK-50).
- The COPYFILE statement tells the program to create the output file using all the data from the input file. The output file is a copy of the input file.

**Printing Part of a File**

*Statement*

```
READY                      - LOAD
**********************
010    LOAD     NAME       - $COPY
011             UNIT       - F1
020    DATE                -
030    SWITCH              -
040    FILE     NAME       - COPYIN  \
041             UNIT       - R1       |
042             PACK       - B2       |  Input file.
043             LABEL      - BACKUP  /
050    FILE     NAME       -
*********************
MODIFY                     -
RUN
```

OCL LOAD Sequence.

Boxed areas are operator responses.

Keywords for which no responses are shown are the ones bypassed.

RUN is the response to MODIFY even though the two words do not appear on the same line.

```
   ENTER '//' CONTROL STATEMENT
// COPYFILE OUTPUT-PRINT
   ENTER '//' CONTROL STATEMENT
// SELECT KEY,FROM-'ADAMS',TO-'BAKER'
   ENTER '//' CONTROL STATEMENT
// END
```

Message printed by Disk Copy/Dump program.

Control statement supplied by operator.

Sequence repeats until operator enters END statement.

*Explanation*

- Disk Copy/Dump program is loaded from the fixed disk on drive 1 (UNIT-F1 in OCL sequence).
- Input file (OCL sequence):
  1. Name that identifies the file on disk is BACKUP (LABEL-BACKUP).
  2. Disk that contains the file is the removable disk on drive 1 (UNIT-R1). Its name is B2 (PACK-B2).
- The file is being printed (COPYFILE statement).
- The file is an indexed file. The part being printed is identified by the record keys from ADAMS to BAKER in the index (SELECT statement).

Your programs are stored on disk in an area called a *library*. You can update or add new entries in this library. In order to do so, you must use the Library Maintenance program.

The Library Maintenance program ($MAINT) has five functions:

| Function | Meaning |
|---|---|
| Allocate | Create (reserve space for), delete, reorganize, and change the sizes of libraries. |
| Copy | Place entries in, and display the contents of, libraries. |
| Delete | Delete library entries. |
| Modify | Modify source library entries. |
| Rename | Change the names of library entries. |

The control statements you must supply depend on the function you are using.

**Library Description**

*Source Library*

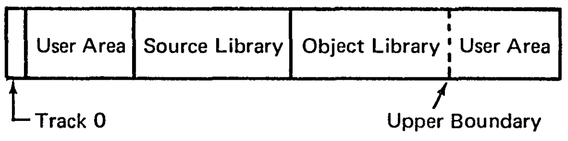
The source library is an area on disk for storing procedures and source statements. Procedures are groups of OCL statements used to load programs. The statements can be followed by input data for the programs. (Procedures for utility programs can, for example, contain utility control statements.) Source statements are sets of data, the most common of which are RPG II source programs and Disk Sort sequence specifications.

*Object Library*

The object library is an area on disk for storing object programs and routines. Object programs are programs and subroutines in such a form that they can be loaded for execution. (They are sometimes called load modules.) Routines are programs and subroutines that need to be link-edited into object programs before they can be loaded for execution. (They are sometimes called object modules.)

*Location of Libraries on Disk*

Libraries can be located anywhere on disk. However, the location of a source library with respect to an object library is always the same:



The boundaries of a source library are fixed. They can be changed only by the allocate function of the Library Maintenance program. The upper boundary of an object library, however, can be moved as additional space is needed when entries are placed in the library. This happens only if space is available following the library and if the entries being placed beyond the normal boundary are not permanent entries.

*Organization of Library Entries*

Entries are stored in the object library serially; that is, a 20-sector program occupies 20 consecutive sectors. Temporary entries follow all permanent entries in the object library.

If necessary, the upper boundary is changed to allow more space for temporary entries. The upper boundary of the library is extended to the end of the pack or to the first temporary or permanent file, allowing the maximum amount of space for the temporary library entry. At the successful completion of the copy, the upper boundary is returned to its original position or to the end of the last temporary entry. If the copy was not completed successfully, the upper boundary may remain extended. When a permanent entry is placed in the library or the library is reorganized, all temporary entries are deleted and the upper boundary returns to its original location. Permanent entries cannot exceed the original upper boundary.

Gaps can occur in the object library when an entry is deleted. The associated directory entries will point to these gaps. When the Library Maintenance program places a new entry in the library, it searches the directory for a gap that has the same number of sectors, or the fewest number of sectors over the number required by the new entry. If the entry is smaller than the gap, the last part of the gap will not be pointed to by a directory entry. Since this gap has no directory entry, it will not be used until the library is reorganized.

If the number of unusable sectors become excessive, the library should be reorganized. In reorganizing entries, the Library Maintenance program deletes temporary entries and shifts permanent entries so that gaps do not appear between them. This makes more sectors available for use.

The source library differs from the object library in that entries within the source library need not be stored in consecutive sectors. An entry can be stored in many widely separated sectors with each sector pointing to the sector that contains the next part of the entry. When an entry is placed in the source library, it is placed in as many sectors as required regardless of where the sectors are located within the library.

The boundary of the source library cannot be expanded; therefore, an entry must fit within the available library space. To provide as much space as possible within the prescribed limits of the source library, the system compresses entries. That is, blanks and duplicate characters are removed from entries. Later, if the entries are printed or punched, the blanks and duplicate characters are reinserted. When the size of the source library is changed or the source library is reorganized, all temporary entries are deleted.

*Library Directories*

The program creates a separate directory for each library. Every library entry has a corresponding entry in its library directory. The directory entry contains such information as the name and location of the library entry. The program also creates a system directory, which contains information about the size and available space in libraries and their directories.

**Organization of This Section**

The five functions are described separately. Every description contains the following:

1. List of specific uses.
2. Control statement summary indicating the form of the control statement needed for each use.
3. Parameter descriptions explaining, in detail, the contents and meanings of the parameters.
4. Function descriptions explaining the details of each function.
5. Examples that include OCL statements, utility control statements, and explanations of their use.

OCL considerations for the program precede the examples.

## ALLOCATE

The allocate function of the Library Maintenance program allows you to:

- Create libraries.
- Change the size of libraries.
- Delete libraries.
- Reorganize libraries.

**Creating Libraries**

Creating a library involves:

- Assigning a library to a disk.
- Assigning space for the library directory.
- Using a work area.

*Assigning a Library to a Disk:* You are allowed one source and one object library per disk. The libraries can be located anywhere on the disk where space is made available as long as the source library precedes the object library. You needn't have both libraries for a disk.

*Assigning Space for the Library Directory:* The Library Maintenance program creates a separate directory for each library. *A directory for a source or object library* contains information concerning each library entry. This information includes the name and location of the library entry. For a source library, the first two sectors of the first track are assigned to the directory. For an object library which includes system programs, the first three tracks are assigned to the directory. If system programs are not included, only the first track is assigned to the directory. The directory size is overridden by the DIRSIZE parameter if used (see *DIRSIZE*).

Another type of directory, the system directory, is also created by this program. The *system directory* contains information concerning the libraries and their directories. This information includes the size of and available space in the libraries and their directories. The system directory is contained in the volume label on any disk pack.

**Library Maintenance Allocate Restrictions**

This program has restrictions and operating conditions that the user must be aware of when maintaining libraries.

*Allocation of Disk Space*

The Library Maintenance program allocates disk space for each of the following functions:

- Allocate a library
- Increase the size of a library
- Reorganize a library
- Dynamically extend an object library to copy temporary entries to the library
- Sort a directory before it is printed

The space allocated by the program is the first contiguous space large enough for the function to be performed. The Library Maintenance program will use as much space as is available to the end of the pack or to the first temporary or permanent data file, removing all scratch files in this area. If within a single load of the program, there are functions performed which require more than four disk areas to be allocated, a halt will occur. The Library Maintenance program must be reloaded to continue.

*Removing Temporary Entries*

When a library is reorganized, its size is changed, or it is moved, all temporary entries in that library are deleted. This applies to both the source and object libraries.

*Library Restrictions*

The Allocate function cannot reference the libraries on the pack from which the Library Maintenance Program or the system was loaded. For example, if the system was loaded (IPL) from F1 and the Library Maintenance Program was loaded from R1, the source or object libraries on F1 and R1 cannot be referenced on an ALLOCATE statement.

*Moving the Object Library*

When allocating or reallocating the source library on a pack that contains an object library, the object library is reorganized and all temporary entries are deleted.

## Allocate Control Statement Summary

$$\text{// ALLOCATE TO-code,SOURCE-}\left\{\begin{matrix}\text{number}\\ \text{R}\end{matrix}\right\}\text{,OBJECT-}\left\{\begin{matrix}\text{number}\\ \text{R}\end{matrix}\right\}\text{,SYSTEM-}\left\{\begin{matrix}\text{NO}\\ \text{YES}\end{matrix}\right\}\text{,DIRSIZE-number,WORK-code}$$

| | **Use** ❶ | **Parameter Needed** ❷ |
|---|---|---|
| Source Library | Create: | TO-code,SOURCE-number,WORK-code ❸ |
| | Change Size: | TO-code,SOURCE-number,WORK-code |
| | Delete: | TO-code,SOURCE-0 |
| | Reorganize: | TO-code,SOURCE-R,WORK-code |
| Object Library | Create: | TO-code,OBJECT-number,SYSTEM- $\left\{\begin{matrix}\text{NO}\\ \text{YES}\end{matrix}\right\}$ |
| | Change Size: | TO-code,OBJECT-number,WORK-code ❹ |
| | Delete: | TO-code,OBJECT-0 |
| | Reorganize: | TO-code,OBJECT-R,WORK-code ❹ |

❶ You can indicate a source library use, any object library use, or uses involving both libraries (for example, deleting the source library and changing the size of the object library).

❷ If you are indicating uses for both libraries, use only one TO parameter. (The libraries must be on the same disk.) Also, use only one WORK parameter if both uses require a WORK parameter.

❸ The WORK parameter is needed only if the disk contains an object library that you are not deleting.

❹ The WORK parameter is needed only if other functions are also being performed.

## Allocate Parameter Summary

| Parameter | Meaning | |
|---|---|---|
| TO-code | Location of disk you are using. Possible codes are R1, F1, R2, and F2 | |
| SOURCE-number (no source library on disk) | Create a source library. Number indicates the number of tracks you want to assign. | |
| SOURCE-number (source library already on disk) | Delete or change the size of the source library. Use depends on number: | |
| | *Number* | *Use* |
| | 0 | Delete |
| | Any number but zero | Change size |
| SOURCE-R | Reorganize the source library. | |
| OBJECT-number (no object library on disk) | Create an object library. Number indicates the number of tracks you want to assign. | |
| OBJECT-number (object library already on disk) | Delete or change the size of the object library. Use depends on number: | |
| | *Number* | *Use* |
| | 0 | Delete |
| | Any number but zero | Change size |
| OBJECT-R | Reorganize the object library. | |
| DIRSIZE-number | Number of tracks you want for the directory when creating, reallocating, or reorganizing the object library. | |
| SYSTEM-NO | Do not create a scheduler work area. This will be a program pack. | |
| SYSTEM-YES | Create a scheduler work area. This will be a system pack. | |
| WORK-code | Location of disk containing space the program can use as a work area. Possible codes are R1, F1, R2, or F2. | |

## PARAMETER DESCRIPTIONS

### TO Parameter

The TO parameter (TO-code) indicates the location of the disk that contains, or will contain, the library. If the program use involves both libraries, the libraries must be on the same disk. The TO parameter cannot be the same unit from which the Library Maintenance program or the system was loaded.

Codes for the possible locations are as follows:

| Code | Location |
|---|---|
| R1 | Removable disk on drive 1. |
| F1 | Fixed disk on drive 1. |
| R2 | Removable disk on drive 2. |
| F2 | Fixed disk on drive 2. |

### SOURCE and OBJECT Parameters

The SOURCE and OBJECT parameters identify library uses:

| Parameter | Use |
|---|---|
| SOURCE-number OBJECT-number (number is not zero) | If the disk contains no library, this parameter means create a library. Number is the number of tracks you want to assign to the library. If the disk contains a library, this parameter means change the library size. Number is the number of tracks you want to assign to the library. |
| SOURCE-0 } OBJECT-0 } | Delete the library. |
| SOURCE-R } OBJECT-R } | Reorganize the library. |

### DIRSIZE Parameter

The DIRSIZE parameter allows the user to specify the size of the object library directory. The number of tracks specified (1-9) overrides the SYSTEM parameter in determining directory size. Each track can contain 288 directory entries. One entry is needed for the directory, so the formula for the number of entries in a directory is (t x 288)-1, where t is the number of tracks. If the DIRSIZE parameter is omitted, the SYSTEM parameter determines the directory size.

### SYSTEM Parameter

The SYSTEM parameter applies when creating, changing the size of, and reorganizing object libraries. It tells the program whether you intend to include system programs in the library. If system programs are to be included, a scheduler work area must be assigned, and the directory must be large enough for all those system programs necessary for program loading and running (minimum system), and those necessary for generating and maintaining a system.

Space for the scheduler work area is assigned immediately preceding the object library. If the disk contains a source library, the work area is between the source and object libraries. For information about the size of the scheduler work area, see *Creating an Object Library* under *Using the Allocate Function.*

The following charts show the results of coding the SYSTEM parameter for different allocate uses.

*Creating an Object Library*

| Parameter | Scheduler Work Area | Directory Size* |
|---|---|---|
| SYSTEM-YES | Created | Three Tracks |
| SYSTEM-NO | Not Created | One Track |
| not coded | Not Created | One Track |

* The directory size is overridden if the DIRSIZE parameter is used.

*Changing the Size of or Reorganizing an Object Library That Contains System Programs*

| Parameter | Scheduler Work Area | Directory Size* |
|---|---|---|
| SYSTEM-YES | Retained | not changed |
| SYSTEM-NO | Removed | not changed |
| not coded | Retained | not changed |

*The directory size is overridden if the DIRSIZE parameter is coded.

*Changing the Size of or Reorganizing an Object Library That Does Not Contain System Programs*

| Parameter | Scheduler Work Area | Directory Size* |
|---|---|---|
| SYSTEM-YES | Created | not changed |
| SYSTEM-NO | Not Created | not changed |
| not coded | Not Created | not changed |

*The directory size is overridden if the DIRSIZE parameter is coded.

### WORK Parameter

The WORK parameter (WORK-code) indicates the location of the disk that contains a work area. Library entries are temporarily stored in the work area while the program moves and reorganizes libraries.

Codes for the possible disk locations are as follows:

| Code | Location |
|---|---|
| R1 | Removable disk on drive 1. |
| F1 | Fixed disk on drive 1. |
| R2 | Removable disk on drive 2. |
| F2 | Fixed disk on drive 2. |

When the WORK parameter is coded on an ALLOCATE statement, an additional allocation of disk space may result. (See index entry *Allocation of Disk Space.)*

*Size of the Work Area:* The work area must be large enough to hold the permanent entries of the source library, object library, or both libraries depending on the program use. If you are combining uses, such as changing the sizes of both libraries, the work area must be large enough to hold the contents of both libraries.

| Use | Contents of Work Area |
|---|---|
| Create a source library (disk contains an object library). | Object library. |
| Change source library size (disk contains an object library). | Source library and object library. |
| Change source library size (disk doesn't contain an object library). | Source library. |
| Reorganize source library. | Source library. |
| Change object library size. | Object library (if not compress in place see *Compress in Place).* |
| Reorganize object library. | Object library (if not compress in place). |

**Compress in Place (OBJECT-$\left\{\begin{matrix}\textbf{R}\\ \textbf{Number}\end{matrix}\right\}$)**

If the object library is being reorganized or the size of the object library is being changed and no other functions are being performed, the object library is compressed in place. This means that the library is reorganized with all gaps removed and all temporary entries are deleted without using a work area. The WORK parameter is ignored if it is supplies.

A work area is needed if a source library function is being performed, the directory size (DIRSIZE parameter) changed, or the pack type (SYSTEM parameter) changed in conjuoction with an object library function.

Compress in place allows the user with a single-spindle or half capacity system to reorganize the object library.

*Location of Work Area on Disk:* The program uses the first available disk area large enough to hold the library, or libraries.

*Location of Disk Containing the Work Area:* The work area can be on either disk on either drive. However, it cannot be the same disk as the one you specified in the TO parameter. The only requirement is that the disk must have an available area large enough for the work area. If your system has two disk drives, the program works faster if the disk containing the libraries is on a different drive than the disk containing the work area.

**Using the Allocate Function**

*Creating a Source Library (SOURCE-number)*

*Source Library Size*

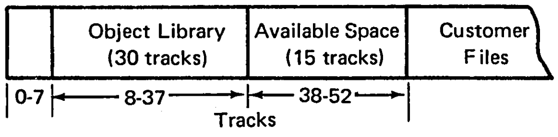
- Minimum: One track
- Maximum: Number of tracks in the available area
- Regardless of the number of tracks you specify, the first two sectors of the first track are assigned to the library directory. Additional sectors are used as needed for the directory.

*Placement of Source Library (Disk With an Object Library)*

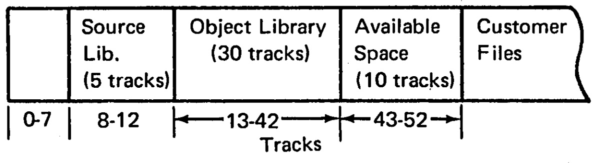
- The source library must immediately precede the object library. A disk area large enough for the source library must follow the object library because the program moves the object library to make room for the source library. To do this, the program needs a work area. (See *WORK Parameter.)* The object library is reorganized and all temporary entries are deleted.
- If you allocate a source library after deleting it, the program automatically moves the object library to make room for the source library. The starting location of the source library is the previous starting location of the object library.

*Placement of the Source Library (Disk Without an Object Library):* The program assigns the source library to the first available disk area large enough for the library. If you allocate a source library after deleting it, the source library is assigned the same way.

Disk Space Before Source Library:



Disk Space After Source Library:



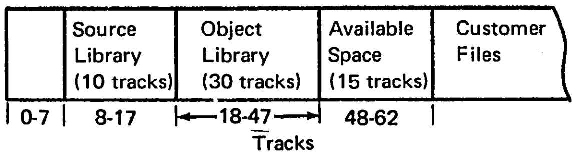
*Changing the Size of a Source Library*

Any time the program changes the source library size, it reorganizes both the source and object libraries and deletes all temporary entries. (See *Reorganizing a Source Library.*) To do this, it needs a work area. (See *WORK Parameter.*)

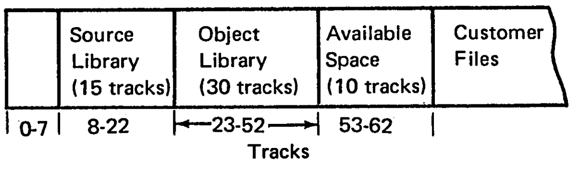
*Making the Source Library Larger*

- If the disk contains an object library, space must be available immediately following the object library. The program moves the object library to make tracks available at the end of the source library.
- If the disk does not contain an object library, space must be available immediately following the source library.

Disk Before Tracks Are Added to Source Library:


Source Library (10 tracks)
Object Library (30 tracks)
Available Space (15 tracks)
Customer Files
0-7
8-17
18-47
48-62
Tracks

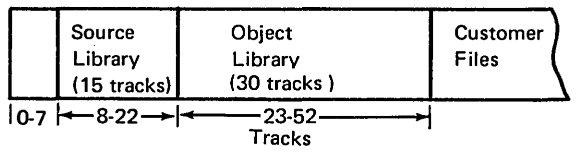
Disk After Five Tracks Are Added to Source Library:


Source Library (15 tracks)
Object Library (30 tracks)
Available Space (10 tracks)
Customer Files
0-7
8-22
23-52
53-62
Tracks

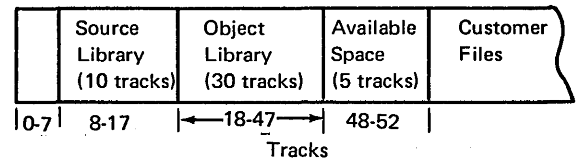
*Making the Source Library Smaller*

- If the disk contains an object library, the program moves the end location of the source library to make the library smaller. The object library is moved and space becomes available following the object library.
- If the disk does not contain an object library, the program moves the end location of the source library to make the source library smaller.

Disk Before Source-Library Size Was Decreased:



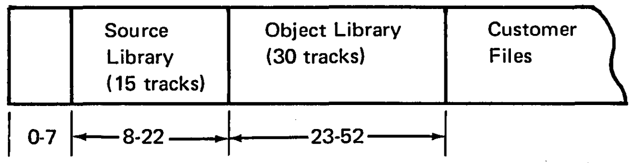
Disk After Five Tracks Were Taken From Source Library:


Source Library (10 tracks)
Object Library (30 tracks)
Available Space (5 tracks)
Customer Files
0-7
8-17
18-47
48-52
Tracks

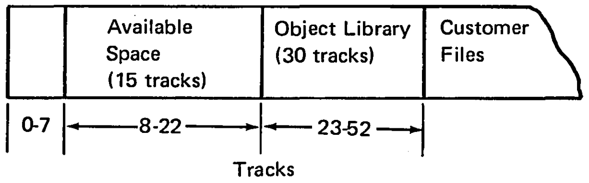
*Deleting a Source Library (SOURCE-0)*

The program makes the disk area occupied by the source library available for other use (disk files).

Disk Before Source Library Deleted



Disk After Source Library Deleted



*Reorganizing a Source Library (SOURCE-R)*

*Reason for Reorganizing the Library:* Areas from which source library entries are deleted are completely reused for new entries. If an entry exceeds the space in such an area, the program puts as much of the entry as will fit in the area and continues the entry in the next available area. In this way, the program efficiently uses library space. This can, however, decrease the speed at which those entries can be read from the library. Therefore, if you frequently add and delete source library entries, you should reorganize your source library periodically.

*Reorganizing the Library:* The program relocates entries so that no entry is started in one area and continued in another. All temporary entries are deleted. The program needs a work area. (See *WORK Parameter.*)

*Creating an Object Library (OBJECT-number)*

*Object Library Size*

- Minimum: Three tracks, including the directory tracks.
- Maximum: Number of tracks in available area.
- Library Directory: The first three tracks in the library are reserved for the library directory if the library is to contain system programs; otherwise, only the first track is used. If the DIRSIZE parameter is entered, the directory size specified is used.
- Scheduler Work Area: If the library is to contain system programs, the space available on the pack must be large enough to contain a work area for the Scheduler program (one of the system programs). The work space is not included in the number you specify in the OBJECT parameter; the space is calculated and assigned by the Library Maintenance program. The amount of space needed depends on whether the inquiry capability is generated in the supervisor. All systems require two tracks, the inquiry feature requires additional tracks for a Roll-in/Roll-out area. The number of tracks needed depends on the main storage size of the system.

| Main Storage Size | Roll-in/Roll-out Tracks |
|---|---|
| 8K | 4 |
| 12K | 4 |
| 16K | 5 |
| 24K | 6 |
| 32K | 8 |

*Placement of Object Library (Disk With a Source Library):* Space for the object library must be available immediately following the source library.

*Placement of Object.Library (Disk Without a Source Library):* The program assigns the object library to the first available disk area that is large enough.

*Changing the Size of an Object (OBJECT-number)*

*Making the Library Larger:* The number of tracks you want to add must be available immediately following the object library. The program assigns the additional tracks to the library. (The starting location of the library remains unchanged.)

*Making the Library Smaller:* The program moves the end location of the object library to decrease the library size. Tracks, therefore, become available following the library.

*Reorganizing the Library:* Any time the program changes the library size, it also reorganizes the library and deletes all temporary entries. (See *Reorganizing an Object Library.*) If other functions are also being performed, the program needs a work area. (See *WORK Parameter.*)

*Deleting an Object Library (OBJECT-0)*

The program makes the disk area occupied by the object library (and the scheduler work area if this was a system pack) available for other use.

*Reorganizing an Object Library (OBJECT-R)*

Gaps can occur between object library entries when you add and delete entries. By reorganizing the library, these gaps are removed. When the library is reorganized, all temporary entries are deleted. If other functions are also being performed, the program needs a work area. (See *WORK Parameter.*)

## COPY

The copy function of the Library Maintenance program allows you to copy:

*Reader-to-Library:* Add or replace a library entry. The reader is the system input device, which is either the keyboard or a card reader.

*File-to-Library:* Add or replace one or more library entries. A disk file is the input. Each entry in the file must have a // COPY statement and a // CEND statement. The file is opened and accessed consecutively.

*Library-to-Library*

- Copy one library entry (or those entries with the same name from all libraries).
- Copy library entries that have names beginning with certain characters.
- Copy all library entries.
- Copy minimum system.

*Library-to-Printer*

- Print one library entry (or those entries with the same name from all libraries).
- Print library entries that have names beginning with certain characters.
- Print all library entries of a certain type.
- Print directory entries for library entries of a certain type.
- Print entries from all directories including the system directory.
- Print system directory only.

*Library-to-Card*

- Punch one library entry (or those entries with the same name from all libraries).
- Punch library entries that have names beginning with certain characters.
- Punch all library entries of a certain type.

*Library-to-Printer And Card*

- Print and punch one library entry (or those entries with the same name from all libraries).
- Print and punch library entries that have names beginning with certain characters.
- Print and punch all temporary or permanent library entries of a certain type.

Copying a library entry involves:

- Identifying the location of an entry.
- Identifying an entry.
- Removing and reinserting blanks and duplicate characters.

*Identifying the Location of an Entry.* An entry may be read from either the system input device (keyboard or card reader) or from disk. It can be copied to disk, printer, or cards.

*Identifying an Entry.* Entries are identified by their type and name. Entries that can be copied include source library, object library, and system directory entries. A name identifies specific entries within the library or directory. You can also further identify an entry by designating whether it is temporary or permanent. This allows the program to make a check before replacing an entry.

*Removing and Reinserting Blanks and Duplicate Characters.* Source statements and procedures are placed in the source library. Before source statements or procedures are put in the source library, blanks and duplicate characters are removed to save space. When the source statements or procedures are used blanks and duplicate characters are reinserted. Procedures are left unchanged when placed in the source library.

## COPY Control Statement Summary: Reader-To-Library

**Add or Replace a Library Entry**

// COPY FROM-READER,LIBRARY-$\left\{\begin{matrix}S\\P\\O\\R\end{matrix}\right\}$,NAME-name,

TO-code,RETAIN-$\left\{\begin{matrix}T\\P\\R\end{matrix}\right\}$

*Library Entry:*

// CEND $\left\{\right.$ Must always follow the source or object entry being placed into the source or object library.

## COPY Control Statement Summary: File-To-Library

**Add or Replace One or More Library Entries**

// COPY FROM-DISK,FILE-filename,RECL-$\left\{\begin{matrix}80\\96\end{matrix}\right\}$,

TO-code,RETAIN-$\left\{\begin{matrix}R\\P\end{matrix}\right\}$

*Example of data in disk file:*

```
// COPY FROM-READER,LIBRARY-O,RETAIN-P,
   NAME-DECK01 ❶
   -
   -
   load module
   -
   -
// CEND
// COPY LIBRARY-S,NAME-DECK02 ❶
   -
   -
   source module
   -
   -
// CEND
```

❶ Only the LIBRARY and NAME parameters are required. Other parameters are ignored.

## COPY Control Statement Summary: Library-To-Library

**Copy One Library Entry (or Entries with the Same Name from All Libraries)**

```
                             S
                             P                                   T
// COPY FROM-code,LIBRARY-  O     ,NAME-name,TO-code,RETAIN-   P   ,NEWNAME-name [1]
                             R                                   R
                             ALL
```

**Copy Library Entries that Have Names Beginning with Certain Characters**

```
                             S
                             P                                              T
// COPY FROM-code,LIBRARY-  O     ,NAME-characters.ALL,TO-code,RETAIN-   P   ,NEWNAME-characters [1]
                             R                                              R
                             ALL
```

**Copy All Library Entries**

```
                             S
                             P                                 T
// COPY FROM-code,LIBRARY-  O     ,NAME-ALL,TO-code,RETAIN-   P
                             R                                 R
                             ALL
```

**Copy Minimum System**

```
// COPY FROM-code,LIBRARY-O,NAME-SYSTEM,TO-code
```

[1] NEWNAME parameter is needed in any of the following cases:

1. If you want the copy to have a different name than the original entry.
2. If you want to replace an entry on the TO disk with an entry from the FROM disk, but the entries have different names.
3. If you want the names of the copies to begin with different characters than the names of the original entries, the same number of characters must be in the NEWNAME parameter as in the NAME parameter.
4. If the FROM and TO packs are the same pack.

*Note:* NEWNAME cannot be DIR, ALL, or SYSTEM.

## COPY Control Statement Summary: Library-To-Printer And/or Card

**Print And/or Punch One Library Entry (or Entries with the Same Name from All Libraries)**

```
                                  ( S   )
                                  ) P   (               ( PUNCH  )
// COPY FROM-code, LIBRARY-      <  O    >,NAME-name,TO- <  PRINT   >
                                  ) R   (               ( PRTPCH )
                                  ( ALL )
```

**Print And/or Punch Temporary and Permanent Library Entries that Have Names Beginning with Certain Characters**

```
                                  ( S   )
                                  ) P   (                           ( PUNCH  )
// COPY FROM-code,LIBRARY-       <  O    >,NAME-characters.ALL,TO- <  PRINT   >
                                  ) R   (                           ( PRTPCH )
                                  ( ALL )
```

**Print And/or Punch All Temporary and Permanent Library Entries of a Certain Type**

```
                                  ( S )
                                  ) P (                 ( PUNCH  )
// COPY FROM-code,LIBRARY-       <  O  >,NAME-ALL, TO- <  PRINT   >
                                  ( R )                 ( PRTPCH )
```

**Print Directory Entries for Library Entries of a Certain Type**

```
                                  ( S )
                                  ) P (
// COPY FROM-code,LIBRARY-       <  O  >,NAME-DIR,TO-PRINT
                                  ( R )
```

**Print Entries from All Directories Including System Directory**

```
// COPY FROM-code,LIBRARY-ALL,NAME-DIR,TO-PRINT
```

**Print System Directory Entries Only**

```
// COPY FROM-code,LIBRARY-SYSTEM,NAME-DIR,TO-PRINT
```

**Print Directory Entries, Omitting Selected Entries**

```
                                  ( S   )
                                  ) P   (                          ( name            )
// COPY FROM-code,LIBRARY-       <  O    >,NAME-DIR,TO-PRINT,OMIT- <                  >
                                  ) R   (                          ( characters.ALL  )
                                  ( ALL )
```

## Copy Parameters

| Parameter | Meaning |
|---|---|
| FROM-READER | Entry to be placed in library is to be read from system input device, which can be a keyboard or card reader. |
| FROM-code | Location of disk containing library entries being copied, printed, or punched. Possible location codes are:<br><br>*Code* — *Meaning*<br>R1 — Removable disk on drive one<br>F1 — Fixed disk on drive one<br>R2 — Removable disk on drive two<br>F2 — Fixed disk on drive two |
| FROM-DISK | The entry or entries to be placed into a library or libraries reside in a disk file. The disk file must be described by an OCL FILE statement. |
| FILE-filename | For a file-to-library copy, this parameter is needed to identify the file on disk. The filename must match the filename on the OCL FILE statement. |
| RECL-$\left\{\begin{matrix}80\\ \underline{96}\end{matrix}\right\}$ | For a file-to-library copy, this parameter gives the size of the disk records. Only 80 or 96 column card image records (unblocked) are allowed. If this parameter is omitted, 96 is assumed. |
| LIBRARY-$\left\{\begin{matrix}S\\P\\O\\R\end{matrix}\right\}$ | Type of library entries involved in copy use. Possible codes are:<br><br>*Code* — *Meaning*<br>S — Source statements (source library)<br>P — OCL procedure (source library)<br>O — Object programs (object library)<br>R — Routines (object library) |
| LIBRARY-ALL | All types of entries (S, P, O, and R) from both libraries are involved in copy use. |
| LIBRARY-SYSTEM | Only system directory entries are being printed. |
| NAME-$\left\{\begin{matrix}\text{name}\\ \text{characters.ALL}\\ \text{ALL}\end{matrix}\right\}$ | Specific library entries on the FROM pack, of the type indicated in LIBRARY parameter, involved in copy use. Possible information is:<br><br>*Information* — *Meaning*<br>name — Name of the library entry involved.<br>characters.ALL — Only those entries beginning with the indicated characters. The name of the copies and original entries will be the same unless you use a NEWNAME parameter (NEWNAME-characters). (You can use up to five characters.)<br>ALL — All entries. (The type indicated in LIBRARY parameter. To copy a system which you can IPL, specify LIBRARY-ALL and NAME-ALL.) |

| Parameter | Meaning |
|---|---|
| NAME-SYSTEM | Only system programs that make up the minimum system are involved in the copy use. The minimum system is made up of system programs necessary to load and run programs. System programs necessary to generate and maintain the system are not included. |
| NAME-DIR | Directory entries for all library entries of the type indicated in the LIBRARY parameter are involved in the copy use. If the LIBRARY parameter is LIBRARY-ALL, system directory entries are also printed. |
| NAME-$cc.ALL | The IBM program with the name beginning with the indicated characters ($cc) is involved in the copy use. For example, $MA.ALL means the Library Maintenance program ($MAINT). |
| RETAIN-$\left\{\begin{matrix}T\\P\\R\end{matrix}\right\}$ | *Adding Entry to Library.* RETAIN gives designation of the TO entry:<br><br>*Code* — *Meaning*<br>T — Temporary<br>P or R — Permanent<br><br>*Replacing Existing Library Entry.* RETAIN gives designation of the TO entry and tells program whether to halt before replacing entry:<br><br>*Code* — *Meaning*<br>T — Temporary designation. Halt before replacing entry.<br>P — Permanent designation. Halt before replacing entry.<br>R — Permanent designation. Do not halt before replacing entry.<br><br>*Printing or Punching Entries.* The RETAIN parameter is ignored. |
| TO-code | Location of disk that is to contain the copies of the entries:<br><br>*Code* — *Meaning*<br>R1 — Removable disk on drive one<br>F1 — Fixed disk on drive one<br>R2 — Removable disk on drive two<br>F2 — Fixed disk on drive two |
| TO-PRINT | Entries are being printed. |
| TO-PUNCH | Entries are being punched. |
| TO-PRTPCH | Entries are being printed and punched. |
| NEWNAME-name | Name you want used on the TO disk to identify the entries being put on that disk. If you omit this parameter, the program uses the NAME parameter in naming the entries. |
| NEWNAME-characters | Beginning characters you want to use in names identifying entries being put on TO disk. You must use the same number of characters as in the NAME parameter (NAME-characters.ALL). If you omit this parameter, the program uses the NAME parameter in naming the entries. |
| OMIT-name | When printing directory entries, omit the entry specified by *name*. |
| OMIT-characters.ALL | When printing directory entries, omit all entries with these beginning characters. |

## Using the Copy Function

### *Library Directories*

#### *Source and Object Library Directories*

- The source and object libraries have separate library directories. Every library entry has a corresponding entry in its library directory. The directory entry contains such information as the name and location of the library entry. (See *Printout of Directory Entries.*)
- The Library Maintenance program makes entries in the directories when it puts entries in the libraries.

#### *System Directory*

- Every disk that contains libraries contains a system directory. The system directory contains information about the sizes of and available space in libraries and their directories. (See *Printout of Directory Entries.*)
- The Library Maintenance program creates and maintains the system directory.

### *Naming Library Entries*

*Characters to Use:* Use any combination of System/3 characters except blanks, commas, quotes, and periods. (Appendix A lists the characters.) The names of all IBM programs begin with a dollar sign ($). Therefore, to avoid possible duplication, do not use a dollar sign as the first character in the names you use for your entries. The first character must be alphabetic.

*Length of Name:* The name can be from one to six characters long.

*Restricted Names:* Do not use the names ALL, DIR, and SYSTEM. They have special meanings in the NAME and NEWNAME parameters.

*Entries with the Same Name:* For each of the two physical libraries, source and object, there are two types of entries. The source library has type P and type S entries. The object library has type O and type R entries. Entries of the same type cannot have the same name, but entries of different types may. For example, two procedures in source library cannot have the same name, but a procedure and a set of source statements can.

### *Retain Types*

#### *Temporary Entries*

- Temporary entries are entries you do not intend to keep in your libraries. They are normally used only once or a few times over a short period.
- In the object library, temporary entries are placed together following the permanent entries. Any time a permanent entry is added to the library, all temporary entries are deleted. Temporary entries are also deleted when you replace one permanent entry with another.
- In the source library, temporary and permanent entries can be in any order. One entry is placed after another regardless of their designations. Temporary entries, therefore, are not automatically deleted every time you add a permanent entry. However, when the source library is reallocated or reorganized, only permanent entries will remain.
- You can use temporary entries as often as you like until they are deleted.
- A temporary entry cannot replace a permanent entry.

#### *Permanent Entries*

- Permanent entries are entries you intend to keep in your libraries. They are normally entries you use often or at regular intervals (once a week, once a month, and so on).
- The program will not delete permanent entries unless you use the delete function of Library Maintenance to delete them, or the allocate function to delete the entire library.

*Reader-to-Library*

*Input:* The program reads one library entry. It can be any one of the following types:

1. Source statements
2. Procedure
3. Object program
4. Routine

The entry is read from the system input device, which is normally the keyboard. The operator can, however, change the system input device by using the OCL READER statement.

The header card on an object deck (H in column 1) contains the date the deck was punched. This date is in columns 58-63 and is in the format of the system date, either mmddyy or ddmmyy.

*Output*

- Blanks and duplicate characters are removed from source statements and procedures before they are put in the source library. The program does not check them for errors.
- Object programs and routines are placed in the object library.

*Adding Entries:* The program can add a new entry to a library. The name of the entry is taken from the NAME parameter. See *Naming Library Entries* for valid names. The RETAIN parameter specifies whether the entry will be temporary or permanent. If the RETAIN parameter is omitted, RETAIN-T is assumed (see *Retain Types*).

*Replacing Existing Entries*

- The program can replace an existing library entry with the entry you are putting in the library. The RETAIN parameter specifies the new retain type. If the RETAIN parameter is omitted, RETAIN-T is assumed. A temporary entry cannot replace a permanent entry.
- The program can halt before replacing an existing entry. Whether it does depends on the RETAIN parameter you use. (See RETAIN parameter.)
- Before the new entry is added, the duplicate entry is deleted. Additional library space is not needed unless the new entry is larger than the old one.

*File-to-Library*

*Input:* The disk file can contain one or more library entries. The entries must be in the format put out by the library-to-card function or by the linkage editor. The // COPY statement at the beginning of each entry contains the name of the entry and the type of library (S, P, O, R). A // CEND statement must follow each entry in the file.

The disk file must be a consecutive file and be defined by a FILE statement in the OCL for the Library Maintenance program.

*Output:* The output from the file-to-library function is the same as for the reader-to-library function except that temporary entries are not allowed.

*Library-to-Library*

*Input:* The program can copy one or more library entries from one disk to another. The types of entries can be:

1. Source statements
2. Procedures
3. Object programs
4. Routines
5. All the preceding types
6. Minimum system

The NAME and LIBRARY parameters specify which entries to copy.

*Output*

- The entries, regardless of their type, are copied from one disk to the other without change. However, if all library entries are copied (LIBRARY-ALL,NAME-ALL), the object library is reorganized, and temporary entries become permanent entries in both the source and object libraries.
- Entries can be copied and renamed on the same disk by using the NEWNAME parameter. (See *NEWNAME parameter* and *Naming Library Entries*.)
- If you are copying a minimum system or all of the types, (LIBRARY-ALL,NAME-ALL), the object library on the disk you specify in the TO parameter must not contain any entries, or deleted entries. When LIBRARY-ALL,NAME-ALL is specified and the FROM disk is a system disk, then the TO disk will be a system disk.
- The RETAIN parameter specifies whether the entries will be temporary or permanent. If the RETAIN parameter is omitted. RETAIN-T is assumed. When the parameters LIBRARY-ALL and NAME-ALL or LIBRARY-O and NAME-SYSTEM are used, RETAIN-P is assumed and RETAIN-T is invalid.

*Adding Entries*

- You can omit the NEWNAME parameter. If you do, the name used for the copy is taken from the NAME parameter. (The copy will have the same name as the original entry.)
- If NAME-ALL is specified, the names by which the entries are identified on the FROM disk are also used on the TO disk to identify the entries.

*Replacing Existing Entries*

- The program can replace existing entries with the entries you are putting in the library. If the entry you are copying (the entry on the disk you identify in the FROM parameter) has the same name as the entry you are replacing (the entry on the disk you identify in the TO parameter), you must omit the NEWNAME parameter because the NEWNAME parameter cannot be the same as the NAME parameter. If the names are not the same, you must use the NEWNAME parameter to give the name of the entry being replaced.
- The program can halt before replacing an existing entry. Whether it does depends on the RETAIN parameter (see *RETAIN Parameter*).
- A temporary entry cannot replace a permanent entry.

*Library-to-Printer and/or Card*

*Types of Entries that Can Be Printed or Punched*

- The program can print or punch one or more library entries. They can be any one of the following types:
    1. Source statements
    2. Procedures
    3. Object programs
    4. Routines
    5. All of the preceding types (limited to entries having the same name and entries beginning with the same characters).

- The program can print (but not punch) the following types of directory entries:

  1. Source statements
  2. Procedures
  3. Object programs
  4. Routines
  5. System directory
  6. All of the preceding types

  The program will sort directory names before printing them only if there is available work space on the FROM pack. This causes an allocation of disk space that counts toward the total of four allowable allocations. (See *Index Entry Allocation of Disk Space*.)

*Printed or Punched Library Entries*

- Blanks and duplicate characters are re-inserted into source statements and procedures to make them readable.
- Object programs and routines are printed and punched as they exist in the library.

*Printout of Directory Entries*

- Source library directory
- Object library directory
- System directory

**Source Library Directory**

*Printout*

```
SOURCE DIRECTORY FROM XX VOL ID XXXXXX MM/DD/YY

                         ADDRESS
TYPE       NAME       FIRST@    LAST@       ATTRI        #SECTORS
X          XXXXXX     TTT-SS    TTT-SS        X            XXXX
```

*Explanation*

| Heading | Meaning |
|---|---|
| TYPE | S=source statements<br>P=procedure |
| NAME | Name of library entry (up to six characters) |
| ADDRESS (FIRST and LAST) | Addresses of first and last secotrs that contain the library entry. Addresses are expressed by track and sector numbers. EXAMPLE: 008-03 means track 8, sector 3. |
| ATTRI (Attribute) | T=temporary<br>P=permanent |
| #SECTORS | Total number of sectors used for the library entry. |

## Object Library Directory

*Printout*

```
OBJECT LIBRARY FROM XX     VOL. ID XXXXXX    MM/DD/YY

                   DISK      CYL/   TXT-   LINK   RLD    ENTRY   CORE                   TOT
TYPE     NAME      ADD       SEC    CAT    ADDR   DISP   PNT     SEC     ATTR   LEVEL   SEC
A L      XXXXXX    TTT/SS    CC/SS  XXX    XXXX   XX     XXXX    XXX     XXXX   XXX     XXXX
```

*Explanation*

| Heading | Meaning |
|---|---|
| TYPE | A { P=permanent, T=temporary } Attribute<br>L { O=object, R=routine } Library |
| NAME | Name of library entry (up to six characters) |
| DISK ADD | Address where library entry begins on disk. EXAMPLE: 015/10 means track 15, sector 10 (in decimal). T = track, S = sector. |
| CYL/SEC | Address where library entry begins on disk (in hexadecimal). C = cylinder, S = sector. |
| TXT-CAT | For object programs, this number indicates the number of sectors used for the text portion of the library entry. Object programs consist of two parts: text and RLD. Text is the program or routine instructions. RLD is information used in loading the program for execution.<br><br>For routines, this number is the category of the routine. This number is used by the Overlay Linkage Editor for determining overlays. |
| LINK ADDR | Object programs only. Assigned hexadecimal core address of this library entry. |
| RLD DISP | Object programs only. It indicates the hexadecimal position in which RLD information begins in the last text sector. If the last text sector contains no RLD information, the RLD displacement is 0, indicating the information starts in the next sector. |
| ENTRY PNT | Object programs only. Main storage address (hexadecimal) where program execution begins before relocations. |
| CORE SEC | Core size given in sectors, required to run the program. |

| Heading | Meaning |
|---|---|
| ATTR | Byte 1:<br><br>Bit 0=1—Permanent entry<br>Bit 0=0—Temporary entry<br><br>Bit 1=1—Inquiry. This program requires that the Inquiry key be pressed to start processing.<br><br>Bit 2=1—Inquiry invoking. This program runs in program level 1 and can be rolled out to allow an Inquiry program to run.<br><br>Bit 3 Reserved<br><br>Bit 4=1—Source required. This program requires the allocation of the $WORK and $SOURCE files. $SOURCE must be filled either from the system input device or a source library.<br><br>Bit 5=1—Deferred mount. This program accepts mounting of packs during its execution.<br><br>Bit 6=1—PTF applied. A program temporary fix (PTF) has been applied to this program.<br><br>Bit 7=1—Overlay object program<br><br>Byte 2:<br><br>Bit 0=1—System Input dedication. The system input device must be dedicated to this program. The device is released at end of job.<br><br>Bit 1 Reserved<br><br>Bit 2=1—Direct source read. This program can have a COMPILE statement and a no-source-required attribute (byte 1, bit 4=0). The program will access the source itself.<br><br>Bit 3-4 Reserved<br><br>Bit 5=1—Program common. This program requires that a new load address be calculated at load time to place it in main storage beyond its own program common region.<br><br>Bits 6-7 Reserved |
| LEVEL | Release level of system programs. For user programs this can be assigned in the Overlay Linkage Editor |
| TOT SEC | Total number of disk sectors occupied by the library entry |

## System Directory Printout

*System Directory from XX Vol. ID XXXXXX DD/MM/YY*

*Source Library*

| | ❶ |
|---|---|
| Source Directory Location | TTT-SS |
| Next Available Library Sector | TTT-SS |
| End of Library | TTT-SS |
| Number of Directory Sectors | XXX |
| Number of Permanent Library Sectors | XXX |
| Number of Active Library Sectors | XXX |
| Number of Available Library Sectors | XXX |
| Allocated Size of Library | YYY |

*Object Library*

| | |
|---|---|
| Object Directory Location | TTT-SS |
| Allocated Size of Directory | YYY |
| Start of Library | TTT-SS |
| Allocated End of Library | TTT-SS |
| Extended End of Library | TTT-SS |
| Number of Available Permanent Directory Entries | XXX |
| Number of Availzble Temporary Directory Entries | XXX |
| First Temporary Directory Entry | TTT-SS-DDD |
| Next Available Temporary Directory Entry | TTT-SS-DDD |
| Next Available Library Sector for Permanents | TTT-SS |
| Next Available Library Sector for Temporaries | TTT-SS |
| Number of Available Library Sectors for Permanents ❷ | XXX |
| Number of Available Library Sectors for Temporaries | XXX |
| Number of Active Library Sectors | XXX |
| Number of Active Object Permanent Library Sectors | XXX |
| Number of Active Routine Permanent Library Sectors | XXX |
| Allocated Size of Library | YYY |
| Roll-in/Roll-out Location | TTT-SS |
| Roll-in/Roll-out Size | YYY |
| Scheduler Work Area Location | TTT-SS |
| Scheduler Work Area Size | YYY |
| Start of Libraries | TTT-SS |
| End of Libraries | TTT-SS |

❶ TTT-SS-DDD means track, sector, and displacement. Displacement is the number of characters from the beginning of the sector. XXX = number of sectors. YYY = number of tracks.

❷ 'Number of Available Library Sectors for Permanents' reflects the space available from the last permanent library entry to the allocated end of the library. Gaps and temporary library entries are not reflected in this figure. The actual number of sectors available for permanent entries may be calculated by subtracting 'Number of Active Object Permanent Library Sectors' from the total number of sectors in the library. If the result is larger than 'Number of Available Library Sectors for Permanents', the library should be reorganized using the ALLOCATE function to remove gaps and temporary object library entries.

## DELETE

*Uses*

- Delete a temporary or permanent entry from a library (or entries with the same name from all libraries).
- Delete temporary or permanent entries that have names beginning with certain characters.
- Delete all temporary or permanent entries of a certain type.

*Restrictions*

The following restrictions apply to the delete function:

- System modules cannot be deleted from the active system pack (the pack the system was loaded from during IPL).
- When all temporary entries are deleted from the object library using LIBRARY-O,NAME-ALL,RETAIN-T, the temporary routines (LIBRARY-R) are also deleted.
- The RETAIN parameter must match the attribute of the entry in the library otherwise the entry is considered not found. RETAIN-T is assumed if the RETAIN parameter is omitted.
- Library Maintenance modules cannot be deleted from the active program pack.

### Control Statement Summary

**Delete a Temporary or Permanent Library Entry (or Entries with the Same Name from All Libraries)**

```
                                 ( S   )
                                 ( P   )                      ( T )
// DELETE FROM-code,LIBRARY-    < O   >,NAME-name,RETAIN-    <   >
                                 ( R   )                      ( P )
                                 ( ALL )
```

**Delete Temporary or Permanent Entries With Names Beginning With Certain Characters**

```
                                 ( S   )
                                 ( P   )                                ( T )
// DELETE FROM-code,LIBRARY-    < O   >,NAME-characters.ALL,RETAIN-    <   >
                                 ( R   )                                ( P )
                                 ( ALL )
```

**Delete All Temporary or Permanent Entries of a Certain Type**

```
                                ( S )
                                ( P )                  ( T )
// DELETE FROM-code,LIBRARY-   < O >,NAME-ALL,RETAIN- <   >
                                ( R )                  ( P )
```

**Delete Parameters**

| Parameter | Meaning | |
|---|---|---|
| FROM-{R1, F1, R2, F2} | Location of disk that contains library entries you are deleting. Possible codes are: | |
| | *Code* | *Meaning* |
| | R1 | Removable disk on drive one |
| | F1 | Fixed disk on drive one |
| | R2 | Removable disk on drive two |
| | F2 | Fixed disk on drive two |
| LIBRARY-{S, P, O, R, ALL} | Type of entries being deleted. Possible codes are: | |
| | *Code* | *Meaning* |
| | S | Source statements (source library) |
| | P | Procedures (source library) |
| | O | Object programs (object library) |
| | R | Routines (object library) |
| | ALL | All types of entries (S, P, O, and R) are being deleted. |
| NAME-{name, characters.ALL, ALL} | Particular entries, of type indicated in LIBRARY parameter, being deleted. These entries are further identified by the RETAIN parameter. Possible codes are: | |
| | *Code* | *Meaning* |
| | name | Name of the library entry, or entries, being deleted. |
| | character.ALL | Entries that have names beginning with the indicated characters. You can use up to five characters. EXAMPLE: NAME-INV.ALL refers to the entries having names that begin with INV. |
| | ALL | All entries (of the type indicated in LIBRARY parameter). NAME-ALL cannot be used with LIBRARY-ALL. |
| RETAIN-{T, P} | Designation of entries being deleted: | |
| | *Code* | *Meaning* |
| | T | Temporary |
| | P | Permanent |

## MODIFY

### Uses

The Modify function is intended primarily for maintenance of source statements and procedures by using card input. The Modify function can be used to:

- Reserialize a source library entry.
- List the statements in a source library entry.
- Remove statements from a source library entry.
- Replace source library statements.
- Insert statements into a source library entry.

*Restrictions*

- At least three control statements must be entered to modify the source library. A // MODIFY statement is needed to describe the library entry. A // REMOVE, // REPLACE, or // INSERT statement describes the type of modification. A // CEND statement indicates the end of the control statements.
- The sequence numbers specified by the FROM-seqno, TO-seqno, and AFTER-seqno parameters on the // REMOVE, // REPLACE, and // INSERT statements must be valid numbers and exist in the source library entry. There are no default values for these parameters. The number of digits entered must be the same as the number of positions specified by the SEQFLD parameter.
- All statements in a source library entry must have ascending sequence numbers in the positions specified by the SEQFLD parameter.
- Multiple operations (REMOVE, REPLACE, INSERT) may be performed within the same MODIFY run if they are done in an ascending sequential order. That is, the FROM sequence number in a REMOVE or REPLACE statement must be greater than the last sequence number in the preceding statement. The AFTER sequence number of an INSERT statement must be equal to or greater than the last sequence number of the preceding statement. Consecutive INSERT statements must not have the same sequence number.
- When modification is complete, the directory entry is written back with a permanent attribute.
- The control statements following the // MODIFY statement are read from the system input device, which can be the keyboard or card reader.
- Sequence numbers are a physical part of the source record and must be placed where they will not conflict with other data in the record. In a procedure the sequence numbers should be placed near the end of the record beyond the OCL and utility control statement's keywords and parameters.

  Invalid responses may result for OCL procedures with delayed responses, because the procedure is called, the sequence number may be recognized as the response.

  The sequence numbers should be placed in source statements where they will not overlay data. For example, data could be destroyed if sequence numbers were placed in RPG II source statements that contained compile-time tables. If the statement contains table data in positions 1-85, the sequence numbers for the source module should begin after positions 85 (86-96).

## Control Statement Summary

Initiate Modification

// MODIFY NAME-name,FROM-code,LIBRARY- $\left\{\begin{matrix}\text{S}\\\text{P}\end{matrix}\right\}$ ,WORK-code,RESER- $\left\{\begin{matrix}\text{YES}\\\text{NO}\\\text{ONLY}\end{matrix}\right\}$ ,LIST- $\left\{\begin{matrix}\text{YES}\\\text{NO}\end{matrix}\right\}$,

SEQFLD-xxyy,INCR-number

Control Statements Following // MODIFY

**Delete all statements between and including the FROM and TO sequence numbers.**

```
// REMOVE FROM-seqno,TO-seqno
```

**Replace all statements between and including the FROM and TO sequence numbers with the statements supplied.**

```
// REPLACE FROM-seqno,TO-seqno
    -
    -
1-n statements to replace those removed
    -
```

**Insert the supplied statements after the statement indicated by the AFTER parameter.**

```
// INSERT AFTER-seqno
    -
1-n statements to be inserted
```

## Modify Parameter

| Parameter | Meaning |
|---|---|
| NAME-name | Current name of the entry you are modifying. This is the name that identifies the entry in the library directory. |
| FROM-code | Location of the disk that contains the entry you are modifying. Possible codes are:<br>*Code* — *Meaning*<br>R1 — Removable disk on drive one<br>F1 — Fixed disk on drive one<br>R2 — Removable disk on drive two<br>F2 — Fixed disk on drive two |
| LIBRARY-$\left\{\begin{matrix}S\\P\end{matrix}\right\}$ | Type of library entry you are modifying. Possible codes are:<br>*Code* — *Meaning*<br>S — Source statements (source library)<br>P — Procedures (source library) |
| WORK-code | Location of the disk containing space the program can use as a work area. Possible codes are:<br>*Code* — *Meaning*<br>R1 — Removable disk on drive one<br>F1 — Fixed disk on drive one<br>R2 — Removable disk on drive two<br>F2 — Fixed disk on drive two |
| RESER-$\left\{\begin{matrix}\text{YES}\\\underline{\text{NO}}\\\text{ONLY}\end{matrix}\right\}$ | Specifies whether reserialization should be done when the entry is placed back in the source library. Possible information is:<br>*Information* — *Meaning*<br>YES — Reserialization is done.<br>NO — Reserialization is not done. NO is assumed if the RESER parameter is omitted.<br>ONLY — Reserialize only; no other maintenance is done. When this is coded, no REMOVE, REPLACE, or INSERT statements can be entered. A // CEND statement is not needed. |
| LIST-$\left\{\begin{matrix}\text{YES}\\\underline{\text{NO}}\end{matrix}\right\}$ | Specifies whether the source library entry should be listed when the MODIFY run is complete. NO is assumed if the LIST parameter is omitted. |
| SEQFLD-xxyy | The starting and ending positions of the field that contains the sequence number. The sequence number can be up to eight digits long. The starting position is entered first (xx) and then the ending position (yy). If this parameter is not entered, 9296 is assumed. |
| INCR-number | Increment value for sequence field if reserialization (RESER-YES or RESER-ONLY) is specified. The value can be up to five digits. If this parameter is not entered, a value of 10 is assumed. |

**Remove, Replace, Insert Parameters**

| Parameter | Meaning |
|---|---|
| FROM-seqno | The sequence number of the first statement to be used in the operation. |
| TO-seqno | The sequence number of the last statement to be used in the operation. |
| AFTER-seqno | The sequence number of the statement after which the new statements are to be added. |

## RENAME

*Uses*

- Change the name of a library entry.
- Change the names of library entries that have names beginning with certain characters.

*Restrictions*

- System modules should not be renamed on the active system pack. (The pack the system was loaded from during IPL.)
- Library Maintenance modules should not be renamed on the active program pack.

**Control Statement Summary**

**Change the Name of a Library Entry or Entries with the Same Name in All Libraries**

$$\text{// RENAME FROM-code,LIBRARY-}\begin{Bmatrix} S \\ P \\ O \\ R \end{Bmatrix}\text{ NAME-name,NEWNAME-name}$$

**Change the Name of Library Entries that have Names Beginning with Certain Characters**

$$\text{// RENAME FROM-code,LIBRARY-}\begin{Bmatrix} S \\ P \\ O \\ R \end{Bmatrix}\text{ NAME-characters.ALL, NEWNAME-characters}$$

**Rename Parameters**

| Parameter | Meaning |
|---|---|
| FROM-code | Location of disk that contains the entry you are renaming. Possible codes are:<br>*Code* *Meaning*<br>R1 Removable disk on drive one<br>F1 Fixed disk on drive one<br>R2 Removable disk on drive two<br>F2 Fixed disk on drive two |
| LIBRARY-$\begin{Bmatrix} S \\ P \\ O \\ R \end{Bmatrix}$ | Type of library entry you are renaming. Possible codes are:<br>*Code* *Meaning*<br>S Source statements (source library)<br>P Procedures (source library)<br>O Object programs (object library)<br>R Routines (object library) |
| NAME-name | Current name of the entry you are renaming. This is the name that identifies the entry in the library directory. |
| NAME-characters.ALL | Only those entries beginning with the indicated characters. (You can use up to five characters.) |
| NEWNAME-name | New name you want to give the entry. Follow these rules to construct the name:<br>1. You can use any System/3 characters except blanks, commas, quotes, and periods. (Appendix A lists the characters.) However, the names of all IBM programs begin with a dollar sign ($). Therefore, to avoid possible duplication, do not use a dollar sign as the first character in the names you use for your entries. The first character must be alphabetic.<br>2. You can use up to six characters, but you cannot use the names ALL, DIR and SYSTEM. They have special meanings in the NAME parameter. |
| NEWNAME-characters | Beginning characters you want to use in names identifying the copies. (You can use up to five characters. |

## OCL CONSIDERATIONS

### LOAD Sequence

| Keywords ❶ | Responses ❷ | Considerations |
|---|---|---|
| READY | LOAD | None |
| LOAD NAME | $MAINT | Name of Library Maintenance program. |
| UNIT | R1, R2, F1, or F2 | Location of disk containing Library Maintenance program. |
| MODIFY | RUN | None |

❶ Only the keywords listed here are required. You can bypass the rest.

❷ You end every response by pressing PROG START.

### BUILD Sequence

| Keywords ❶ | Responses ❷ | Considerations |
|---|---|---|
| READY | BUILD | None |
| BUILD NAME | procedure name | Name by which procedure will be identified in source library. |
| UNIT | R1, R2, F1, or F2 | Location of disk containing source library. |
| LOAD NAME | $MAINT | Name of Library Maintenance program. |
| UNIT | R1, R2, F1, or F2 | Location of disk containing Library Maintenance program. |
| MODIFY | INCLUDE<br>utility control statements<br>RUN | Response when including control statements in procedure. |
| | OR | |
| | RUN | Response when not including control statements in procedure. |

❶ Only the keywords listed here are required. You can bypass the rest.

❷ You end every response by pressing PROG START.

## ALLOCATE EXAMPLES

### Creating Both Source and Object Libraries on a Disk

*Statements*

```
READY                         - LOAD
*******************
010     LOAD   NAME           - $MAINT
011              UNIT         - F1
020     DATE                  -
030     SWITCH                -
040     FILE     NAME         -
*******************
MODIFY                        -
RUN
  ENTER '//' CONTROL STATEMENT
// ALLOCATE TO-R1,SOURCE-12,OBJECT-45,SYSTEM-YES

  ENTER '//' CONTROL STATEMENT

// END
```

**OCL LOAD Sequence.**

Boxed areas are operator responses.

Keywords for which no responses are shown are the ones bypassed.

RUN is the response to MODIFY even though the two words do not appear on the same line.

Message printed by Library Maintenance program.

Control statement supplied by operator.

Program creates libraries, then asks for another control statement.

END statement, supplied by operator, ends the program.

*Explanation*

- Library Maintenance program is loaded from the fixed disk on drive 1 (UNIT-F1 in OCL sequence).
- Libraries are being created on the removable disk on drive 1 (TO-R1 in ALLOCATE statement).
- Source library space is twelve tracks long (SOURCE-12).
- Object library space is 45 tracks long (OBJECT-45). The object library will contain system programs (SYSTEM-YES). Thus, the disk area will also include space for the Scheduler work area.

**Changing the Size of a Source Library**

*Statements*

```
READY                   -   LOAD
*******************
010    LOAD   NAME      -   $MAINT
011           UNIT      -   F1
020    DATE             -
030    SWITCH           -
040    FILE   NAME      -
*******************
MODIFY                  -
RUN
```

**OCL LOAD Sequence.**

Boxed areas are operator responses.

Keywords for which no responses are shown are the ones bypassed.

RUN is the response to MODIFY even though the two words do not appear on the same line.

```
   ENTER '//' CONTROL STATEMENT
```

Message printed by Library Maintenance program.

```
// ALLOCATE TO-R1,SOURCE-15,WORK-F1
```

Control statement supplied by operator.

```
   ENTER '//' CONTROL STATEMENT
```

Program changes size of library, then asks for another control statement.

```
// END
```

End statement, supplied by operator, ends the program.

*Explanation*

- Library Maintenance program is loaded from the fixed disk on drive 1 (UNIT-F1 in OCL sequence).
- Source library is located on the removable disk on drive 1 (TO-R1 in ALLOCATE statement).
- Size of the source library is being changed to 15 tracks (SOURCE-15).
- Any time the program changes the size of a library, it reorganizes the library. To do this, it needs a work area. This area is on the fixed disk on drive 1 (WORK-F1).

## Deleting the Object Library From a Disk

*Statements*

```
READY                          - LOAD
*************************
010     LOAD    NAME           - $MAINT
011             UNIT           - F1
020     DATE                   -
030     SWITCH                 -
040     FILE    NAME           -
*************************
MODIFY
RUN
```

**OCL LOAD Sequence.**

Boxed areas are operator responses.

Keywords for which no responses are shown are the ones bypassed.

RUN is the response to MODIFY even though the two words do not appear on the same line.

```
   ENTER '//' CONTROL STATEMENT
```

Message printed by Library Maintenance program.

```
// ALLOCATE TO-R1,OBJECT-0
```

Control statement supplied by operator.

```
   ENTER '//' CONTROL STATEMENT
```

Program deletes library, then asks for another control statement.

```
// END
```

END statement, supplied by operator, ends the program.

*Explanation*

- Library Maintenance program is loaded from the fixed disk on drive (UNIT-F1 in OCL sequence).
- Object library is located on the removable disk on drive 1 (TO-R1 in ALLOCATE statement).
- OBJECT-0 parameter tells the program to delete the object library. If a Scheduler work area precedes the object library, it is also deleted.

## COPY EXAMPLES

### Copying Minimum System from One Disk to Another

*Statements*

```
READY                              - LOAD
************************
010     LOAD      NAME             - $MAINT
011               UNIT             - F1
020     DATE                       -
030     SWITCH                     -
040     FILE      NAME             -
************************
MODIFY
RUN
```

**OCL LOAD Sequence.**

Boxed areas are operator responses.

Keywords for which no responses are shown are the ones bypassed.

RUN is the response to MODIFY even though the two words do not appear on the same line.

```
   ENTER '//' CONTROL STATEMENT
```

Message printed by Library Maintenance program.

```
// COPY FROM-F1,LIBRARY-O,NAME-SYSTEM,TO-R1
```

Control statement supplied by the operator.

```
   ENTER '//' CONTROL STATEMENT
```

Program copies programs, then asks for another control statement.

```
// END
```

END statement, supplied by operator, ends the program.

*Explanation*

- Library Maintenance program is loaded from the fixed disk on drive (UNIT-F1 in OCL sequence).
- System programs are in the object library on the fixed disk on drive 1 (LIBRARY-O and FROM-F1 in COPY statement).
- The NAME parameter (NAME-SYSTEM) tells the program to copy the system programs.
- The disk that is to contain the copy is the removable disk on drive 1 (TO-R1).

**Printing Library Directories**

*Statements*

```
READY                     - LOAD
*************************
010   LOAD   NAME         - $MAINT
011          UNIT         - F1
020   DATE                -
030   SWITCH              -
040   FILE   NAME         -
*************************
MODIFY
RUN
```

**OCL LOAD Sequence.**

Boxed areas are operator responses.

Keywords for which no responses are shown are the ones bypassed.

RUN is the response to MODIFY even though the two words do not appear on the same line.

```
   ENTER '//' CONTROL STATEMENT
```

Message printed by Library Maintenance program.

```
// COPY FROM-R1,LIBRARY-ALL,NAME-DIR,TO-PRINT
```

Control statement supplied by the operator.

```
   ENTER '//' CONTROL STATEMENT
```

Program prints directories, then asks for another control statement.

```
// END
```

END statement, supplied by operator, ends the program.

*Explanation*

- Library Maintenance program is loaded from the fixed disk on drive 1 (UNIT-F1 in OCL sequence).
- All library directories and the system directory on the removable disk on drive 1 are printed (COPY statement):

  1. FROM identifies the disk containing the directories.
  2. LIBRARY indicates which directories are to be printed.
  3. NAME and TO indicates that the program is to be printing directories.

## Replacing a Library Entry: Replacement Coming From Another Disk

*Situation*

Assume that you have two versions of an object program:

1. New version on the removable disk on drive 1.

2. Old version on the fixed disk on drive 1.

Both versions have the same name (ACT) and designation (permanent). You want to replace the old version with the new version.

*Statements*

```
READY                          - LOAD
*************************
010     LOAD    NAME           - $MAINT
011             UNIT           - F1
020     DATE                   -
030     SWITCH                 -
040     FILE    NAME           -
*************************
MODIFY                         -
RUN
```

**OCL LOAD Sequence.**

Boxed areas are operator responses.

Keywords for which no responses are shown are the ones bypassed.

RUN is the response to MODIFY even though the two words do not appear on the same line.

```
   ENTER '//' CONTROL STATEMENT
```

Message printed by Library Maintenance program.

```
// COPY FROM-R1,LIBRARY-O,NAME-ACCT,TO-F1,RETAIN-R
```

Control statement supplied by operator.

```
   ENTER '//' CONTROL STATEMENT
```

Program replaces library entry, then asks for another control statement.

```
// END
```

END statement, supplied by operator, ends the program.

*Explanation*

- Library Maintenance program is loaded from the fixed disk on drive 1 (UNIT-F1 in OCL sequence).

- LIBRARY-O, NAME-ACCT, and FROM-R1 in the COPY statement tell the program to read the object program named ACCT from the removable disk on drive 1.

- TO-F1 tells the program to copy the object program to the fixed disk on drive 1. There is no NEWNAME parameter in the COPY statement. Therefore, the name the program will have on the fixed disk is ACCT (NAME-ACCT). Since the old version of the program already exists on the fixed disk under that name, the old version is replaced.

- The Library Maintenance program normally halts before replacing a library entry. The RETAIN-R parameter, however, tells the program to omit that halt.

## Disk File-To-Library

*Statements*

```
READY                                       LOAD
**************************************************
010 LOAD                NAME-               $MAINT
011                     UNIT-               F1
020 DATE (XX/XX/XX)
030 SWITCH (00000000)   -
040 FILE                NAME-               BSCAFILE
041                     UNIT-               R1
042                     PACK-               BSCA
043                    LABEL-               PROGRAMS
050 FILE                NAME-
**************************************************
MODIFY
RUN
  ENTER'//' CONTROL STATEMENT

// COPY FROM-DISK,TO-F1,RETAIN-P,FILE-BSCAFILE

XX COPY LIBRARY-P,NAME-PAYREC
XX COPY LIBRARY-O,NAME-PAYREC
XX END

   ENTER '//' CONTROL STATEMENT

// END
```

**OCL LOAD Sequence.**

Boxed areas are operator responses.

Keywords for which no responses are shown are the ones bypassed.

RUN is the response to MODIFY even though the two words do not appear on the same line

Message printed by Library Maintenance program.

Control statement supplied by operator.

Control statements from disk file.

Program copies programs, then asks for another control statement.

END statement, supplied by operator, ends the program.

*Explanation*

- The OCL for a File-to-Library copy must contain a FILE statement for the disk file.
- The filename on the // COPY statement (FILE-BSCAFILE) matches the filename on the OCL FILE statement (NAME-BSCAFILE).
- The // COPY statement does not contain a RECL parameter, so a record length of 96 is assumed.
- All source and object decks in the disk file must have a // COPY statement as the first card image. These // statements (including the // END statement) are printed with XX replacing the // to indicate they were read from disk rather than from the system input device.
- The // END statement read from the file (printed XX END), causes the next statement to be read from the system input device. A // END statement must still be read from the system input device to indicate the end of the Library Maintenance control statements.

## DELETE EXAMPLES

### Deleting a Temporary Entry From a Library

*Statements*

```
READY                          -  LOAD
************************
010     LOAD    NAME           -  $MAINT
011             UNIT           -  F1
020     DATE                   -
030     SWITCH                 -
040     FILE    NAME           -
************************
MODIFY                         -
RUN
```

**OCL LOAD Sequence.**

Boxed areas are operator responses.

Keywords for which no responses are shown are the ones bypassed.

RUN is the response to MODIFY even though the two words do not appear on the same line.

```
   ENTER '//' CONTROL STATEMENT
// DELETE FROM-R1,LIBRARY-S,NAME-PAYROL
   ENTER '//' CONTROL STATEMENT

// END
```

ENTER '//' CONTROL STATEMENT — Message printed by Library Maintenance program.

// DELETE FROM-R1,LIBRARY-S,NAME-PAYROL — Control statement supplied by operator.

ENTER '//' CONTROL STATEMENT — Program deletes library entry, then asks for another control statement.

// END — END statement, supplied by operator, ends the program.

*Explanation*

- Library Maintenance program is loaded from the fixed disk on drive 1 (UNIT-F1 in OCL sequence).
- The program deletes a set of source statements (LIBRARY-S in DELETE statement) named PAYROL (NAME-PAYROL) from the removable disk on drive 1 (FROM-R1).
- The absence of a RETAIN parameter implies that the entry designation is temporary. If the designation were permanent, RETAIN-P would have been required.

**Deleting All Temporary Entries With Names That Begin With Certain Characters**

*Statements*

```
READY                          - LOAD
*************************
010    LOAD    NAME            - $MAINT
011            UNIT            - F1
020    DATE                    -
030    SWITCH                  -
040    FILE    NAME            -
*************************
MODIFY
RUN                            -
```

**OCL LOAD Sequence.**

Boxed areas are operator responses.

Keywords for which no responses are shown are the ones bypassed.

RUN is the response to MODIFY even though the two words do not appear on the same line.

```
   ENTER '//' CONTROL STATEMENT
```

Message printed by Library Maintenance program.

```
// DELETE FROM-R1,LIBRARY-ALL,NAME-INV.ALL
```

Control statement supplied by operator.

```
   ENTER '//' CONTROL STATEMENT
```

Program deletes entries, then asks for another control statement.

```
// END
```

END statement, supplied by operator, ends the program.

*Explanation*

- Library Maintenance program is loaded from the fixed disk on drive 1 (UNIT-F1 in OCL sequence).
- The entries being deleted are on the removable disk on drive 1 (FROM-F1 in DELETE statement).
- The program deletes all entries from both source and object libraries (LIBRARY-ALL) that have names beginning with the characters INV (NAME-INV.ALL).
- The absence of a RETAIN parameter implies that temporary entries are being deleted.

## Deleting All Permanent Library Entries of One Type

*Statements*

```
READY                           - LOAD
************************
010    LOAD    NAME             - $MAINT
011            UNIT             - F1
020    DATE                     -
030    SWITCH                   -
040    FILE    NAME             -
************************
MODIFY                          -
RUN
```

**OCL LOAD Sequence.**

Boxed areas are operator responses.

Keywords for which no responses are shown are the ones bypassed.

RUN is the response to MODIFY even though the two words do not appear on the same line.

```
  ENTER '//' CONTROL STATEMENT
```

Message printed by Library Maintenance program.

```
// DELETE FROM-R1,LIBRARY-P,NAME-ALL,RETAIN-P
```

Control statement supplied by operator.

```
  ENTER '//' CONTROL STATEMENT
```

Program deletes entries, then asks for another control statement.

```
// END
```

END statement, supplied by operator, ends the program.

*Explanation*

- Library Maintenance program is loaded from the fixed disk on drive 1 (UNIT-F1 in OCL sequence).
- The entries being deleted are on the removalbe disk on drive 1 (FROM-R1 in DELETE statement).
- All permanent procedures are being deleted from the source library (LIBRARY-P,NAME-ALL RETAIN-P).

## MODIFY EXAMPLES

### Replacing Statements in a Procedure

*Statements*

```
READY-                                       LOAD
************************************************************
010 LOAD             NAME-                   $MAINT
011                  UNIT-                   F1
020 DATE (XX/XX/XX)     -
030 SWITCH (00000000)   -
040 FILE             NAME-
************************************************************
MODIFY
RUN
   ENTER '//' CONTROL STATEMENT
```

**OCL LOAD Sequence.**

Boxed areas are operator responses.

Keywords for which no responses are shown are the ones bypassed.

RUN is the response to MODIFY even though the two words do not appear on the same line.

Message printed by Library Maintenance program.

```
// MODIFY NAME-PROC01,FROM-R2,LIBRARY-P,WORK-R1,RESER-NO,LIST-YES
// REPLACE FROM-00101,TO-00102
// FILE NAME-INV,PACK-VOL2,UNIT-R1,RECORDS-300,RETAIN-P            00101
// FILE NAME-WORK,PACK-VOL2,UNIT-R1                                00102
// CEND
```

Control statement supplied by operator.

```
// LOAD BUILD,F1                                                   00100
// FILE NAME-INV,PACK-VOL2,UNIT-R1,RECORDS-300,RETAIN-P            00101
// FILE NAME-WORK,PACK-VOL2,UNIT-R1                                00102
// RUN                                                             00103
```

Program lists procedure, then asks for another control statement.

```
   ENTER '//' CONTROL STATEMENT
// END
```

END statement, supplied by operator, ends the program

*Explanation*

- The procedure named PROC01 on disk drive R2 is being modified.
- The work space will be on R1.
- The sequence numbers are in default positions 92-96.
- Statements with sequence number 00101-00102 are being replaced.
- The module is not reserialized.
- The module is listed.

## Removing Source Statements From a Module

*Statements*

**Removing Source Statements From a Module.**

```
READY-                                        LOAD
*************************************************
010 LOAD             NAME-                    $MAINT
011                  UNIT-                    F1
020 DATE (XX/XX/XX)     -
030 SWITCH (00000000)   -
040 FILE             NAME-
*************************************************
MODIFY
RUN
    ENTER'//' CONTROL STATEMENT
```

**OCL LOAD Sequence.**

Boxed areas are operator responses.

Keywords for which no responses are shown are the ones bypassed.

RUN is the response to MODIFY even though the two words do not appear on the same line.

Message printed by Library Maintenance program.

```
// MODIFY NAME-INPUT1,FROM-R1,LIBRARY-S,WORK-R1,RESER-YES,
//        LIST-NO,SEQFLD-0105,INCR-1
// REMOVE FROM-00124,TO-00156
// CEND
```

Control statements supplied by the operator.

```
   ENTER '//' CONTROL STATEMENT
```

Program removes statements, then asks for another control statement.

```
// END
```

END statement, supplied by operator, ends the program.

*Explanation*

- The source module named INPUT1 on disk drive R1 is being modified.
- The work space will be on R1.
- The sequence numbers are in positions 1-5 of the statements.
- Sequence numbers 00124-00156 are being deleted from the module.
- The module is reserialized with increments of one.
- The module is not listed.

**Inserting a Statement in a Source Module**

*Statements*

```
READY-                                           LOAD
**************************************************
010 LOAD                NAME-                    $MAINT
011                     UNIT-                    F1
020 DATE (XX/XX/XX)     -
030 SWITCH (00000000)   -
040 FILE                NAME-
**************************************************
MODIFY
RUN
   ENTER '//' CONTROL STATEMENT
// MODIFY FROM-F1,WORK-F1,NAME-COST,LIBRARY-S,
//        RESER-YES,SEQFLD-8084,LIST-YES
// INSERT AFTER-00070
00080I                                                 3    8 DATE
// CEND
```

Source module listed with new entry

```
   ENTER '//' CONTROL STATEMENT

// END
```

**OCL LOAD Sequence.**

Boxed areas are operator responses.

Keywords for which no responses are shown are the ones bypassed.

RUN is the response to MODIFY even though the two words do not appear on the same line.

Message printed by Library Maintenance program.

Control statements supplied by the operator.

Program inserts statements, then asks for another control statement.

END statement, supplied by operator, ends the program.

*Explanation*

- The source module COST on fixed disk drive one is being modified.
- The work space is on F1.
- The sequence numbers are in positions 80-84 of the statements.
- A statement is being inserted after statement number 00070.
- The module is reserialized with the default increment value of 10.
- The module is listed.

## RENAME EXAMPLE

**Renaming a Set of Source Statements in a Source Library**

*Statements*

```
READY                           -   LOAD
***********************************
010     LOAD        NAME   -   $MAINT
011                 UNIT   -   F1
020     DATE               -
030     SWITCH             -
040     FILE        NAME   -
***********************************
MODIFY                     -
RUN
```

**OCL LOAD Sequence.**

Boxed areas are operator responses.

Keywords for which no responses are shown are the ones bypassed.

RUN is the response to MODIFY even though the two words do not appear on the same line.

```
   ENTER '//' CONTROL STATEMENT
```

Message printed by Library Maintenance program.

```
// RENAME FROM-R1,LIBRARY-S,NAME-ACCT,NEWNAME-ACCT1
```

Control statement supplied by operator.

```
   ENTER '//' CONTROL STATEMENT
```

Program renames entry, then asks for another control statement.

```
// END
```

END statement, supplied by operator, ends the program.

*Explanation*

- Library Maintenance program is loaded from the fixed disk on drive 1 (UNIT-F1 in OCL sequence).
- The removable disk on drive 1 contains the entry being renamed (FROM-R1 in RENAME statement).
- The entry is a set of source statements in the source library (LIBRARY-S). Its name is ACCT (NAME-ACCT).
- The entry name is being changed to ACCT1 (NEWNAME-ACCT1).

# Appendix A: IBM System/3 Standard Character Set

| Character | Hexadecimal Equivalent | | | | |
|---|---|---|---|---|---|
| Blank | 40 | | 7E | W | E6 |
| ¢ | 4A | ≠ | 7F | X | E7 |
| | 4B | A | C1 | Y | E8 |
| < | 4C | B | C2 | Z | E9 |
| ( | 4D | C | C3 | 0 | F0 |
| + | 4E | D | C4 | 1 | F1 |
| \| | 4F | E | C5 | 2 | F2 |
| & | 50 | F | C6 | 3 | F3 |
| ↑ | 5A | G | C7 | 4 | F4 |
| $ | 5B | H | C8 | 5 | F5 |
| • | 5C | I | C9 | 6 | F6 |
| ) | 5D | } | D0 | 7 | F7 |
| : | 5E | J | D1 | 8 | F8 |
| ¬ | 5F | K | D2 | 9 | F9 |
| - (minus) | 60 | L | D3 | | |
| / | 61 | M | D4 | | |
| . | 6B | N | D5 | | |
| % | 6C | O | D6 | | |
| — (underscore) | 6D | P | D7 | | |
| > | 6E | Q | D8 | | |
| ? | 6F | R | D9 | | |
| : | 7A | S | E2 | | |
| # | 7B | T | E3 | | |
| @ | 7C | U | E4 | | |
| ' (Apostrophe) | 7D | V | E5 | | |

# Appendix B: Records — Tracks Conversion

## For Sequential or Direct Files

To determine how many tracks will be required for a sequential or direct file:

1. Number of records x record length = total number of characters.

2. Total number of characters ÷ 6144 (number of characters in a track) = number of tracks. (Round result up to nearest whole number.)

## For Indexed Files

To determine how many tracks will be required for an indexed file:

Step 1. (Tracks Required for Data)

A. Number of records x record length = total number of characters.

B. Total number of characters ÷ 6144 = number of tracks. (Round result up to nearest whole number.)

Step 2. (Tracks Required for Index)

A. Key Field length + 3 = index entry length.

B. 256 (number of characters in a sector) ÷ index entry length = number of entries per sector. (Round result down to nearest whole number.)

C. Number of records ÷ number of entries per sector = number of sectors. (Round result up to nearest whole number.)

D. Number of sectors ÷ 24 (number of sectors per track) = number of tracks. (Round result up to nearest whole number.)

Step 3. (Total Track Requirement)

Result of step 1 + result of step 2 = total number of tracks required for the indexed file.

## Appendix C: Disk Organization

| Disk Area | Contents |
|---|---|
| VTOC* | Detailed information about each file on disk |
| Source Library | Source Library Directory<br>RPG II Source Programs<br>Sort Specifications<br>Procedures<br>KSE Input (Format Descriptions or Source Statements) |
| Object Library | Object Library Directory<br>Compiled Programs<br>System Programs |
| Files | User files<br>System files |

*Volume Table of Contents

### Volume Table of Contents (VTOC)

The VTOC contains detailed information about each file on the disk. Much of this information is for system use only and is of no interest to the programmer. The VTOC file information significant to the programmer is:

1. Name.
2. Starting track location and number of tracks.
3. Designation (Permanent, Temporary, or Scratch).
4. Organization (Sequential, Direct, or Indexed).
5. Creation date.

### Source Library

Procedures, RPG II source programs, and KSE input always reside in a source library. The source library directory contains the name and address (track and sector) of each procedure, RPG II source program, and set of KSE input in the library.

### Object Library

Compiled programs and system programs always reside in an object library. The object library directory contains the name and address (track and sector) of each program in the library.

### Files

Identifying information about every file on a disk is contained in the disk VTOC.

A disk is limited to 50 files because the VTOC has space for identifying only that many files.

Capsule definitions of some common computer terms used in this manual.

| | |
|---|---|
| CPU | (Central Processing Unit) Nucleus of the Model 6 hardware. |
| end-of-job-halt | system halt at the end of every job to give the operator time for any necessary housekeeping chores before beginning the next job. |
| IPL | (Initial Program Load) The process by which the operator loads into core storage the program that controls the operation of the system. |
| KDE | Keyboard Data Entry Utility Program |
| KSE | Keyboard Source Entry Utility Program |
| object library | contains compiled programs, system programs, and routines. |
| object library directory | contains name and address (track and sector) of each object program in the object library. |
| OCL | (Operation Control Language) An OCL statement consists of a keyword and a response. |
| overlay | to erase data on disk by writing new data over it. |
| procedure | sequence of OCL statements in a source library. |
| sector | section of a disk track. Each track is divided into 24 sectors. |
| source library | contains procedures, RPG source programs, and KSE input. |
| source library directory | contains name and address (track and sector) of each source program and procedure in the source library. |
| source statements | program instructions that have not been compiled (translated) into machine language. |
| sysgen | (system generation) Process required to get a system ready to run after installation. |
| system printer | displays OCL statements, utility control statements, job comments, and error codes. (The system printer can also display the normal output of the job being run.) |
| track | Each disk is divided into concentric circles called tracks. |
| VTOC | (Volume Table of Contents) That part of a disk which contains detailed information about every file on the disk. |

# Index

# READER'S COMMENT FORM

| | |
|---|---|
| IBM System/3 Model 6<br>Operation Control Language and<br>Disk Utility Programs<br>Reference Manual | GC21-7516-3 |

## YOUR COMMENTS, PLEASE . . .

Your comments assist us in improving the usefulness of our publications; they are an important part of the input used in preparing updates to the publications. All comments and suggestions become the property of IBM.

Please do not use this form for technical questions about the system or for requests for additional publications; this only delays the response. Instead, direct your inquiries or requests to your IBM representative or to the IBM branch office serving your locality.

Corrections or clarifications needed:

*Page* *Comment*

Please include your name and address in the space below if you wish a reply.

● Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

GC21-7516-3

Fold

Fold

FIRST CLASS
PERMIT NO. 387
ROCHESTER, MINN.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
General Systems Division
Development Laboratory
Publications, Dept. 245
Rochester, Minnesota 55901

Fold

Fold

Cut Along Line

IBM System/3 Printed in U.S.A. GC21-7516-3

IBM

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

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

GC21-7516-3

IBM System/3 Printed in U.S.A. GC21-7516-3

IBM

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

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