IBM System/3 Model 10 Disk System Operation Control Language and Disk Utilities Reference Manual

Program Number 5702—SC1

GC21-7512-4

PREFACE

This manual provides the new programmer with the information he needs to run programs on the IBM System/3 Model 10 Disk System and to use the disk utility programs for doing jobs such as preparing disks for use or updating system libraries. This information is divided into two parts:

- Part I - operation control language (OCL) statements needed to run programs in the Disk System.
- Part II - disk utility programs and utility control statements needed to run them.

Note: In this publication there are some references to support of 64K bytes of main storage. A System/3 Model 10 with a 64K processing unit is available only as an RPQ. Your IBM Marketing Representative can provide information about this.

Related Publications

Publications that are related (not prerequisites) to this one are:

- *IBM System/3 Disk System Introduction,* GC21-7510
- *IBM System/3 Disk System RPG II Reference Manual,* SC21-7504
- *IBM System/3 Disk System Operator's Guide,* GC21-7508
- *IBM System/3 Disk System Halt Procedure Guide,* GC21-7540
- *IBM System/3 Disk System RPG II and System Additional Topics Programmer's Guide,* GC21-7511
- *IBM System/3 Disk Concepts and Planning Guide,* GC21-7571

Fifth Edition (April 1972)

This is a major revision of and obsoletes GC21-7512-3. Additions have been made to support the IBM 5445 Disk Storage Drive. The IBM 1442 Card Read Punch can now replace the IBM 5424 Multi-Function Card Unit (MFCU). The system punch device can be changed using the // PUNCH statement.

This edition applies to version 06, modification level 00 of the IBM System/3 Model 10 Disk System 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.

This manual has been extensively revised and should be reread in its entirety.

Requests for copies of 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.

© Copyright International Business Machines Corporation 1969, 1970, 1971, 1972

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. What each OCL statement is used for (function).

3. Where each OCL statement is placed in relation to others and when it is needed (placement).

4. How each statement must be coded (format).

5. What each statement must contain (contents).

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. OCL STATEMENTS

INTRODUCTION TO OCL STATEMENTS

WHAT IS OCL?

Operation control language (OCL) is your means of communicating with the IBM System/3 Model 10 Disk System. You must write a set of OCL statements for each program you want to run. Based on the information supplied by the OCL statements, the Disk System will load and execute your Disk System programs or perform system utility functions.

You can supply OCL statements in two ways: (1) punch the statements into cards, which are then read by the Disk System; (2) use the printer-keyboard to key the statements directly to the Disk System.

After the Disk System reads a set of OCL statements for a program, it runs the program. When the program ends, the Disk System reads the set of statements for the next program, then runs that program. This procedure is repeated until all OCL statements have been read and the corresponding programs have been run.

The running of your programs is controlled by system control programs. System control programs must be in core storage before your jobs can be run. These programs are located on disk and are brought into storage by a procedure called initial program load (IPL). IPL is performed by the operator when the system is turned on. For more information on IPL, see the *IBM System/3 Disk System Operator's Guide,* GC21-7508.

The DATE statement is part of the IPL process and must be the first statement provided for your program. (See *DATE Statement* in *Statement Descriptions* for more information.)

OCL and the Job Stream

The OCL statements you supply form the basis of the *job stream*. If your program requires the use of data from the system input device (the device used to read OCL statements) your program and that data must follow the corresponding OCL. The job stream, therefore, can contain programs and program data as well as OCL statements. Figure 1 is an example of a card input job stream.

You can also store sets of OCL statements for your programs outside of the job stream in a source library on disk. These sets are called *procedures*. You can instruct the system to merge procedures into the job stream. The ability to store sets of frequently used OCL statements on disk makes it possible to avoid recoding the statements every time they are used. (See *Procedures* under *Statement Descriptions* for more information.)

ORGANIZATION OF PART I

Part I is divided into:

1. *Coding Rules* defines the general contents of the OCL statements and explains the rules for writing the statements.

2. *Statement Descriptions* explains the functions, format, and contents of each OCL statement, and the places in the job stream the statement may be used.

3. *Statement Examples* presents and explains a job stream containing most of the OCL statements.

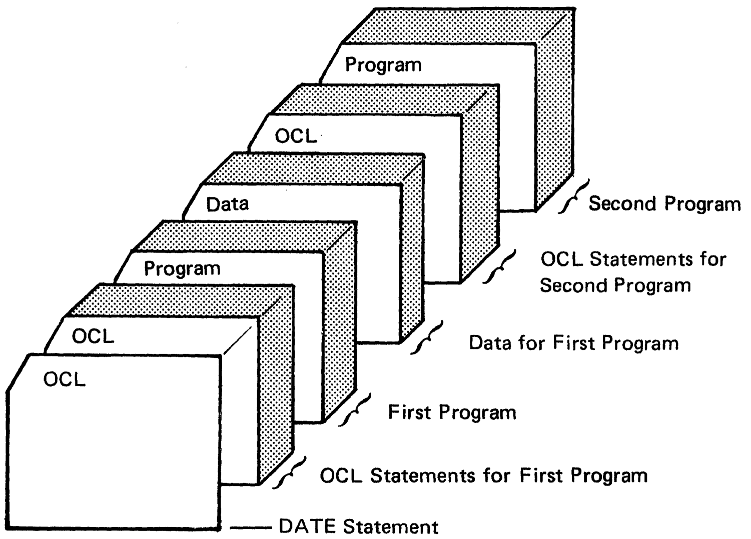


Figure 1. Job Stream

TYPES OF INFORMATION

Operation control language (OCL) statements contain, at most, two types of information: a *statement identifier* and *parameters*. A statement identifier is information that tells one statement from another. A parameter is additional information supplied with the statement identifier. Figure 2 shows the general form of OCL statements.

```
Identifier     Parameter 1, Parameter 2, ..., Parameter n
```

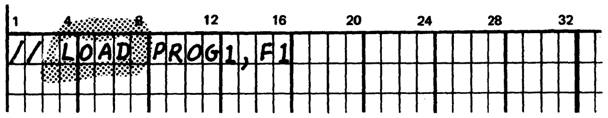
Figure 2. General Form of OCL Statements

Statement Identifiers

Every OCL statement needs a statement identifier. The identifiers are as follows:

DATE	FORMS	* (asterisk)
LOAD	LOG	PAUSE
RUN	READER	/&
SWITCH	PUNCH	FILE
COMPILE	NOHALT	CALL
IMAGE	HALT	PARTITION

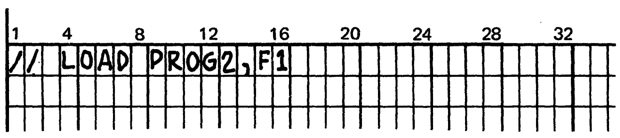
LOAD is an example of a statement identifier.



```
// LOAD PROG1,F1
```

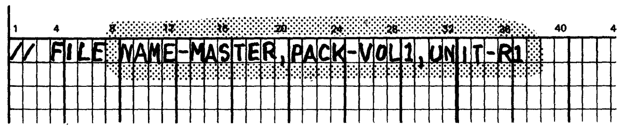
Parameters

Some statements need parameters. Others do not. (See *Statement Descriptions* for an explanation of the statements which need parameters.) Parameters can be either *codes* or *data*. A code is a word or group of characters that has a certain meaning. Data is information such as the names, locations, and lengths of files on disk. (See *Statement Descriptions* for data and code restrictions on parameters.) In the following example, PROG2 is the name of an RPG II object program, and F1 is a code that stands for the fixed disk on drive one. PROG2 is a data parameter and F1 is a code parameter.



```
// LOAD PROG2,F1
```

Some statements require certain words in parameters to tell one parameter from another. The words are called *keywords*. Parameters containing keywords are called *keyword parameters*. In Figure 3, NAME-MASTER, PACK-VOL1, and UNIT-R1 are keyword parameters. NAME, PACK, and UNIT are keywords. MASTER and VOL1 are data parameters. R1 is a code parameter. There should always be a hyphen between the keyword and the code or data parameter.



```
// FILE NAME-MASTER,PACK-VOL1,UNIT-R1
```

Figure 3. Keyword Parameters

GENERAL CODING RULES

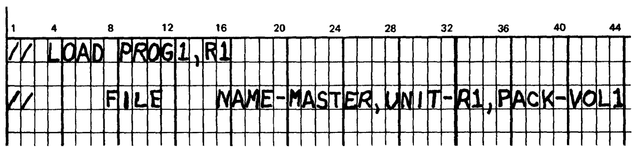
In Part 1 of this book, the numbers that appear above statement formats and examples indicate the card columns or line positions occupied by the statements. In statement formats, special characters, such as //, and words written in capital letters are information that must be used exactly as shown. Words written in small letters, such as code, program-name, and unit, represent information that you must supply.

Statements Beginning with //

The rules for coding the statements are as follows (the term position refers to either card column or line position):

- Place the // in positions 1 and 2.
- Leave one or more blanks between the // and the word that forms the statement identifier (LOAD, RUN, CALL, etc.).
- Leave one or more blanks between the end of the statement identifier and the first parameter.
- If you need more than one parameter, use a comma to separate them. No blanks are allowed within or between parameters. (For the exception to this rule, see the description for the HIKEY parameter under *FILE Statement.*) Anything following the first blank is considered a comment (see *Comments*).
- If you are writing keyword parameters, place the keyword first and use a hyphen to separate the keyword from the code or data parameter.
- If the parameter is not a keyword parameter, write the parameters in the order in which they are discussed in this manual.

Figure 4 illustrates the coding rules. The statement identifiers are LOAD and FILE. The parameters are PROG1, R1, NAME-MASTER, UNIT-R1, and PACK-VOL1. The last three parameters are keyword parameters.



```
// LOAD PROG1,R1
//     FILE    NAME-MASTER,UNIT-R1,PACK-VOL1
```

Figure 4. Illustration of General Coding Rules

Statements Beginning with Other Than //

* and /& statements do not require // preceding them when coded. (See *Statement Descriptions* for * and /& statements.)

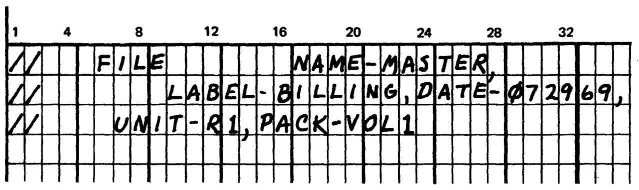
Continuation

All OCL statements except FILE must not exceed 96 characters, including blanks and comments. Because of the large number of parameters possible in a FILE statement, you can use two or more cards or lines for those statements. Each card or line you use must not exceed 96 characters. (Data for the IMAGE statement requires continuation for the cards or lines containing the chain image characters, but the data follows different continuation rules. See *IMAGE Statement* under *Statement Descriptions* for more information.)

The continuation rules are as follows:

- Place a comma after the last parameter in every card or line except the last. The comma, followed by a blank, tells the system that the statement is continued in the next card or line.
- Begin each new card or line with a // in positions 1 and 2.
- Leave one or more blanks between the // and the first parameter in the card or line.

Figure 5 illustrates the continuation rules.



```
//   FILE          NAME-MASTER,
//       LABEL-BILLING,DATE-Ø72969,
//    UNIT-R1,PACK-VOL1
```

Figure 5. Illustration of Continuation Rules

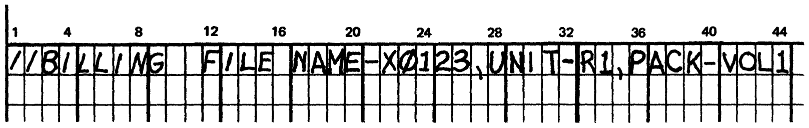
Comments

You can include comments in the following places in your statements:

- Following the // in statements beginning with //. Begin the comment in position 3, immediately following the //. You can use up to eight characters without blanks. Leave one or more blanks between the comment and the word forming the statement identifier. Figure 6 contains such a comment. The word BILLING is the comment.

- After the last parameter. Leave one or more blanks between the last parameter and your comment. The comment can be any combination of characters. If the statement is continued in subsequent cards or lines, you can place comments after the last parameter in any of the cards or lines.

In addition to writing comments within your OCL statements, you can include whole cards or lines of comments. The OCL comment statement is provided for that purpose. (See * *(Comment) Statements* under *Statement Descriptions* for more information.)



```
1  4   8   12  16  20  24  28  32  36  40  44
//BILLING FILE NAME-XØ123,UNIT-R1,PACK-VOL1
```

Figure 6. Comment Following //

STATEMENT DESCRIPTIONS

Each OCL statement is described separately in this section. The following information is given for each statement:

1. The function of the statement.

2. The placement of the statement in regard to other statements and the circumstances under which the statement is needed.

3. The format of the statement.

4. The contents of the statement, explaining the parameters that can be used in the statement.

Figure 7 gives the function, placement, and restrictions on use for each OCL statement.

Figure 8 describes the contents of the OCL statements. It is meant for reference only. If you are not familiar with an entry, or you do not know when to use or omit it, refer to the proper statement in the remainder of this section.

When using Figure 8, remember that words written in small letters such as filename or value require a choice on your part, depending on the functions you want the statement to perform. Refer to Figure 8 to see which parameters are available. Those parameters that are capitalized must be coded along with the data or code parameter.

STATEMENT	FUNCTION	PLACEMENT		RESTRICTIONS ON USE
		STATEMENT APPEARS IN JOB STREAM	STATEMENT APPEARS IN A PROCEDURE	
// DATE	Supplies the system with a date, this date is given to disk files being created.	Must follow LOAD or CALL statement and precede the RUN statement except at IPL time, when it must precede the first LOAD or CALL statement.	Must follow the LOAD statement and precede the RUN statement (if RUN is used).	Must be supplied during the Initial Program Load. The effect of the statement is for that job only.
// LOAD *	Indicates that the object program will be loaded from the system input device following the RUN statement.	Must precede the RUN statement	Not allowed in a procedure.	LOAD * cannot be used in program level 2.
// LOAD	Identifies the program to be run and indicates the disk that contains the object library from which it is to be loaded.	Must precede the RUN statement.	Must be the first // statement.	
// RUN	Indicates the end of the OCL statements for a program and tells system to run the program.	Must be the last OCL statement.	May be the last statement.	Required in the job stream for each program which is to be run.
// SWITCH	Used to set one or more external indicators on or off or leave the indicator as it is.	Must follow LOAD or CALL statement and precede the RUN statement.	Must follow the LOAD statement and precede the RUN statement (if RUN is used).	
// COMPILE	Tells the system where the source program to be compiled is located and where to place the object program.	Must follow LOAD or CALL statement and precede the RUN statement.	Must follow the LOAD statement and precede the RUN statement (if RUN is used).	
// IMAGE	Tells the system to replace the chain-image area with characters indicated in the following data cards or characters keyed in or read from source library.	Anywhere among the OCL statements.	Must precede the RUN statement (if RUN is used).	Required if the printer chain has been changed.
// FORMS	Instructs the system to change the number of lines printed per page.	Anywhere among the OCL statements.	Must precede the RUN statement (if RUN is used).	
// LOG	Instructs system to start or stop printing OCL statements and codes and indicates the device to be used to print them.	Anywhere among the OCL statements.	Must precede the RUN statement (if RUN is used).	Device cannot be specified in program level 2.

Figure 7 (Part 1 of 2). Table of OCL Statements

STATEMENT	FUNCTION	PLACEMENT		RESTRICTIONS ON USE
		STATEMENT APPEARS IN JOB STREAM	STATEMENT APPEARS IN A PROCEDURE	
// READER	Changes the system input device used to read OCL statements.	Must precede LOAD or CALL statement or follow the RUN statement and precede the next LOAD or CALL statement	Must precede the LOAD statement (if LOAD is used).	Can be used in job stream only.
// PUNCH	Enables you to change the system punch device.	Anywhere among the OCL statements.	Must precede the RUN statement.	
// NOHALT	Instructs system to continue without stopping when a program ends.	Anywhere among the OCL statements.	Must precede the RUN statement (if RUN is used).	Ignored in program level 2.
// HALT	Instructs system to halt when program ends; cancels the effect of the NOHALT statement.	Anywhere among the OCL statements.	Must precede the RUN statement (if RUN is used).	Ignored in program level 2.
*(Comment)	Used to explain the job or give the operator instructions; does not affect the program in operation.	Anywhere.	Anywhere.	
// PAUSE	Tells the program to stop in order to give the operator time to perform a function. Operator must restart program.	Anywhere among the OCL statements.	Must precede the RUN statement (if RUN is used).	
/&	Provides OCL security from previous job.	Recommended as the first statement of a job.	Not allowed in a procedure.	Can be used in the job stream only.
// FILE	Supplies information about the file to the system.	Must follow LOAD or CALL statement and precede the RUN statement.	Must follow the LOAD statement and precede the RUN statement (if RUN is used).	Required for every new file created and existing files being used.
// CALL	Identifies procedure to be merged into job stream and the disk containing the source library from which to read the procedure.	Must precede the RUN statement.	Indicates chained procedures.	Can be no more than nine levels of nested chained procedures.
// PARTITION	Guarantees a minimum size to level 2 for a program in that level.	Anywhere, among the OCL statements.	Must precede the RUN statement (if RUN is used).	Cannot be submitted in program level 2 or when program level 2 is processing.

Figure 7 (Part 2 of 2). Table of OCL Statements

STATEMENT	PARAMETER	CODE	MEANING OF CODE
// DATE	date	mmddyy or ddmmyy	System date or date within a set of statements
// LOAD	asterisk	*	Program is to be loaded from the system input device
	program name	name	Name of program that is to be loaded from disk
	unit		Object library resides upon:
		R1	Removable disk on drive one
		R2	Removable disk on drive two
		F1	Fixed disk on drive one
		F2	Fixed disk on drive two
// RUN	none		
// SWITCH	indicator-settings	Refer to *SWITCH Statement* under *Statement Descriptions*	
// COMPILE	SOURCE	SOURCE-name	Name of source program
	UNIT	UNIT-R1 R2 F1 F2	Where disk that contains the source library is located (the meanings of the unit codes are the same as for LOAD)
	OBJECT	OBJECT-R1 R2 F1 F2	Where to place the object program (the meanings of the unit codes are the same as for LOAD)
// IMAGE	format	HEX	To indicate characters from cards are in hexadecimal form
		CHAR	To indicate characters from cards are in EBCDIC form
		MEM	To indicate characters are from the source library
	number	value	Number of new characters
	name	name	Identifies the characters in the library
	unit	R1 R2 F1 F2	Where the disk that contains the library is located (the meanings of the unit codes are the same as for LOAD)
// FORMS	DEVICE	DEVICE-name	Indicates which printer is used
	LINES	LINES-value	Indicates number of lines to be printed per page
// LOG	code	CONSOLE	Use printer-keyboard as logging device
		PRINTER	Use printer as logging device
		OFF	Stop printing
		ON	Start printing
// READER	system input device	CONSOLE	Printer-keyboard
		MFCU2	Secondary hopper of MFCU
		MFCU1	Primary hopper of MFCU
		1442	Card Read/Punch
// PUNCH	system punch device	MFCU2	Secondary hopper of MFCU
		MFCU1	Primary hopper of MFCU
		1442	Card Read/Punch

Figure 8 (Part 1 of 2). Table of Parameters

STATEMENT	PARAMETER	CODE	MEANING OF CODE
// NO HALT	none		
// HALT	none		
* (Comment)	none		
// PAUSE	none		
/&	none		
// FILE	NAME	NAME-filename	Name of the program uses to refer to the file
	UNIT	UNIT-R1 R2 F1 F2	Where the 5444 disk that contains or will contain the file is located (the meanings of the unit codes are the same as for LOAD)
		D1 D2	Where the 5445 disk that contains or will contain the file is located.
	PACK	PACK-name	Name of disk that contains or will contain the file
	LABEL	LABEL-filename	Name by which your file is identified on disk
	RECORDS or TRACKS	RECORDS-number of TRACKS-number	Amount of space needed on a disk for a file
	LOCATION	LOCATION-track number	Number of track on which file begins or is to begin (5444 disk only)
		LOCATION-cylinder number	Cylinder number on which file begins or is to begin. Track assumed zero (5445 disk only).
		LOCATION-cylinder number/track number	Cylinder number, track number on which file begins or is to begin (5445 disk only).
		LOCATION-filename	Filename of a split cylinder file that is the first split cylinder file in a group, or is an already existing split cylinder file. (5445 disk only). For further discussion see *Split Cylinder Files.*
	RETAIN	RETAIN-T S P A	Temporary file Scratch file Permanent file Reactivate scratch file
	DATE	DATE-mmddyy ddmmyy	Tells the system the date the file was created
	HIKEY	HIKEY-'highest key fields allowed'	List of highest key fields allowed on each pack
	SPLIT	SPLIT-tracks/cylinders or SPLIT-tracks	The number of tracks per cylinder needed for the split cylinder file; the number of cylinders needed for a group of split cylinder files (5445 disk only). For further discussion see *Split Cylinder Files.*
// CALL	procedure name	name	Name that identifies the procedure in the source library
	unit	R1 R2 F1 F2	Where the disk containing the procedure is located (the meanings of the unit codes are the same as for LOAD)
// PARTITION	size	value	Minimum size of program level 2 in decimal bytes

Figure 8 (Part 2 of 2). Table of Parameters

DATE STATEMENT

Function

The DATE statement gives the Disk System a date, called the *system date.* The system date is referred to by RPG II field names UDATE, UMONTH, UDAY, and UYEAR. The preceding field names can also be used when referring to the date given to the disk files when they were created.

A DATE statement within the set of statements for a program changes the system date, but only for that program. When the program ends, the date supplied in the DATE statement at IPL time is again used. There can only be one DATE statement per job.

Placement

A DATE statement is always required during Initial Program Load (IPL). It is the only OCL statement required by the system at that time.

A DATE statement can also appear within any of the sets of statements for your programs. The DATE statement must follow the LOAD or CALL statement and precede the RUN statement.

Format

// DATE date

Contents

The system date can be in either of two forms: month-day-year (mmddyy) or day-month-year (ddmmyy). You must specify the form at System Generation time. (See *IBM System/3 Disk System Operator's Guide,* GC21-7508, for more information on System Generation.) The date you specify must be in that form.

Example

The date can be written with or without punctuation. For example, July 25, 1970, could be specified in any one of the following ways:
07-25-70
25-07-70
072570
250770

Month, day, and year must each be 2-digit numbers but lead zeros in month and day may be omitted when punctuation is used (7-25-70 or 25-7-70). In the punctuated form, any characters except commas, quotes, numbers and blanks can be used as punctuation.

LOAD STATEMENT

Function

The LOAD statement identifies the program to be run and indicates whether the program will be loaded from the system input device or disk.

Placement

One LOAD statement is required within each of the sets of statements for your programs. If the set of statements appears on the job stream, the only requirement for the LOAD statement is that it must precede the RUN statement. In procedures, the LOAD statement must be the first // statement. (For more information about procedures, see *Procedures* in this section)

Format

The LOAD statement has two formats. The first format is used for object programs loaded from the system input device and cannot be used in a procedure. The second format is used for programs loaded from disk.

```
// LOAD *
// LOAD program-name,unit
```

Contents

Asterisk: An asterisk indicates that the object program will be loaded from the system input device. Program-name and unit parameters must not be included. The cards or lines that contain the program must follow the RUN statement for the program and must be followed by /* or /& to signify the end of the program. LOAD * cannot be used in programming level 2 (see *Using OCL, Loading Programs in a DPF Environment,* for more information on dual programming).

Program-name: The program-name parameter is the name used on disk to identify the program.

The names you must use for your RPG II programs depend on the way the programs were placed on disk. One way includes an RPG II compiler option. You can specify, in the RPG II Control Card specifications, that your RPG II program be placed on disk immediately after it is compiled. The name you supply in columns 75-80 of the Control Card specifications is the name used to identify the program. If you left columns 75-80 blank, the name RPGOBJ is used.

Another way to place your RPG II Program on disk is by using the Library Maintenance program. If you used that program, the program-name you supplied in the Library Maintenance control statements is the name used to identify your program. (For more information, see *Library Maintenance* in Part II of this book.)

The Disk System programs are identified by the following names:

Program	*Name*
Alternate Track Assignment	$ALT
Alternate Track Rebuild	$BUILD
Assembler	$ASSEM
Data Recording	$DREC
Data Verifying	$DVER
Disk Copy/Dump	$COPY
Disk Initialization	$INIT
Disk Sort	$DSORT
File and Volume Label Display	$LABEL
File Delete	$DELET
Library Maintenance	$MAINT
List	$CLIST
MFCU Sort/Collate	$CSORT
Reproduce and Interpret	$REPRO
Restart	$$RSTR
RPG II Compiler	$RPG
80-96 Conversion	$CNVRT

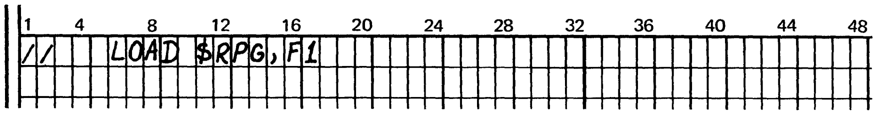
Unit: The unit parameter is a code. It indicates where the disk that contains the program is located. The codes are as follows:

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 unit parameter is required because your programs can be on any of the disks on your disk unit. The disk area containing your object program is called an object library. You can create an object library on any of the disks on your disk unit by using the Library Maintenance program. (See *Library Maintenance* in Part II of this manual.)

Example

In the following sample LOAD statement, $RPG is the name that identifies the RPG II Compiler.



```
1  4   8   12  16  20  24  28  32  36  40  44  48
//   LOAD $RPG,F1
```

F1 is the code indicating the fixed disk on drive one, where the compiler would be located in this case.

RUN STATEMENT

Function	The RUN statement indicates the end of the OCL statements for a program. After the system reads the RUN statement, it runs the program.
Placement	A RUN statement is needed for each of the programs you want the system to run. In the job stream, it must be the last statement within each of the sets of OCL statements for your programs. It can also be the last OCL statement in a procedure. (For more information about procedures, see *Procedures* in this section.)
Format	// RUN
Contents	None

SWITCH STATEMENT

Function

The purpose of the SWITCH statement is to set one or more RPG II external indicators on or off. The indicators are always off after the operator uses the IPL procedure to start the system. If a SWITCH statement is used to set an indicator on, the indicator remains on until another SWITCH statement sets it off, or until the operator again uses the IPL procedure to start the system. There can be only one SWITCH statement per job.

Placement

The SWITCH statement can appear within any of the sets of statements for your programs. The only requirements for the SWITCH statement are that it must follow the LOAD or CALL statement and precede the RUN statement.

Format

```
// SWITCH indicator-settings
```

Contents

Indicator-settings: The indicator-settings parameter is a code that consists of eight characters, one for each of the eight external indicators (U1-U8). The first, or leftmost, character gives the setting of indicator U1; the second character gives the setting of U2; and so on.

The code must always contain eight characters. For each indicator, one of the following characters must be used:

Character	*Meaning*
0	Set the indicator off
1	Set the indicator on
X	Leave the indicator as it is

Example

The code 1X0110XX would cause the following results:

Indicator	*Result*
U1	Set on
U2	Unaffected
U3	Set off
U4	Set on
U5	Set on
U6	Set off
U7	Unaffected
U8	Unaffected

COMPILE STATEMENT

Function

The COMPILE statement tells the system two things: (1) where the source program to be compiled is located if it is coming from a disk source library; (2) where the object program is to be placed. (An object program is a source program which has been compiled or translated into machine language.)

Placement

The COMPILE statement must be within the set of OCL statements that apply to the compilation. The COMPILE statement must follow the LOAD or CALL statement and precede the RUN statement.

Format

// COMPILE parameters

Contents

All the parameters are keyword parameters (keywords are in capital letters). The keywords are: SOURCE, UNIT, and OBJECT.

SOURCE: The SOURCE parameter tells the system the name of the source program. The keyword SOURCE must be followed by the name of the source program on disk. The name is the name by which the source program is identified on disk in the source library. (For more information concerning the source library see *CALL Statement* in this section.)

The only way you can place source programs in a source library is by using the Library Maintenance program. The program name you supply in Library Maintenance control statements is the name used to identify the source program in the library. (For more information, see *Library Maintenance* in Part II of this manual.)

If the SOURCE parameter is not used, the source program is assumed to be in the job stream following the RUN statement.

The SOURCE parameter must always be accompanied by the UNIT parameter.

UNIT: The UNIT parameter is used only when the SOURCE parameter is used.

The UNIT parameter is a code indicating where the disk that contains the source library is located. The codes are as follows:

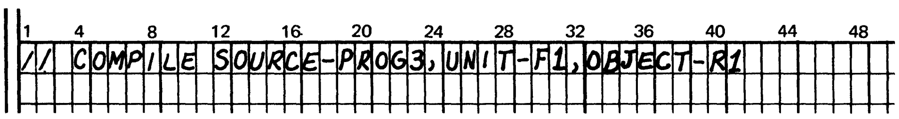
Code	*Meaning*
R1	Removable disk on drive one
F1	Fixed disk on drive one
R2	Removable disk on drive two
F2	Fixed disk on drive two

OBJECT: The OBJECT parameter tells the system where to place the object program. The OBJECT parameter may be specified without using the SOURCE and UNIT parameters. The codes which are used to indicate the disk unit on which the object program is to be placed are R1, F2, R2, or F2.

Note: If the OBJECT parameter is omitted, it is assumed that the object program is to be placed on the same disk as the compiler. The object program name is a function of the RPG II Compiler. (For more information see the *IBM System/3 Disk System RPG II Reference Manual,* SC21-7504.)

Example

The following sample COMPILE statement tells the system that the source program with the name PROG3 is located on the fixed disk on drive one (F1).



```
1  4   8   12  16  20  24  28  32  36  40  44  48
// COMPILE SOURCE-PROG3,UNIT-F1,OBJECT-R1
```

The parameter, OBJECT-R1, tells the system to place the object program on the removable disk on drive one.

IMAGE STATEMENT

Function

To operate correctly, the printer requires characters matching those on the printer chain to be in a special area of core storage called the chain-image area. When you replace the printer chain with one having different characters, you must also change the contents of the chain-image area.

The IMAGE statement instructs the system to replace the contents of the chain-image area with the characters indicated by the statement. The characters can be entered from the system input device, or contained in a source library on disk. The effect of the IMAGE statement is temporary and the system chain image is returned to the chain-image area when IPL occurs.

Placement

The IMAGE statement can appear anywhere among the OCL statements. In a procedure, it must precede the RUN statement.

Format

// IMAGE parameters

Contents

The IMAGE statement tells the system either of two things: (1) the new chain characters are to be read from the system input device; or (2) the new chain characters are to be read from the source library.

The IMAGE parameters are:

- format-*HEX, CHAR, or MEM*
- number-value
- name-name
- unit-code

(Coding only HEX, CHAR, or MEM is preferable for format but HEXADECIMAL, CHARACTER, or MEMBER can be coded.)

Characters From the System Input Device

If you wish to indicate that the new chain characters are to be read from the system input device, use the following parameters:

Format: Use the word CHAR to indicate that the characters are in EBCDIC form. Use the word HEX to indicate that the characters are in hexadecimal form.

Number: The number parameter must be used with HEX and CHAR. It must be a value which is equal to the number of columns or line positions in the data cards or the data keyed in following the IMAGE statement that contains the new characters. This number must not exceed 240 when the characters are hexadecimal, 120 when characters are EBCDIC. The name and unit parameters must not be coded.

Following are the rules for punching or keying the new characters:

1. The characters must begin in column or line position 1.

2. Consecutive card columns or line positions must be used; however, only the first 80 columns or line positions of the card or line can be used. Hexadecimal requires an even number of columns or line positions, two per character.

3. To continue the characters on another card or line begin the characters in column or line position 1.

Characters From the Source Library on Disk

To indicate that new chain characters are to be read from the source library on disk, the format parameter must specify the word MEM.

The following parameters must also be included:

Name: The name parameter identifies the source member containing the characters in the library. The only way you can place the characters in a source library is by using the Library Maintenance program. The name you supply in Library Maintenance control statements is the name used to identify the characters in the source library.

Unit: The unit parameter must be used with the name parameter. It is used to tell the system where the disk containing the source library is located on the disk unit. The codes which are used 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

Example

The IMAGE statement in example A tells the system that the new characters are on data cards or keyed in. The format parameter indicates that new characters are in hexadecimal form; the number parameter indicates that there are 120 columns or line positions containing the new characters.

In example B, the new characters, on data cards or keyed in, are in EBCDIC. The number parameter indicates that there are 48 columns or line positions containing the new characters.

Example C tells the system that the new characters are to be read from the source library on disk. The format parameter indicates that the new chain characters are in the source library. The name parameter indicates that the characters were named CHAIN in the source library. The unit parameter indicates that the source library containing them is on the removable disk on drive one (R1). Examples of the member specified in example C are the data portions of examples A and B. The member itself requires a // IMAGE card with the characters either in hexadecimal or EBCDIC. The number of columns or line positions containing the characters must also be specified.

(See *Library Maintenance* in Part II for restrictions on the name used in coding MEM.)

Ⓐ

```
1  4   8   12  16  20  24  28  32  36  40  44  48  52  56  60  64
// IMAGE HEX,12Ø
F1F2F3F4F5F6F7F8F9FØE7E861E2E3E4E5E64F7A6D7F6B7ED1D2D3D4D5D6
D7D8D96ØE94DC1C2C3C4C5C6C7C8C94E4B5D6C5B5C7B5Ø7C4C5E5F7D6F6E
```

Ⓑ

```
1  4   8   12  16  20  24  28  32  36  40  44  48  52  56  60  64
// IMAGE CHAR,48
1234567890#@/STUVWXYZ&,%JKLMNOPQR-$*ABCDEFGHI+.'
```

Ⓒ

```
1  4   8   12  16  20  24  28  32  36  40  44  48  52  56  60  64
// IMAGE MEM,CHAIN,R1
```

FORMS STATEMENT

Function

The FORMS statement enables you to change the number of lines that the printer will print per page. The printer automatically assumes the number of lines per page specified at system generation time unless a FORMS statement is used or an RPG II program specifies some other number. This number of lines is effective until another FORMS statement is used or an RPG II program with a line counter specification is run. (See *IBM System/3 Disk System RPG II Reference Manual,* SC21-7504.)

Placement

The FORMS statement can be placed anywhere among the OCL statements. In a procedure it must precede the RUN statement.

Format

// FORMS parameters

Contents

All of the parameters are keyword parameters (keywords are in capital letters). The parameters are as follows:

- DEVICE-name
- LINES-value

DEVICE: The keyword for this parameter is DEVICE. It must be followed by the name of the printing device. The name of the printing device is 5203L or 5203. You may omit the DEVICE parameter entirely.

LINES: The LINES parameter is used to indicate the number of lines per page. The maximum number of lines that can be specified per page is 112. The LINES parameter remains in effect until either an IPL is performed or another FORMS statement for the same device is read. If a line counter specification is used in an RPG II program, it remains in effect only for the duration of the program.

Example

In the following FORMS statement, the system is using the left carriage of the 5203 Printer. The statement tells the system that the forms length is 88 lines per page.

```
1  4   8   12  16  20  24  28  32  36  40  44  48
// FORMS DEVICE-5203L,LINES-88
```

LOG STATEMENT

Function

OCL statements and message codes are printed on the printer-keyboard. If your system has no printer-keyboard, the statements and codes are printed on the printer. The device used to print OCL statements and message codes is called the *logging device.* If you want to change the logging device, or specify whether or not the statements and codes are to be printed, you must use a LOG statement.

The LOG statement tells the system to do one of four things:

- Use the printer as the logging device
- Use the printer-keyboard as the logging device
- Stop printing OCL statements and message codes
- Start printing OCL statements and message codes

Placement

You can use the LOG statement within any of the sets of OCL statements for your programs. In a procedure it must precede the RUN statement.

Format

```
// LOG code
```

Contents

Four codes can be used as parameters. The codes are as follows:

Code	*Meaning*
CONSOLE	Use printer-keyboard as logging device
PRINTER	Use printer as logging device
OFF	Stop printing
ON	Start printing

Only one code can be used in one LOG statement. The starting of the logging device is implied when coding CONSOLE or PRINTER.

When the system reads a LOG statement that contains the OFF code, it stops printing OCL statements and message codes. The only way you can instruct the system to start printing them again is by using a LOG statement that contains the ON, PRINTER, or CONSOLE code. When ON is specified printing resumes on the last logging device specified.

READER STATEMENT

Function

The device used to read OCL statements is called the *system input device.* The system assumes that the system input device is the primary hopper of the MFCU. You must use a READER statement if you want to use the printer-keyboard, secondary hopper of the MFCU, or the 1442 Card Read/ Punch as the system input device.

Placement

The READER statement must not come between the LOAD or CALL statement and a RUN statement. The READER statement must precede the initial LOAD or CALL statement or follow the RUN statement, preceding the next LOAD or CALL statement. If you use the READER statement to change the system input device, the device you specify is used to read source programs, control statements, or OCL statements. Changing the system input device affects the placement of source programs and control statements as well as OCL statements.

You must place the READER statement in the current system input device.

Format

// READER system input device

Contents

The codes are:

Code	*Meaning*
CONSOLE	Printer-keyboard
MFCU2	Secondary Hopper of the MFCU
MFCU1	Primary Hopper of the MFCU
1442	Card Read/Punch

PUNCH STATEMENT

Function The PUNCH statement enables you to change the system punch device.

Placement The PUNCH statement can be placed anywhere among the OCL statements. In a procedure it must precede the RUN statement.

Format // PUNCH punch device

Contents Three codes can be used as parameters. They are:

Code	*Meaning*
MFCU1	Primary Hopper of the MFCU
MFCU2	Secondary Hopper of the MFCU
1442	Card Read/Punch

NOHALT STATEMENT

Function

Normally the system halts when a program ends. The NOHALT statement tells the system to continue by reading the next set of OCL statements without stopping, when a program ends. The effect of this statement lasts until the system reads a HALT statement or an IPL occurs. The effect of the NOHALT statement is ignored temporarily when an abnormal job halt occurs. The system reverts to the NOHALT mode after a response.

Placement

A NOHALT statement can be placed anywhere among the OCL statements. In a procedure it must precede the RUN statement. The NOHALT statement is ignored if loaded in program level 2.

Format

// NOHALT

Contents

None

HALT STATEMENT

Function

The HALT statement tells the system to halt when a program ends. The operator can restart the system when he is ready, and the system continues reading the next OCL statements.

The HALT statement is needed only if you want to cancel the effect of a NOHALT statement.

Placement

A HALT statement can be placed anywhere among the OCL statements. In a procedure it must precede the RUN statement. The HALT statement is ignored if loaded in program level 2.

Format

// HALT

Contents

None

*(COMMENT) STATEMENTS

Function

Comment statements are commonly used either to explain the jobs or to give the operator instructions. Operator instructions are usually given in connection with a PAUSE statement. Comment statements are printed along with the other OCL statements. They have no other effect on the system.

Placement

You can include, in OCL statements, special statements that contain only comments. Comment statements must contain as asterisk (*) in column 1. They can be placed anywhere among the OCL statements in either a job stream or a procedure.

Format

*comment

Contents

The comment can be any combination of words and characters. The only requirement is that an asterisk (*) be in column 1.

PAUSE STATEMENT

Function	The PAUSE statement causes a halt. It usually is used to give the operator time to prepare for the next program. He might, for example, have to place removable disks on the disk units or insert special forms into the printer. Comment statements that give the operator instructions usually precede PAUSE statements. When the operator is ready, he can restart the system. The system continues reading the OCL statements that follow the PAUSE statement.
Placement	PAUSE statements can be placed anywhere among the OCL statements. In a procedure it must follow the LOAD statement and precede the RUN statement.
Format	// PAUSE
Contents	None

/& STATEMENT

Function	/& statements are used as a precautionary measure. Placed in front of your OCL set, a /& statement signals the system that a new set of OCL statements is coming. It prevents your statements from being read as a part of the preceding set of statements or data. Any attempt to read more data from that device will be blocked.
Placement	/& statements are not required. It is recommended, however, that you use them as the first statement in each of the sets of OCL statements for your programs. They are not allowed in a procedure.
Format	/&
Contents	None

/* STATEMENT

Function	/* statements indicate the end of a data file read in from a card reader or console.
Placement	A /* statement should be the last card of an input data file or program deck.
Format	/*
Contents	None

FILE STATEMENT

Function

The FILE statement supplies the system with information about disk files. The system uses this information to read records from and write records on disk.

Placement

You must supply a FILE statement for each of the new disk files that your programs create, and for each of the existing disk files that your programs use. The FILE statement must follow the LOAD or CALL statement and precede the RUN statement.

Format

// FILE parameters

Contents

All of the parameters are keyword parameters. The parameters are as follows (keywords are in capital letters):

- NAME-filename (in program)
- UNIT-code
- PACK-name
- LABEL-filename (on disk)
- RECORDS-number or TRACKS -number
- LOCATION- $\left\{\begin{array}{l}\text{track number (5444 disk only)}\\ \left.\begin{array}{l}\text{cylinder number}\\ \text{cylinder number/track number}\\ \text{filename}\end{array}\right\}\ \text{5445 only}\end{array}\right.$
- RETAIN-code
- DATE-date
- HIKEY-highest allowed key fields (on pack)
- SPLIT-number of tracks per cylinder or both the number of tracks per cylinder and the number of cylinders (5445 disk only)

The NAME, PACK, and UNIT parameters are always required. The others are required only under certain conditions.

NAME: The NAME parameter is always needed. It tells the system the name that your program uses to refer to the file. The NAME parameter must be placed on the first card or line if two or more cards or lines are used for the FILE statement. (See *General Coding Rules* for rules on continuation.)

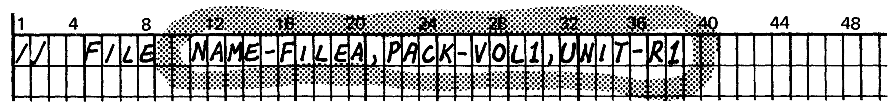
If you are executing a program compiled by RPG II that uses a disk file, the filename in this parameter must be the same name used on the File Description specifications at compile time.

For some of the programs, you must use specific names for certain files.

Program	*File*	*Name*
Disk Copy/Dump	Input	COPYIN
	Output	COPYO
Disk Sort	Input	INPUT
	Work	WORK
	Output	OUTPUT
Assembler	Input	$SOURCE
	Work	$WORK
	Output	$WORK 2 (optional)
RPG II Compiler	Input	$SOURCE
	Work	$WORK

These files must be on a 5444 disk device. (Assembler and RPG II Compiler files)

The keyword for the parameter is NAME. It must be followed by the filename used by the program. The name can be any combination of characters except commas, quotes, or blanks. The first character must be alphabetic. The number of characters must not exceed 8. The following example shows how the NAME parameter for a file named FILEA would be coded.


```
// FILE NAME-FILEA,PACK-VOL1,UNIT-R1
```

UNIT: The UNIT parameter is always needed. It tells the system the disk that contains or will contain the file. The keyboard for this parameter is UNIT. It must be followed by a code that indicates the unit. The codes are as follows:

R1	Removable disk on 5444 drive one
F1	Fixed disk on 5444 drive one
R2	Removable disk on 5444 drive two
F2	Fixed disk on 5444 drive two
D1	Removable disk on 5445 drive one
D2	Removable disk on 5445 drive two

The previous example shows how the UNIT parameter for a file located on the removable disk on drive one would be coded.

PACK: The PACK parameter is always needed for disk files. It tells the system the name of the disk that contains or will contain the file. The system checks this name to ensure that the proper disk is being used. (For information about how a disk is given a name, see *Disk Initialization* in Part II of this manual).

The keyword for this parameter is PACK. It must be followed by the name of the disk. Figure 15 shows how the PACK parameter for a file on a disk named VOL1 would be coded.

LABEL: The LABEL parameter tells the system the name by which your file is identified on disk.

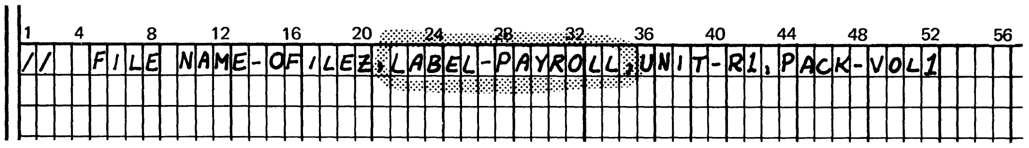
If the file is being created, the name you supply in the LABEL parameter is used to identify the file on disk. If you omit the LABEL parameter from a disk FILE statement, the name from the NAME parameter is used.

If the file is an existing disk file, you must supply a LABEL parameter when the name your program uses to refer to the file differs from the name by which the file is identified on disk.

Several versions of a file can be created on the same disk and be given the same name. If the TRACKS or RECORDS parameter you are using in creating a file is the same as the TRACKS or RECORDS specified for an existing file you must specify LOCATION. You can reference each of these files by its name and date, or by its name and location on disk. Both date and location must be unique for each version. (See Example 2 for an example of how to reference one version of a file.)

If a space parameter (TRACKS or RECORDS) is given when creating another version of an existing file it must be equal to the original value for the existing file.

The keyword for the parameter is LABEL. It must be followed by the name of the file on disk. The name can be any combination of characters except commas, quotes, or blanks. The first character must be alphabetic. The number of characters must not exceed 8. The LABEL parameter for a file named PAYROLL is coded in the following example.


1
4
8
12
16
20
24
28
32
36
40
44
48
52
56

```
//  FILE NAME-OFILEZ,LABEL-PAYROLL,UNIT-R1,PACK-VOL1
```

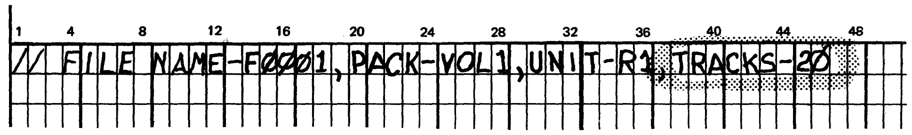
TRACKS or RECORDS: The TRACKS or RECORDS parameter is needed for files that are being created. The parameter tells the system the amount of space needed on disk for the file.

If you use the TRACKS keyword, you specify the number of disk tracks needed for the file.

If you use the RECORDS keyword, you specify the approximate number of records for the file. The total space allocated will be rounded up to full tracks allowing adequate space to accomodate at least the number of records indicated.

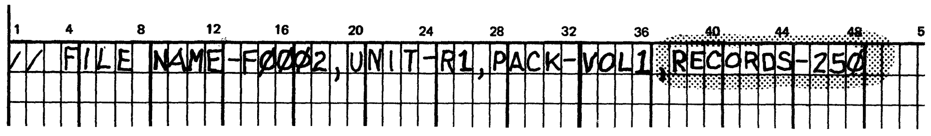
Either of these two keywords, TRACKS or RECORDS, can appear in the FILE statement, but not both. The keyword must be followed by a number indicating the amount of space needed.

If TRACKS is used, the number must be within the range 1-398 if you are using full capacity 5444 disk packs. If you are using half capacity 5444 disk packs, the number must be within the range 1-198. If you are using 5445 disk packs, the number must be in the range of 1-3980. The following example shows how the TRACKS parameter for a file requiring 20 tracks is coded.



```
// FILE NAME-FØØØ1,PACK-VOL1,UNIT-R1,TRACKS-2Ø
```

If RECORDS is used, the number can be up to six digits long. The RECORDS parameter for a file containing 250 records is coded as follows:



```
// FILE NAME-FØØØ2,UNIT-R1,PACK-VOL1,RECORDS-25Ø
```

LOCATION: The LOCATION parameter is not required. It can, however, be used for files that are being created. LOCATION is required when creating several versions of a file or when loading an offline multivolume file to packs which contain other files. (See Example 4.) It can also be used in referencing one of several files housing the same name and same size. LOCATION is not required if sizes differ.

For files that are being created, the parameter tells the system the number of the track on which the file is to begin. If it is omitted, the track is chosen for you.

For files that are being referenced, the parameter tells the system the number of the track on which the file begins. In this case, the system uses the track number to tell one file from another.

The keyword for this parameter is LOCATION. For the 5444 disk the LOCATION format is:

LOCATION-track number

Track number must be between 8-205 (half-capacity disk) or 8-405 (full-capacity disk). Tracks 0-7 are reserved for the system.

For the 5445 disk the LOCATION format is:

LOCATION-cylinder number/track number

Slash is needed to separate cylinder number and track number (when both are specified

Cylinder number must be between 1-199. Cylinder 0 is reserved for the system.

Track number must be between 0-19. Track number 0 is assumed if track number is not specified.

Split cylinder file support on the 5445 disk allows for an additional LOCATION parameter:

LOCATION-file name

For a discussion on how the LOCATION parameter is used when specifying split cylinder files, see *Split Cylinder Files.*

RETAIN: The RETAIN parameter is used to classify files according to their use: scratch, temporary, or permanent.

A scratch file is normally used only once in a program and not retrieved after the program has ended. However, a scratch file can be retrieved if a previous program has defined it as a permanent or temporary file and then redefined it as a scratch file. To change a permanent file to a scratch file you must use a utility program. A temporary file can become a scratch file by using a utility program or by using a RETAIN-S parameter. A RETAIN-A parameter is needed to change a scratch file to a temporary file. A scratch file cannot become a permanent file unless it becomes a temporary file first. A temporary file can be changed to a permanent file only if the file name is changed and copied as a permanent file. The system will overlay a scratch file if the disk pack is full and/or file space is needed by a new file or by a system program.

A temporary file is usually used more than once. The area containing a temporary file can be only given to another file under one of the following conditions:

1. A FILE statement containing the RETAIN-S parameter is supplied for the temporary file. This converts the temporary file to a scratch file.

2. Another file with the same LABEL name is loaded into the exact area occupied by the temporary file but this only changes the data. Space and location parameters are required.

3. The File Delete program is used to delete the file.

The area containing a permanent file cannot be used for any other file until the File Delete program has deleted the permanent file.

A disk file is classified as scratch, temporary, or permanent when it is created. If the RETAIN parameter is omitted from the FILE statement when the file is created, the file is assumed to be a temporary file. The RETAIN parameter may be omitted when accessing an existing file; however, RETAIN-A must be coded to reactivate a scratch file.

The keyword for the parameter is RETAIN. It must be followed by a code that indicates the classifications of the file. The codes are:

Code	*Meaning*
S	Scratch file
T	Temporary file
P	Permanent file
A	Reactivate scratch file

The RETAIN parameter for a permanent file is coded as follows:

```
1  4   8   12  16  20  24  28  32  36  40  44  48  52  56
// FILE NAME-INV,PACK-FIXED1,UNIT-F1,TRACKS-15,RETAIN-P
```

DATE: The DATE parameter tells the system the date of a file. It is used to ensure that the proper version of the file is referenced.

When a file is created on disk, its LABEL name and creation date are written on the disk as identification. The system date is the date used. (The system date is explained under *DATE Statement.*) More than one file on a disk can be given the same name. The creation dates of these files must, however, be different. To reference such a file, you can use its name and date (see Example 4), or its name and location on disk. If neither the date nor the location is given, the file having the latest date is the one automatically referenced.

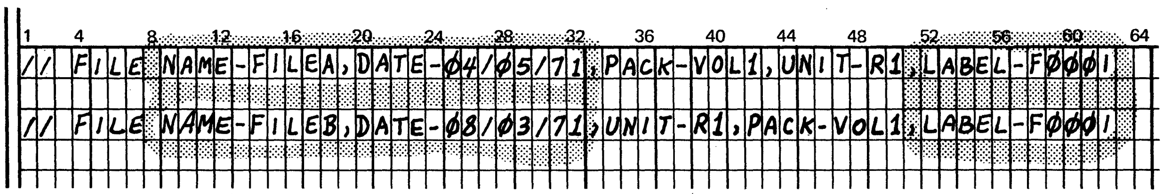
The keyword for this parameter is DATE. It must be followed by a 6-digit number representing the date (two more spaces are allowed for punctuation delimiters).

The date can be coded in one of two forms: month-day-year (mmddyy) or day-month-year (ddmmyy). You must specify the form when the system is generated. The date you specify in the DATE parameter must be in that form. The date can be coded with or without punctuation. For example, July 31, 1971, might be coded in any one of the following ways:

```
073171
310771
07/31/71
31/07/71
```

Month, day, and year must each be 2-digit numbers but lead zeros in month and day may be omitted when punctuation is used (7-31-71 or 31-7-71). A blank, comma, number, or quote cannot be used to punctuate the date.

To illustrate this parameter, assume that two versions of a file are written on the same disk. In the next example are the NAME, LABEL, and DATE parameters for two versions of a file on the same disk, one written on April 5, 1971, the other on August 3, 1971. Both files have the same label: F0001.



```
1  4   8   12  16  20  24  28  32  36  40  44  48  52  56  60  64
// FILE NAME-FILEA,DATE-Ø4/Ø5/71,PACK-VOL1,UNIT-R1,LABEL-FØØØ1

// FILE NAME-FILEB,DATE-Ø8/Ø3/71,UNIT-R1,PACK-VOL1,LABEL-FØØØ1
```

HIKEY: The HIKEY parameter must be used when you define a multivolume indexed file. The highest keyfield for each pack must be entered. For further information and an example of HIKEY see *Multivolume Files* under *Using OCL.*

SPLIT: The SPLIT parameter is used when creating and maintaining split cylinder files on a 5445 disk. For further information on SPLIT see *Split Cylinder Files.*

Examples

The following are examples of FILE statements. In each example, the file is described first, then the corresponding FILE statement is shown.

Example 1: Suppose that each week you create a disk file that contains the records for the transactions you had made that week. Assume the following facts about that file:

- The name your program uses to refer to the file is TRANS, which is also the name you want to use to identify the file on disk.
- You are placing the file on a removable disk named VOL03.
- You intend to mount the disk on drive one.
- You want to save the file for use at the end of the month.
- The file contains 225 records.
- You are letting the system choose the disk area that will contain the file.

The following example shows how the FILE statement for the preceding file is coded when using a 5444 disk.

```
1  4   8   12  16  20  24  28  32  36  40  44  48  52  56  60
// FILE NAME-TRANS,PACK-VOLØ3,UNIT-R1,RETAIN-T,RECORDS-225
```

The FILE statement when using a 5445 disk would be:

```
Name        Operation   Operand                                                          Remarks
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 ...                                              59 60 61
// FILE NAME-TRANS,PACK-VOLØ3,UNIT-D1,RETAIN-T,RECORDS-225
```

Example 2: Suppose you had created, on the same disk (VOL03), four versions of the transaction file described in the preceding example—one for each of the weeks in February, 1970. Assume the following:

- You had created the files on the following days: 2/6/70, 2/13/70, 2/20/70, and 2/27/70 (these were the system dates used for each of the files).
- You want to reference the third file (the one created 2/20/70).
- You intend to mount the disk on drive one.

The file statement you would need is:

```
1  4   8   12  16  20  24  28  32  36  40  44  48  52  56  60
// FILE NAME-TRANS,DATE-Ø2/2Ø/7Ø,PACK-VOLØ3,UNIT-R1
```

Example 3: Suppose at the end of the month you combine the files referred to in Example 2, for use in preparing your monthly bills. Further assume the following:

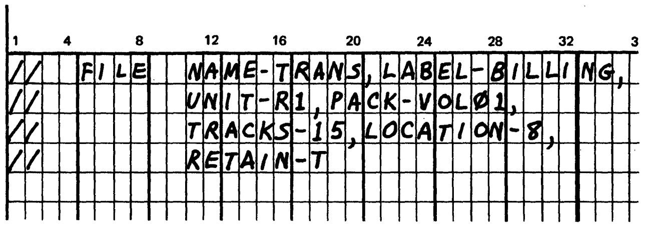
– Your program uses the name TRANS to refer to the file, but you want to use the name BILLING to identify the file on disk.

– You are expressing the amount of disk space as the number of tracks required to contain the file (assume the number is 15), and you want the file to begin on track 8.

– You are placing the file on a removable disk named VOL01.

– You intend to mount the disk on drive one.

The following example shows the FILE statement you would use for this file.



```
//  FILE   NAME-TRANS,LABEL-BILLING,
//         UNIT-R1,PACK-VOL01,
//         TRACKS-15,LOCATION-8,
//         RETAIN-T
```

Example 4: Suppose you want to create two versions of two files on disk and later to access one version of each file. Further assume the following:

– The names your program uses to refer to the files are AA and BB, which are also the names you want to use to identify the files on disk.

– File AA is being placed on a fixed disk on drive two named FIXED2.

– File BB is being placed on a removable disk named REM5.

– You intend to mount the disk on drive two.

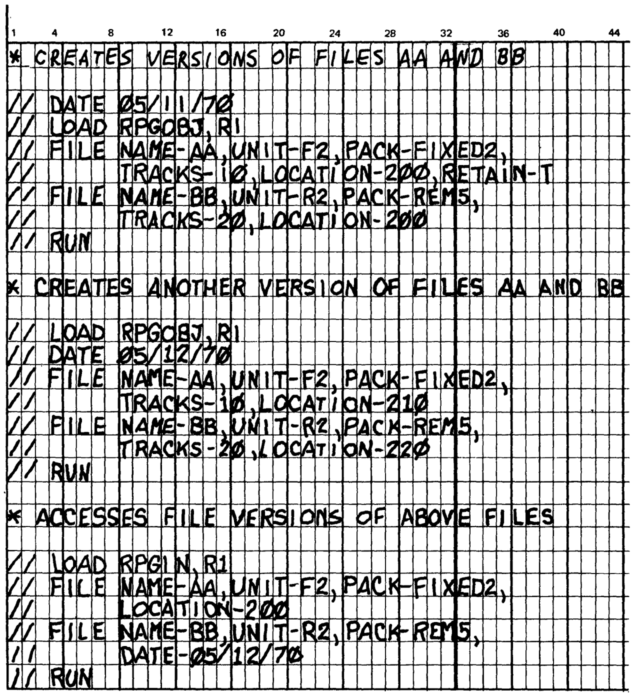
– One version of each file is created on 5/11/70 and 5/12/70.

– Disk space and location for the files are:

File	*Version*	*Tracks*	*Location*
AA	5/11/70	10	200
	5/12/70	10	210
BB	5/11/70	20	200
	5/12/70	20	220

– You want to access file AA, version 5/11/70 and file BB, version 5/12/70.

The following OCL statements are needed to create the above versions of files AA and BB and to access a version of each file.



```
1  4   8   12  16  20  24  28  32  36  40  44
* CREATES VERSIONS OF FILES AA AND BB

// DATE Ø5/11/7Ø
// LOAD RPGOBJ,R1
// FILE NAME-AA,UNIT-F2,PACK-FIXED2,
//      TRACKS-1Ø,LOCATION-2ØØ,RETAIN-T
// FILE NAME-BB,UNIT-R2,PACK-REM5,
//      TRACKS-2Ø,LOCATION-2ØØ
// RUN

* CREATES ANOTHER VERSION OF FILES AA AND BB

// LOAD RPGOBJ,R1
// DATE Ø5/12/7Ø
// FILE NAME-AA,UNIT-F2,PACK-FIXED2,
//      TRACKS-1Ø,LOCATION-21Ø
// FILE NAME-BB,UNIT-R2,PACK-REM5,
//      TRACKS-2Ø,LOCATION-22Ø
// RUN

* ACCESSES FILE VERSIONS OF ABOVE FILES

// LOAD RPGIN,R1
// FILE NAME-AA,UNIT-F2,PACK-FIXED2,
//      LOCATION-2ØØ
// FILE NAME-BB,UNIT-R2,PACK-REM5,
//      DATE-Ø5/12/7Ø
// RUN
```

File Processing Considerations

- LOCATION and space (TRACKS or RECORDS) must be specified when you are loading to an existing temporary file.
- If you are referencing a file by the DATE parameter and space is given, the space must be equal to the space given when that file was created.
- If you are accessing a file by the LOCATION parameter and space is given, the space must be equal to the space given when that file was created.
- You can create several versions of a file with a program by changing the locations of the files and using different system dates.
- You can create different versions of a file without LOCATION if the space parameters as well as the system dates are different.
- The system assumes that a new file is being created if space is given without LOCATION or DATE and the given filename was found but its space does not match.
- The DATE parameter is only allowed for accessing existing files.
- Whenever a load is performed to an existing file, the system date replaces the previous date for that file.

- If a RETAIN parameter is not specified when loading to an existing file, the existing file classification is assumed.

- When a scratch 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 to change a T or P entry to S by using a $DELET SCRATCH statement.

CALL STATEMENT

Function

CALL statements are needed only when you want to merge procedures into the job stream.

To understand the funtion of the CALL statement, you must understand the relationship between the job stream and procedures. The job stream contains the OCL statements that control the system. The system reads it either from cards or the printer-keyboard. Procedures are sets of OCL statements in a source library on disk. They have no effect on the system until they are merged into the job stream.

You can modify the procedure identified by a CALL statement, by providing other other OCL statements (procedure override statements, see *Changing Procedure Parameters)* after the CALL statement. These statements temporarily modify the procedure. The last statement of the CALL sequence must be a RUN statement. The RUN statement is required, however, whether or not you supply other OCL statements. (Procedures are further explained in *Procedures.*)

Placement

CALL statements can be used in the job stream or in a procedure. They are, in effect, replaced by the procedures they identify. The last statement of the CALL sequence must be a RUN statement.

Format

// CALL procedure-name,unit

Contents

Procedure-name: The procedure-name is the name that identifies the procedure in the source library. You supply the procedure-name in the Library Maintenance control statements when you use the program to place the procedure in the library. (See *Library Maintenance* in Part II of this manual for restrictions on procedure-name.)

Unit: The unit parameter is a code. The code indicates where the disk that contains the procedure is located on the disk unit. 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

Example

There is no CALL statement example here. The following section, *Procedures,* contains CALL statement examples.

PROCEDURES

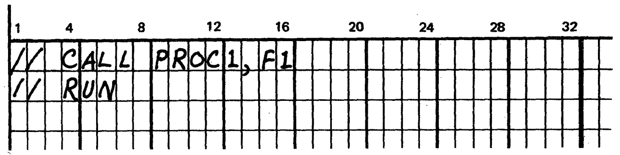
Procedures are sets of OCL statements in a source library on disk. Procedures can be put into the source library by using the Library Maintenance program. (See Part II of this manual, *Library Maintenance, Copy Function, Reader-to-Disk.)* Procedures must begin with a LOAD statement as the first OCL statement. All OCL statements except READER, CALL, LOAD, and /&, can follow the LOAD statement in a procedure. Object programs loaded from cards (LOAD*) are not allowed in procedures. The object programs are loaded from the system input device. However, LOAD* statements are allowed in procedures.

A maximum of 25 utility control statements can be included in procedures for the utility programs. The utility statements must follow the OCL statements in the procedure. (See *Library Maintenance,* Part II of this manual.) A RUN statement must be the last OCL statement in the procedure to separate the OCL statements from the utility control statements. The RUN statement in the job stream, rather than the one in the procedure, causes the system to run the program.

An example of a procedure is shown in Figure 9. This procedure will be referred to in all of the following examples. Assume that the name of the procedure is PROC1. The procedure-name is the name that identifies the procedure in the source library. Further assume that the procedure is contained on the fixed disk on drive one (F1).

Normal Procedure Call

To merge the procedure (unchanged) into the job stream, the statements in Figure 10 would be used in the job stream.



```
// CALL PROC1,F1
// RUN
```

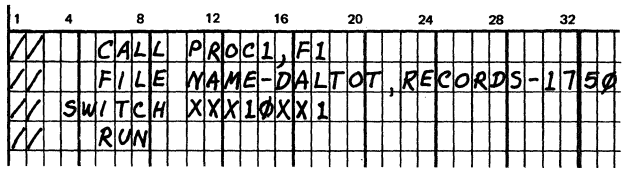
Figure 10. Normal Call for Procedure

Changing Procedure Parameters

You can change any of the parameters in any of the statements in the procedure for one job, by placing procedure override statements between the CALL and RUN statements. Procedure override statements modify the procedure for one job only. For example, assume you wanted to make the following changes to procedure PROC1 (see Figure 9):

- In the first FILE statement (NAME-DALTOT), change the RECORDS parameter from RECORDS-1500 to RECORDS-1750.
- Change the parameter in the SWITCH statement from XXX01XX0 to XXX10XX1.

Figure 11 shows the statements needed in the job stream to call and modify PROC1. Note that the NAME parameter is also supplied in the FILE statement. This is necessary to identify the FILE statement to which the change applies.

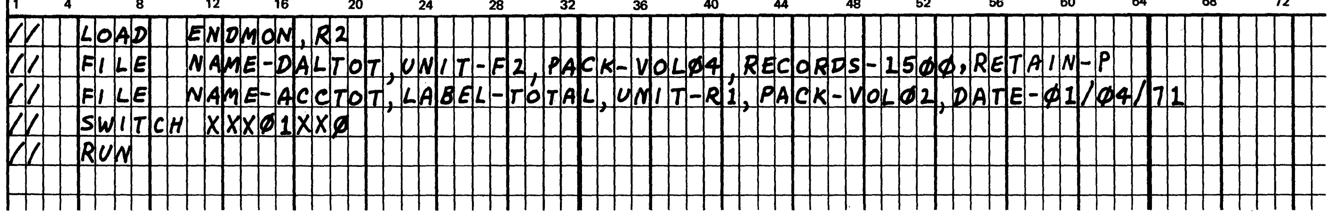


```
//   CALL PROC1,F1
//   FILE NAME-DALTOT,RECORDS-1750
//  SWITCH XXX10XX1
//   RUN
```

Figure 11. Call for Procedure: Changing Parameters

Delete a Procedure Parameter

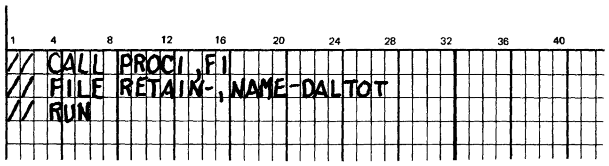
Besides changing a parameter you can delete a parameter in a procedure statement entirely if it is a keyword parameter. To delete a parameter in any of the statements you must code the keyword and the hyphen and follow them immediately with a



```
//  LOAD  ENDMON,R2
//  FILE  NAME-DALTOT,UNIT-F2,PACK-VOL04,RECORDS-1500,RETAIN-P
//  FILE  NAME-ACCTOT,LABEL-TOTAL,UNIT-R1,PACK-VOL01,DATE-01/04/71
//  SWITCH XXX01XX0
//  RUN
```

Figure 9. Procedure Example

comma. The statement in Figure 12 deletes the RETAIN parameter completely.

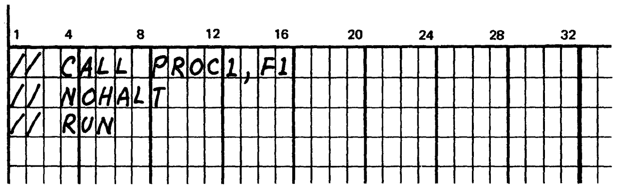


```
// CALL PROCI,F1
// FILE RETAIN-,NAME-DALTOT
// RUN
```

Figure 12. Deleting a Procedure Parameter

Adding a Statement

You can add statements to the procedure by placing the statements you are adding between the CALL and RUN statements. For example, assume that you wanted to add a NOHALT statement to the procedure. Figure 13 shows the statements needed in the job stream.

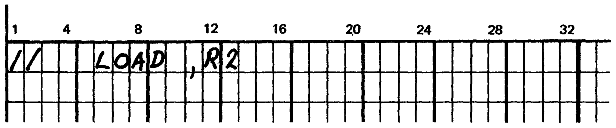


```
// CALL PROC1,F1
// NOHALT
// RUN
```

Figure 13. Call for Procedure: Adding a Statement

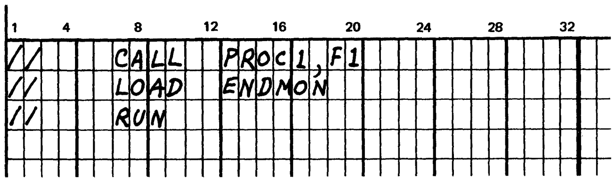
Add Missing Parameter

You can omit any of the parameters from all OCL statements in a procedure. If you do, you must supply the missing parameters between the CALL and RUN statements. For example, assume that the procedure contained the LOAD statement shown in Figure 14. The statements in Figure 15 would be needed in the job stream to run the ENDMON program. Note that the entire LOAD statement did not have to be supplied. Only the missing parameter was included.



```
//   LOAD ,R2
```

Figure 14. LOAD Statement Missing a Parameter



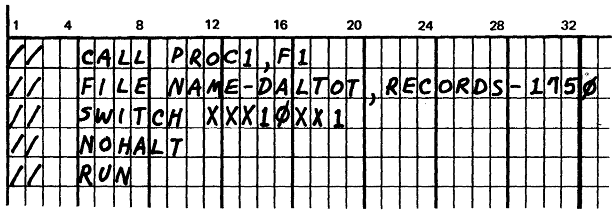
```
//   CALL   PROC1,F1
//   LOAD   ENDMON
//   RUN
```

Figure 15. Call for Procedure: Supplying a Missing Parameter

Example

Procedure override statements are printed on the logging device along with the statements in the job stream. Assume that the statements in Figure 16 are used in the job stream. The statements from the procedure would be merged with the preceding statements and printed as shown in Figure 17.

Statements preceded by XX represent the procedure statements as they appear in the source library. The CALL and RUN statements and any statements which are intended as overrides to procedure statements or additions to the procedures begin with //.



```
//  CALL PROC1,F1
//  FILE NAME-DALTOT,RECORDS-175Ø
//  SWITCH XXX1ØXX1
//  NOHALT
//  RUN
```

Figure 16. Call for Procedure Example

```
// CALL PROC1,F1
XX LOAD ENDMON,R2
XX FILE NAME-DALTOT,UNIT-F2,PACK-VOL04,RECORDS-1500,RETAIN-P
// FILE NAME-DALTOT,RECORDS-1750
XX FILE NAME-ACCTOT,LABEL-TOTAL,UNIT-R1,PACK-VOL02,DATE-01/04/71
XX SWITCH XXX01XX0
// SWITCH XXX10XX1
// NOHALT
XX RUN
// RUN
```

Figure 17. Printout of Sample Case

Nested Procedures

Some procedures are done in the same order every time a job is performed. Nesting procedures is a convenient way to link the procedures together and requires you to call only the first procedure. Each procedure will call the next procedure until the job has been completed.

By nesting procedures together several benefits can be realized.

- Programs are always run in the correct sequence.
- Operator intervention (and 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 nonrelated programs between programs of a job.

Here is an example of how nested procedures might be used. Suppose you want to back up a fixed disk pack containing files which will be used in the future. The OCL statements and utility control statements to copy one disk pack (F2) to another disk (R2) would look like this if nested procedures were *not* used:

```
//  LOAD  $MAINT,F1
//  RUN
//  ALLOCATE TO-R2,SOURCE-Ø,OBJECT-Ø
//  END

//  LOAD  $DELET,F1
//  RUN
//  REMOVE  UNIT-R2,PACK-XXXXXX,LABEL-VTOC
//  END

//  LOAD  $COPY,F1
//  RUN
//  COPYPACK  FROM-F2,TO-R2
//  END
```

By using nested procedures these control statements could be stored on disk and the job could be performed by calling only one procedure. Figure 18 shows the three procedures needed to perform the copy job described. There is only one CALL statement necessary in the job stream from the system input service.

This CALL statement links the job stream to a master procedure (CPYF22) which is used to call the procedure necessary to perform the job. CPYF22 contains three CALL statements that call the three procedures necessary to copy F2 to R2. Notice that CPYF22 contains only CALL statements. Any procedure within nested procedures can consist entirely of CALL statements and does not need a RUN statement to indicate the end of the procedure. Nested procedures allow you to have an unrestricted number of CALL statements in a procedure. Therefore CPYF22 could have more then three CALL statements if you felt it necessary to add any procedures.

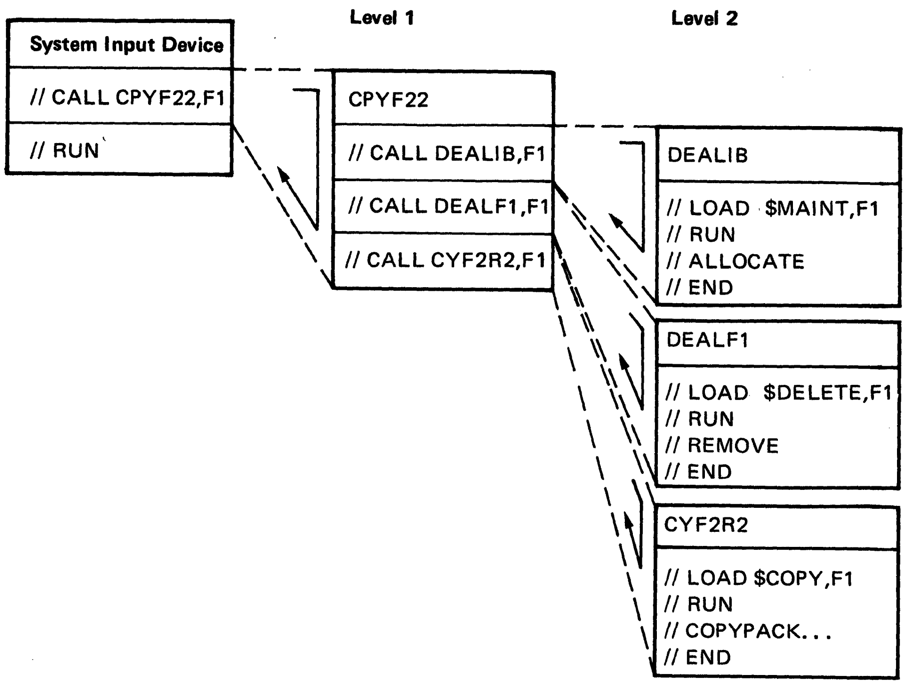


Figure 18. Nested Procedures

Figure 19 is an inventory application of nested procedures. A company issues daily reports on goods bought and sold by calling the DAY procedure. By nesting procedures together a daily report and a weekly report can be written by calling the WEEK procedure. Once a month // CALL MONTH is used to write out daily, weekly, and monthly reports. Finally, monthly, weekly, daily, and yearly reports are written once a year by calling the YEAR procedure which nests all of the other procedures together.

No more than nine levels of CALL procedures can be nested together. Levels of procedures are determined by the number of CALL statements away from the system input device a procedure is located. For instance, in Figure 19 when // CALL YEAR is given in the system input device, the

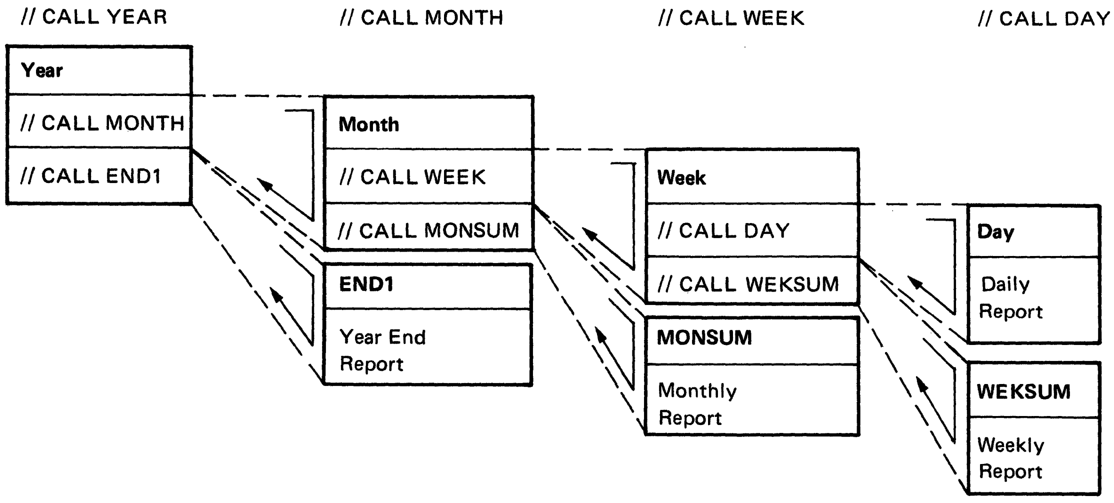


Figure 19. Inventory Example

YEAR procedure would be one level away from the system input device. MONTH and END1 procedures are two levels away from the system input device when // CALL YEAR is given.

By using nested procedures, fewer control statements are needed in the job stream from the system input device. However, certain rules must be followed to make nested procedures work:

1. No more than nine levels of procedures are permitted.

2. Each procedure may have an unrestricted number of CALL statements to the next level of procedures.

3. Only utility control statements can follow a RUN statement.

4. Procedure additions or overrides supplied between the CALL and RUN statements in the job stream are merged between the first LOAD and RUN statements encountered in the procedures (see *Example of Nesting Procedures).*

5. Any OCL statements permitted before the RUN statement in the job stream are also permitted anywhere before the RUN statement in a procedure (see *Example of Nesting Procedures).*

Example of Nesting Procedures

Suppose you want to decrease operator intervention by using the NOHALT statement. In Figure 18 the NOHALT statement could be placed between the CALL and RUN statements in the system input device. In this case it would be read as an additional OCL statement for the DEALIB procedure. However, it could be placed anywhere in the master procedure, CPYF22, or anywhere before the RUN statement in the DEALIB, DEALF1, or CYF2R2 procedures. The rule would still be followed no matter what procedure contained the additional OCL statement.

USING OCL

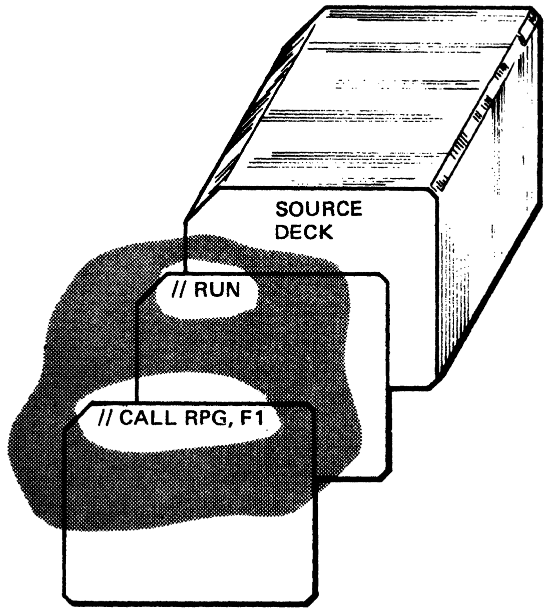
This section is designed to aid you in your use of OCL. The topics described in this manual involving the use of OCL are:

- Compiling an RPG II program
- Processing a card file
- Creating and processing a disk file
- Processing two disk files
- Processing a disk file that uses external indicators
- Creating and processing multivolume files
- Creating and processing split cylinder files
- Automatic file allocation
- Storing programs and procedures into libraries
- Checkpoint/restart
- Dual programming feature
- Statement examples

For a more complete explanation of the statements, their parameters, and coding rules refer to *Statement Descriptions* and *Coding Rules* in Part I of this manual.

COMPILING AN RPG II PROGRAM

After your RPG II program is written and recorded in cards, it must be compiled. To compile an RPG II program, two OCL statements are required, CALL and RUN.



In the preceding example the first statement, // CALL RPG,F1, tells the system to get the RPG II Compiler from the fixed disk. The second statement // RUN, tells the system to run the compiler program. The source deck always follows the RUN statement.

CREATING A DISK FILE

To create a disk file, sequential or indexed, you must tell the system the size of the file and the use of the file. To state the file size (using the FILE statement), two keywords are available: TRACKS and RECORDS. You may use one or the other, but not both.

If you use RECORDS, the system calculates the disk space required and converts it to tracks for you. If you use the TRACKS parameter, there is no need for the system to perform these calculations.

A file is classified as scratch, temporary, or permanent when it is created. You use the RETAIN parameter of the FILE statement to tell the system how to classify the use of a file. If you omit the RETAIN parameter, the file is assumed to be a temporary file.

For example, you want to create a master file of names and addresses. You would code the following:

```
1  4   8   12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
// LOAD *
// FILE NAME-SEQDISK,PACK-VOL1,UNIT-R1,RECORDS-672Ø,RETAIN-P
// RUN
```

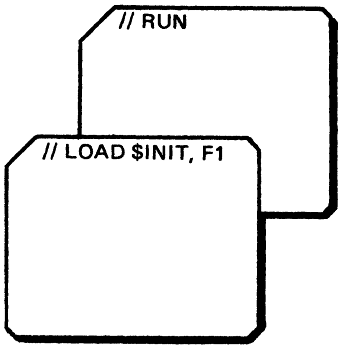
(This master file is classified as permanent.)

LOADING AND RUNNING PROGRAMS

IBM Programs

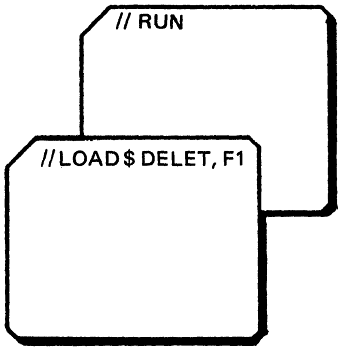
Many IBM programs require only two OCL statements, LOAD and RUN.

The following examples show the OCL cards needed to load and run two IBM programs. (The Disk Initialization and File Delete programs are discussed in Part II of this manual.)



```
// LOAD $INIT, F1
// RUN
```

The Disk Initialization program is loaded and run.



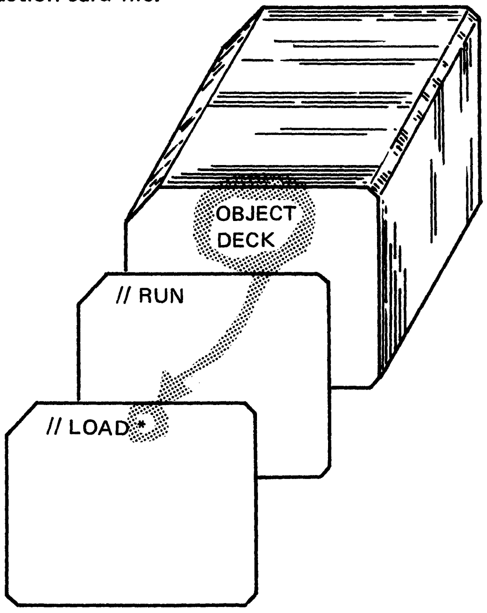
```
//LOAD $ DELET, F1
// RUN
```

The File Delete program is loaded and run.

Object Programs Using Card Files

LOAD and RUN are the only two OCL statements needed to load and run RPG II programs that use no disk files. To run a certain job, the object program must be loaded into storage. To load an object program that is on cards (object deck), an * must follow the word LOAD. (The * tells the system that an object deck follows the RUN statement.)

For example, only these two statements are required for a program that prints data from a transaction card file.



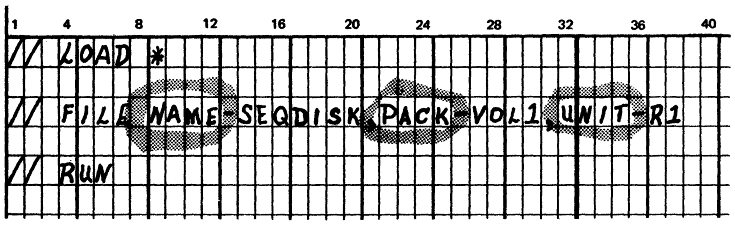
```
// LOAD *
// RUN
```

Object Programs Using One Disk File

To load and run an object program that uses a disk file, another OCL statement is required: FILE. Three items of information must follow the word FILE:

- The name of the file.
- The name of the disk pack the file is on.
- The location of the disk pack.

For example, you want to load and run an object program using a disk file named SEQDISK. The file resides on removable disk pack named VOL1. You would code the following:



```
// LOAD *
// FILE NAME-SEQDISK,PACK-VOL1,UNIT-R1
// RUN
```

Object Programs Using More Than One Disk File

One FILE statement is required for each disk file used by a program. To load and run an object program that uses two disk files, two FILE statements are required.

In the following example, two disk files are used: an input file (INDISK) and an output file (OUTDISK).

```
1  4   8   12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
// LOAD *
// FILE NAME-INDISK,PACK-VOL1,UNIT-R1
// FILE NAME-OUTDISK,PACK-VOL1,UNIT-R1,RECORDS-672Ø,RETAIN-P
// RUN
```

The first FILE statement contains information needed to access the data in that file. The second FILE statement contains information needed to create an output file.

Object Programs Using One Disk File and External Indicators

The SWITCH statement is used to set external indicators (U1-U8 on RPG II specifications sheets) on or off. External indicators are used to regulate when certain functions are performed.

In the following example,you are running a program using one disk file (INVMSTR), an inventory master file.

```
1  4   8   12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
// LOAD *
// FILE NAME-INVMSTR,PACK-VOL1,UNIT-R1
// SWITCH 1ØØØØØØØ
// RUN
```

In order for the program to perform certain functions, such as updating and output, the first external indicator (U1) must be turned on. In the SWITCH statement the eight characters correspond to the eight external indicators. In this program only one external indicator (U1) is used.

MULTIVOLUME FILES

Coding the FILE statement to process multivolume files differs from single volume files in that you must define and code additional parameters for these keywords: PACK, UNIT, TRACKS, RECORDS, and LOCATION.

These additional parameters are necessary for two reasons:

1. When processing files contained on more than a single volume, the system requires information about each volume in order to perform all the protection and checking functions necessary.

2. Additional information is needed to determine and check the sequence in which the volumes are processed and when they are to be mounted on the disk drives.

Because a multivolume file involves more than one disk, some FILE statement parameters 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 in the parameters.

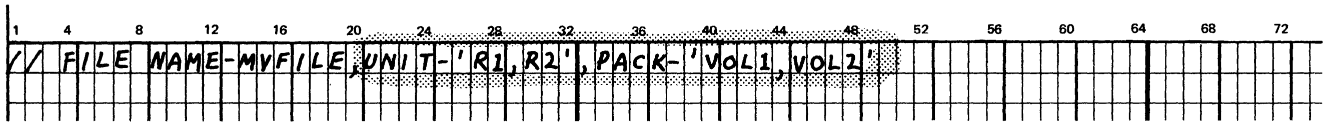
The rules for coding a list of data or codes after a keyword are as follows:

1. The list must be enclosed by apostrophes.

2. The items in the list must be separated by commas. No blanks are allowed within or between items.

Figure 20 shows an example of lists in parameters. The file is online.

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 parameter discussions following. The functions of the parameters are explained under FILE statement. (Parameters not mentioned here are used as explained under *FILE Statement.)*



```
// FILE NAME-MYFILE,UNIT-'R1,R2',PACK-'VOL1,VOL2'
```

Figure 20. Lists of FILE Statements

FILE STATEMENT PARAMETER CONSIDERATIONS FOR MULTIVOLUME FILES

PACK

The names of the disks that contain or will contain the multivolume file must follow the keyword PACK. (PACK names must be unique for proper functioning.)

When a multivolume 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 multivolume 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 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:

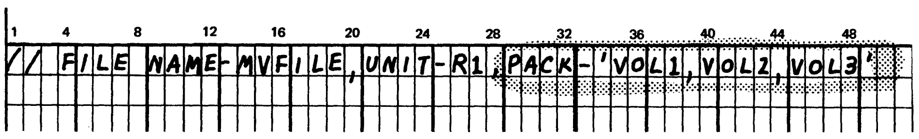
1. Mount the proper disk and restart the system.

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 multivolume. If the file is multivolume created and the sequence is ascending but not consecutive, a diagnostic halt is given which allows the proceed option.

The following is an example of the PACK parameter for an offline multivolume file that is contained on three disks, named VOL1, VOL2, and VOL3.



```
// FILE NAME-MVFILE,UNIT-R1,PACK-'VOL1,VOL2,VOL3'
```

UNIT

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

Codes	*Meaning*
R1	Removable disk on 5444 drive one
F1	Fixed disk on 5444 drive one
R2	Removable disk on 5444 drive two
F2	Fixed disk on 5444 drive two
D1	Removable disk on 5445 drive one
D2	Removable disk on 5445 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 disks must then all be removable disks. If you do, 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.

For the 5445 the UNIT parameter can have a maximum of two unit codes. When two unit codes are given, the volumes must be mounted alternately in the order indicated by the unit codes. If all the volumes are to be mounted on the same drive, you specify only one unit code.

If any fixed unit, F1 or F2, is specified, the file must be online multivolume.

Assume that your program processes an offline file consecutively. Further assume the following:

- The disks containing the file are named VOL1, VOL2, and VOL3, respectively.
- You intend to mount VOL1 and VOL3 on 5444 drive one, and VOL2 on 5444 drive two.

In the following examples, line A shows the PACK and UNIT parameters for the file. If all three disks were used on 5444 drive one, the UNIT parameter in line B would have been used.

Ⓐ
```
// FILE NAME-MVFILE,PACK-'VOL1,VOL2,VOL3',UNIT-'R1,R2'
```

Ⓑ
```
// FILE NAME-MVFILE,PACK-'VOL1,VOL2,VOL3',UNIT-R1
```

TRACKS or RECORDS

A 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 multivolume file. TRACKS or RECORDS must be specified. Any multivolume file load requires a TRACKS or RECORDS parameter whether the file previously existed or not. The order of these numbers must correspond to the order of the names in the PACK parameter. For example, assume the following:

- Your program is creating a sequential (offline) file on three disks: VOL1, VOL2, and VOL3.
- The first 50 records are to be placed on VOL1, the next 500 on VOL2, and the last 200 on VOL3.

The PACK and RECORDS parameters for the file are:

```
// FILE NAME-MVFILE,UNIT-R1,
//     PACK-'VOL1,VOL2,VOL3',RECORDS-'5Ø,5ØØ,2ØØ'
```

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. For example, assume the following:

- The disks containing the file are: VOL1, VOL2, and VOL3.
- The tracks on which the file is to begin on each disk are: track 198 in VOL1, track10 in VOL2, and track 8 in VOL3.

The PACK and LOCATION parameters for the file are shown in the following example. 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 multivolume file exists, LOCATION must be given for all disks and must be identical to the LOCATION parameters specified when the file was created.

```
1  4   8   12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
// FILE NAME-MVFILE,RECORDS-'50,550,150',
//      UNIT-'F1,R1,R2',
//       PACK-'VOL1,VOL2,VOL3',
//      LOCATION-'198,1Φ,8'
```

RETAIN

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

HIKEY

The HIKEY parameter is used only for multivolume indexed files. HIKEY limits the highest keyfield that can be put on each pack of a multivolume file. The following example contains an example of a HIKEY parameter list using the file used in example A under *Unit.* In this case the three volumes contain lists of names. The highest keyfield allowed on the first volume is JONES. This means that all the records beginning with A and including JONES will be processed on this volume. Since HIKEY parameters must be in ascending order, the next volume should contain all of the records with names following JONES and including NICHOL. The last volume will contain all the records with names that come after NICHOL.

```
1  4   8   12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
// FILE NAME-MVFILE,UNIT-'R1,R2',PACK-'VOL1,VOL2,VOL3',
// HIKEY-'JONES,NICHOL,ZZZZZZ'
```

OCL considerations for the HIKEY parameter are:

1. All characters except commas are valid.

2. The list of HIKEY parameters must begin and end with an apostrophe even if only one parameter is specified. A single apostrophe in a key field must be written as a double apostrophe in the HIKEY parameter.

3. For each PACK parameter specified, there must be a corresponding HIKEY keyfield parameter for that pack.

4. The HIKEY fields must be equal in length and must be specified in ascending order.

5. The maximum length of a HIKEY field is 29 characters.

6. The HIKEY fields must be the same length as the keys on file.

Packed HIKEY: The packed HIKEY parameter has all the OCL considerations for HIKEY including the following restrictions:

1. The first character following the HIKEY keyword and dash (HIKEY-) must be a P to indicate packed HIKEY.

2. All characters in the packed HIKEY must be zoned numerics (0-9).

3. The number of digits in each packed key must be the same.

4. The number of zoned numeric characters per packed HIKEY must not exceed 15, since the maximum packed key field length is 8.

The following example shows a packed HIKEY parameter. In the example the key field length of MVFILE is 2. The HIKEYs are X'085F', X'092F', and X'l08F' for VOL1, VOL2, and VOL3 respectively. The first two packed keys required a leading zero to make the lengths consistent.

```
1  4   8   12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
// FILE NAME-MVFILE,UNIT-'R1,R2',PACK-'VOL1,VOL2,VOL3',
// HIKEY-P'Ø85,Ø92,1Ø8'
```

SPLIT CYLINDER FILES

To use split cylinder file support, two parameters (SPLIT and LOCATION) are specified on the FILE statement. The SPLIT parameter specifies the size of each split cylinder file. It can also be used to specify the size of the group of split cylinder files you want on disk. The LOCATION parameter determines where on the 5445 disk each split cylinder file can be found. For further discussion of split cylinder file concepts, see *IBM System/3 Disk Concepts and Planning Guide,* GC21-7571.

Restrictions for Using Split Cylinder Files

1. Split cylinder files can only be direct or consecutive files and cannot be multivolume files.
2. Split cylinder files can only be used with the 5445 disk and not the 5444 disk.
3. TRACKS or RECORDS parameters must not be specified.
4. Labels must be unique. Therefore, the DATE parameter is used only to further qualify the split cylinder file. The file date is always the current system date for the job.
5. A data block cannot overlap cylinders. This means that a data block cannot be longer than the space available on one cylinder of a split cylinder file.

Creating the First Split Cylinder File in a Group

The SPLIT parameter is required when creating the first split cylinder file in a group of split cylinder files. The LOCATION parameter is optional.

The SPLIT parameter entries are:

SPLIT-tracks per cylinder/number of cylinders

The tracks per cylinder entry specifies the amount of space needed on each cylinder for the first split cylinder file. The cylinders entry shows the number of cylinders needed for the whole group of split cylinder files to be specified.

The LOCATION parameter is optional since the system will find a starting location for the split file group. However, if you want to specify a particular cylinder, you may.

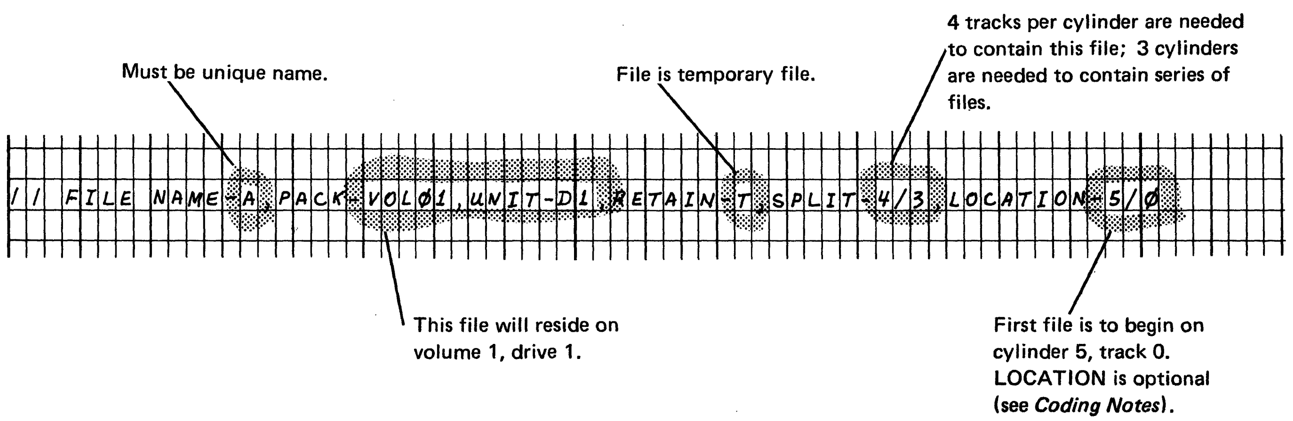
The LOCATION entries are:

LOCATION-cylinder number/track number

The split cylinder file group must always start at track 0. Since 0 will always be the entry for track, you can omit it from the LOCATION parameter and use:

LOCATION-cylinder number

File Statement Example: First Split Cylinder File in a Group



```
// FILE NAME-A,PACK-VOL01,UNIT-D1,RETAIN-T,SPLIT-4/3,LOCATION-5/0
```

Must be unique name.
File is temporary file.
4 tracks per cylinder are needed to contain this file; 3 cylinders are needed to contain series of files.
This file will reside on volume 1, drive 1.
First file is to begin on cylinder 5, track 0. LOCATION is optional (see *Coding Notes*).

Coding Notes:

1. On the SPLIT parameter, tracks per cylinder, must be 1-19 and the number of cylinders specified must be 1-199.

2. On the LOCATION parameter, the cylinder number must be 1-199 and the track number, if specified, must be 0.

3. LOCATION-5 could be the location entry in this example since track 0, the required track entry, need not be specified. The LOCATION parameter itself is optional.

Creating Other Split Cylinder Files

To create the rest of the split cylinder files in a group both the SPLIT and LOCATION parameters are required. The SPLIT parameter must be in the format:

SPLIT-tracks per cylinder

This entry, tracks per cylinder, indicates the number of tracks needed on each cylinder for the file specified.

The LOCATION parameter must be the filename of either the first split cylinder file in the group or any other split cylinder file in the group that was created in a previous job.

LOCATION-filename

File Statement Example: Other Split Cylinder Files

Coding Notes:

1. On the SPLIT parameter, tracks per cylinder must be 1-19.

2. On the LOCATION parameter, the filename must be the name of a temporary or permanent split cylinder file in the same group.

Accessing Existing Split Cylinder Files

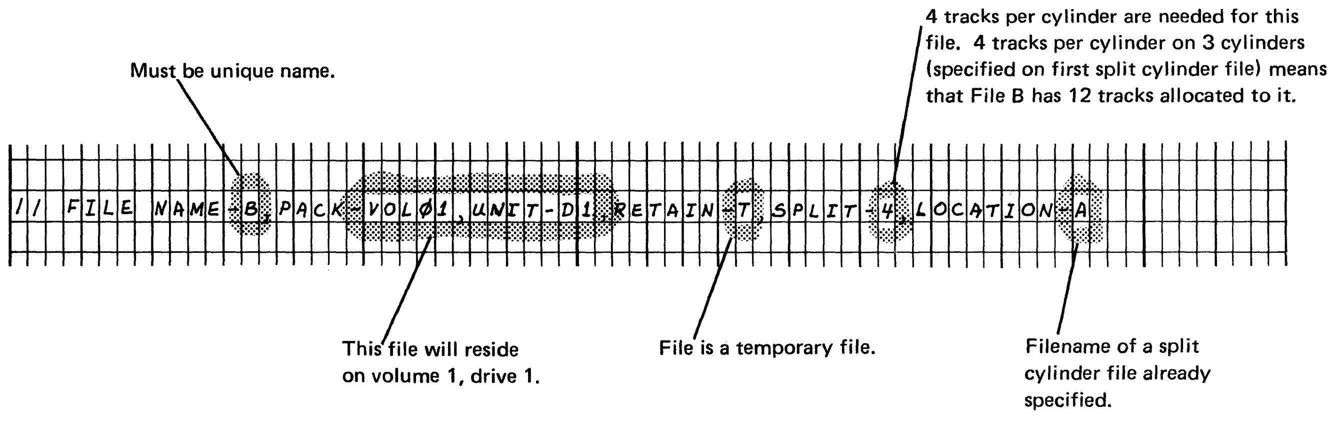
To access existing split cylinder files, the SPLIT and LOCATION parameters are not required. Their use would only be needed to further qualify the file being accessed.

Loading to Existing Split Cylinder Files

To load to existing split cylinder files, the SPLIT parameter is required and the LOCATION may be required or optional. The SPLIT parameter specified for loading must agree with the SPLIT parameter of the existing split cylinder file. If the format of the SPLIT parameter is *tracks per cylinder/cylinders,* the LOCATION parameter is required and must match the cylinder number/track number of the existing split cylinder file. If the format of the SPLIT parameter is *tracks per cylinder,* the LOCATION parameter is optional.

Scratch Split Cylinder Files

Split cylinder files may be created as temporary or permanent files and in subsequent jobs made scratch files. However, the scratch files remain on the 5445 disk only until the area is needed for the allocation of a new file. Then, the scratch split cylinder file is deleted. If you have scratched split cylinder files and you want to make sure they are not deleted, you may reactivate them to temporary files by using a RETAIN-A on the FILE statement.



```
// FILE NAME-B,PACK-VOLØ1,UNIT-D1,RETAIN-T,SPLIT-4,LOCATION-A
```

Must be unique name.
4 tracks per cylinder are needed for this file. 4 tracks per cylinder on 3 cylinders (specified on first split cylinder file) means that File B has 12 tracks allocated to it.
This file will reside on volume 1, drive 1.
File is a temporary file.
Filename of a split cylinder file already specified.

AUTOMATIC FILE ALLOCATION

You can allocate disk space for a file by determining the size of the file and the location of an available number of tracks that can contain that file. (If you have planned the location of your files, you know where files are located and the tracks that are available for further allocation. The Disk File Layout Chart, GX21-9108, is available to document your file locations.) After you have determined where to place your file, you can code the LOCATION parameter of the FILE statement to tell disk system management on which track the file is to begin. Figure 21, part A, is a sample FILE statement containing a LOCATION parameter to tell disk system management that FILEA is to be located on disk VOL1 beginning on track 10.

If, as in Figure 21, part B, no LOCATION parameter is coded, FILEA is located on the disk pack automatically for you. The process used by disk system management to allocate file space for you is known as *automatic file allocation.*

COMPILING AND STORING A SOURCE PROGRAM IN AN OBJECT LIBRARY

The COMPILE OCL statement tells disk system management to:

1. Compile a source program from a source library and store the object program in an object library, or

2. Compile a source program from cards and store the object in an object library.

The format of the COMPILE statement looks like this:

$$\text{// COMPILE SOURCE–name,UNIT–}\left\{\begin{matrix}\text{R1}\\\text{F1}\\\text{R2}\\\text{F2}\end{matrix}\right\}\text{, OBJECT–}\left\{\begin{matrix}\text{R1}\\\text{F1}\\\text{R2}\\\text{F2}\end{matrix}\right\}$$

The SOURCE keyword parameter is used if the source program is located in a source library. You must supply the same name given to the source program when it was stored in the library by the Library Maintenance program. The UNIT parameter must be used with the SOURCE parameter to identify the disk location of the source program to be compiled.

If the SOURCE keyword parameter is not used, the source program is assumed to be on cards following the RUN statement in the job stream.

The OBJECT keyword parameter tells the system where the disk which will contain the object program is located. If the source program is on cards, the OBJECT keyword parameter is the only parameter which can be specified. If the OBJECT keyword parameter is omitted in either case, the object program is placed on the same disk pack as the compiler. The name assigned to the object program in the object library is the name you assigned in the Program Identification (columns *75-80) on the RPG II Control Card Sheet. If you did not assign a name in these columns, RPGOBJ is assumed.

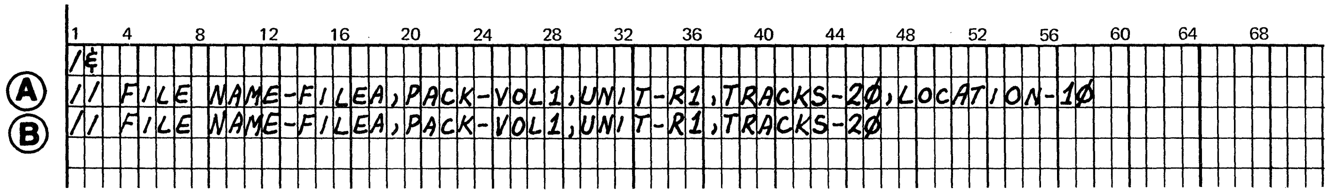

```
    1  4   8   12  16  20  24  28  32  36  40  44  48  52  56  60  64  68
    /¢
(A) // FILE NAME-FILEA,PACK-VOL1,UNIT-R1,TRACKS-2Ø,LOCATION-1Ø
(B) // FILE NAME-FILEA,PACK-VOL1,UNIT-R1,TRACKS-2Ø
```

Figure 21. File Statement and Use of the LOCATION Parameter

Sample Statements

```
1  4   8   12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
/&
// CALL RPG,F1
// COMPILE SOURCE-SALES,UNIT-F1,OBJECT-R1
// RUN
```

This sample job stream tells the system that the source program named SALES is located on a fixed disk on drive one (F1). The OBJECT-R1 keyword parameter tells the system to place the object program on a removable disk on drive one (R1).

```
/&
// LOAD $PRG,F1
// COMPILE OBJECT-R1
// FILE NAME-$WORK,UNIT-F1,PACK-F1F1F1,RETAIN-S,TRACKS-2Ø
// FILE NAME-$SOURCE,UNIT-F1,PACK-F1F1F1,RETAIN-S,TRACKS-2Ø
// RUN

 (SOURCE DECK)
```

This sample job stream compiles a source program on cards and stores it in an object library on R1. If the OBJECT parameter was not coded, the program would be compiled and placed into the same object library as the compiler (F1).

LOADING PROGRAMS IN A DPF ENVIRONMENT

A program can be loaded into either program level first. You tell the supervisor which system input device contains the job streams for the programs by selecting the device on the Dual Program Control Switch. (Refer to the *IBM System/3 Disk System Operator's Guide,* GC21-7508 for further operating procedures.) When preparing your job streams, you should be aware of the following OCL considerations:

OCL CONSIDERATIONS FOR LOADING PROGRAMS IN A DPF ENVIRONMENT

DATE statement

The DATE statement you use as an IPL statement to set the system date must be supplied with the first program loaded in one program level. The DATE statement must precede the set of statements for the first program. In the device associated with the other program level, a DATE statement must not precede the sets of statements for the programs being run in that level.

A DATE statement that temporarily changes the system date can be used within the set of OCL statements for programs in either program level. This DATE statement applies only to the program for which it is used.

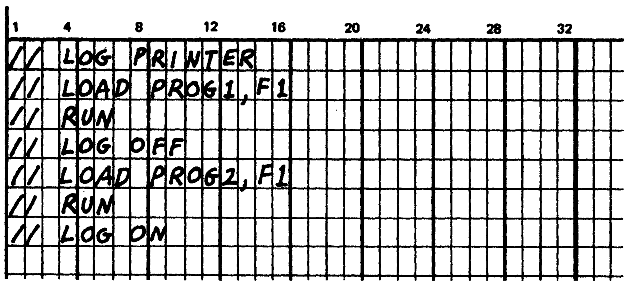
LOG statement

LOG statements can be placed anywhere among the statements in either job stream. There are, however, certain restrictions on their use.

– Only LOG statements for program level 1 can tell the system to use a different logging device. Only ON or OFF can be specified in program level 2. The device used for level 1 is also used for level 2.

- LOG must be on for both program levels before logging can occur. If a LOG statement for either program level stops the logging function, logging is stopped for both levels. The program level that turned the logging device off must turn it back on before logging can resume. If both levels specify OFF, then both program levels must turn the logging device back on before logging can resume.
- When the printer is the logging device, OCL statements and message codes are not printed if the program in either level uses the printer as an output device.

The following example shows sample LOG statements in a job stream:

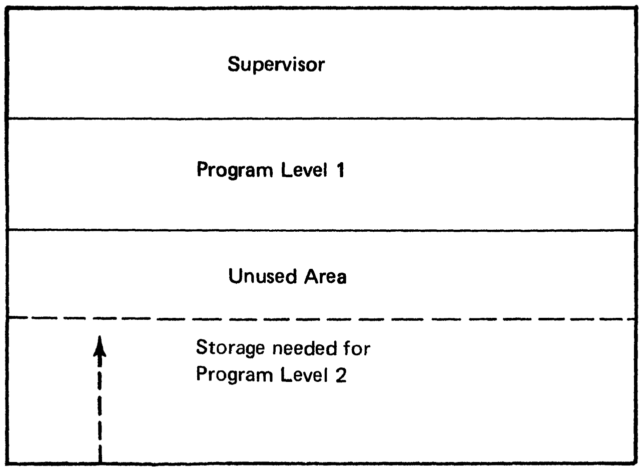


```
// LOG PRINTER
// LOAD PROG1,F1
// RUN
// LOG OFF
// LOAD PROG2,F1
// RUN
// LOG ON
```

Note: The first LOG statement indicates that the printer is used as the logging device while program PROG1 is being run. OCL statements and error messages are not printed for program PROG2 because of the second LOG statement. The third LOG statement causes the logging device to be used again.

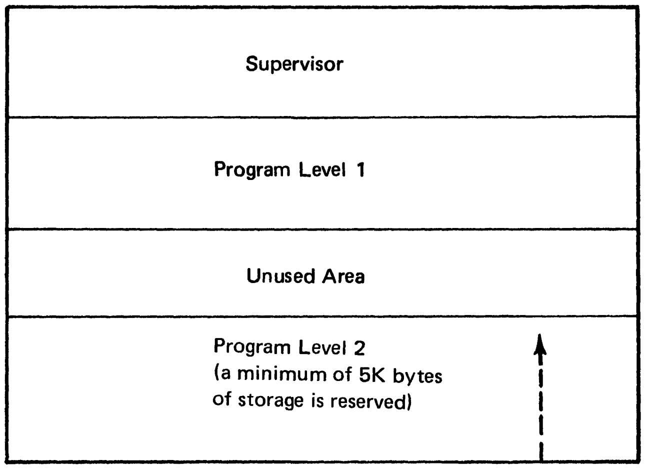
NOHALT statement	The NOHALT statement is ignored for program level 2. The program in this level always stops after each job.
HALT statement	The HALT statement is ignored by program level 2.
IMAGE statement	The IMAGE statement is invalid and the job cannot be run, if the other level has the printer allocated to it.
FORMS statement	The FORMS statement is invalid and the job cannot be run, if the other level has the printer allocated to it.
LOAD statement	The LOAD* statement cannot be used in program level 2.
PARTITION statement	The PARTITION statement is used only in DPF.

The PARTITION statement is used to guarantee a minimum size to level 2 for a subsequent program in that level.



Without a PARTITION Statement

If level 1 is not using the storage and a program is loaded into level 2, it is assigned the number of bytes requested by program attributes. When the program in level 2 comes to end of job, the storage for level 2 is no longer reserved and level 1 can use it.


Supervisor
Program Level 1
Unused Area
Program Level 2 (a minimum of 5K bytes of storage is reserved)

With a PARTITION Statement

If a PARTITION statement is used, the assigned storage can only be used by the program in level 2. It is reserved. Even when the program in level 2 comes to end of job that storage is reserved for future programs in level 2.

If you do not use a PARTITION statement and, therefore, do not indicate the minimum size of program level 2, the system automatically assigns, during execution, the storage needed to level 2. You cannot submit a PARTITION statement in program level 2 or when program level 2 is processing. In a procedure the PARTITION statement must follow the LOAD statement and precede the RUN statement.

The format of the PARTITION statement is:

// PARTITION size

You must state the minimum number of bytes of storage you want to save for program level 2. The number must be equal to or greater than 5120. The amount of storage you specify is rounded to the next highest .25K by the supervisor, if it is not a multiple of .25K.

DPF Considerations

All programs require 5K bytes of storage for initiation and termination even though a program may occupy less than 5K. System programs use this storage for performing system functions just prior to loading the user's object program (initiation) and again immediately following the end of object program execution (termination).

This 5K requirement also affects DPF. For independent initiation and termination of a program in DPF, at least 5K bytes of storage must be available for each program level, regardless of the size of the program to be executed. If a program needs less than 5K while another program requires the remaining storage which is 5K or larger, the smaller program must be initiated first so that the storage required by the system for initiation will be available. The system can then use all the storage not required by the smaller program for the larger program. However, the smaller program must wait for termination of the larger program, so that 5K is available for the smaller program's termination.

In a 12K DPF system only limited independent initiation and termination is allowed. With a 4K minimum size requirement for the supervisor only 8K is available for user programs. Independent program initiation and termination for each program is possible if each program being run occupies 3K or less of storage. The remaining 2K of storage is used alternately by either program to satisfy the 5K system requirement. If one program needs more than 3K, the smaller program must be initiated first and can have a maximum executing size of 3K. The larger program is then initiated and can occupy the remaining storage. The larger program level must be terminated before the smaller program level.

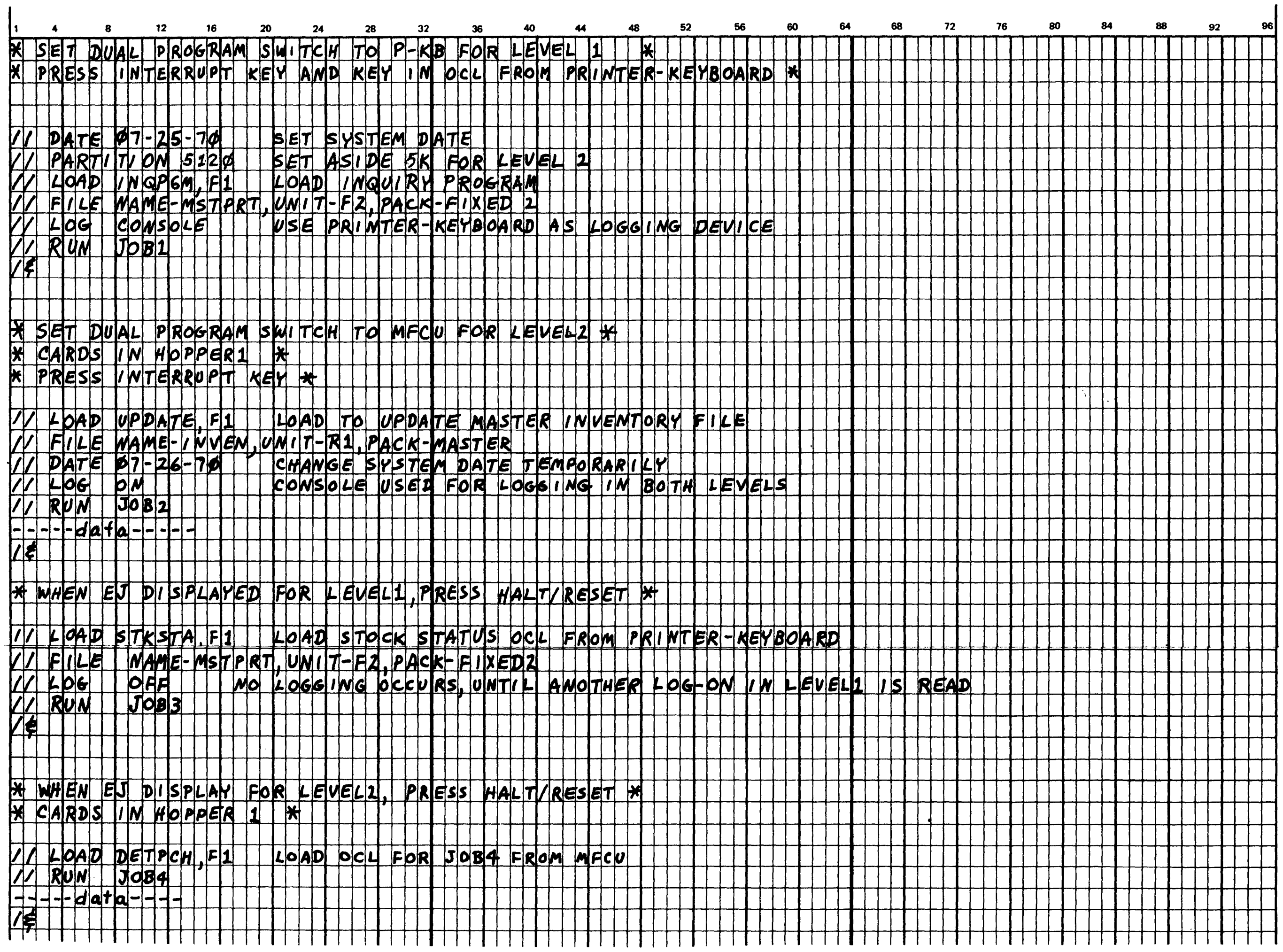
Sample Job Streams

Suppose you had four jobs to be run requiring the I/O shown in Figure 22. Jobs 1 and 2 and Jobs 3 and 4 can be run together, because they do not require the same I/O devices. If Job 2 finishes before Job 1, you could run Job 4 because Jobs 1 and 4 do not require the same devices. If, on the other hand, Job 1 finishes first, Job 3 could not be run with Job 2, because both jobs require the printer for output.

Figure 23 shows the job streams required to load the four jobs. Assume the system has the minimum system configuration plus the 5471 Printer-keyboard and dual drives. The Dual Program Switch indicates from what device OCL statements are read. MFCU is always hopper 1, and at system generation time P-KB was assigned to the 5471 Printer-keyboard.

Program Level 1	JOB1 An inquiry program that: • Reads printer-keyboard. • Reads disk. • Writes printer-keyboard.	JOB3 A stock status report that: • Reads disk. • Prints.
Program Level 2	JOB2 An inventory updating program that: • Reads cards. • Reads disk. • Updates disk. • Prints.	JOB4 A detail punching job that: • Reads cards. • Punches cards.

Figure 22. Job Scheduling for DPF



```
* SET DUAL PROGRAM SWITCH TO P-KB FOR LEVEL 1   *
* PRESS INTERRUPT KEY AND KEY IN OCL FROM PRINTER-KEYBOARD *


// DATE Ø7-25-7Ø    SET SYSTEM DATE
// PARTITION 512Ø   SET ASIDE 5K FOR LEVEL 2
// LOAD INQP6M,F1   LOAD INQUIRY PROGRAM
// FILE NAME-MSTPRT,UNIT-F2,PACK-FIXED 2
// LOG  CONSOLE     USE PRINTER-KEYBOARD AS LOGGING DEVICE
// RUN  JOB1
/&


* SET DUAL PROGRAM SWITCH TO MFCU FOR LEVEL2 *
* CARDS IN HOPPER1  *
* PRESS INTERRUPT KEY *

// LOAD UPDATE,F1   LOAD TO UPDATE MASTER INVENTORY FILE
// FILE NAME-INVEN,UNIT-R1,PACK-MASTER
// DATE Ø7-26-7Ø    CHANGE SYSTEM DATE TEMPORARILY
// LOG  ON          CONSOLE USED FOR LOGGING IN BOTH LEVELS
// RUN  JOB2
-----data-----
/&

* WHEN EJ DISPLAYED FOR LEVEL1,PRESS HALT/RESET *

// LOAD STKSTA.F1   LOAD STOCK STATUS OCL FROM PRINTER-KEYBOARD
// FILE  NAME-MSTPRT,UNIT-F2,PACK-FIXED2
// LOG   OFF      NO LOGGING OCCURS,UNTIL ANOTHER LOG-ON IN LEVEL1 IS READ
// RUN   JOB3
/&


* WHEN EJ DISPLAY FOR LEVEL2, PRESS HALT/RESET *
* CARDS IN HOPPER 1  *

// LOAD DETPCH,F1   LOAD OCL FOR JOB4 FROM MFCU
// RUN   JOB4
----data----
/&
```

Figure 23. Sample Job Stream

RESTARTING A CHECKPOINTED PROGRAM

Checkpoint is a means of recording the status of a problem program at desired intervals. Restart is a means of resuming the execution of the program from the last checkpoint rather than from the beginning, if processing is terminated for any reason (with the exception of a controlled cancel) before the normal end of job. For example, a power failure may occur and cause an interruption.

Programming Considerations

- Checkpoint/Restart enables the user to restart a checkpointed program from the last checkpoint taken provided no intervening program executions have taken place.
- Sufficient disk space is allocated by Library Maintenance on a checkpoint system pack (5444) at System Generation or Library Maintenance time to allow one active checkpoint. On a system with Checkpoint and Inquiry, the disk space will be used by both functions. The checkpoint program cannot be an inquiry evoking program since the disk space is used by both facilities.
- Checkpoint requests are accepted only in program level 1. Checkpointed programs must be restarted in program level 1. If program level 2 is used to execute a checkpointed program, the checkpoint requests are ignored.

Restart Procedure

To restart the interrupted job at the last checkpoint submit the following OCL statements:

```
// LOAD $$RSTR, unit
// RUN
```

The unit in this example is a pack with module $$RSTR. If an IPL occurs it must be from the pack other than the pack that contains the active checkpoint is allowed, but programs executed under control of the new IPL system cannot access disk volumes used in the active checkpointed program or modify the object library where the checkpointed program resides.

Other OCL statements that may be required are the PARTITION and LOG statements.

OCL CONSIDERATIONS FOR USING CHECKPOINT/RESTART

PARTITION statement

A PARTITION statement may be required at restart to guarantee the required minimum level 2 size. See *Loading Programs in a DPF Environment* for further information on the PARTITION statement.

- A halt will occur if restart is attempted without sufficient space in program level 1. An immediate cancel is taken.
- Checkpoints can only be taken in program level 1. To restart a checkpointed program, program level 1 must be used. If level 2 is used to execute a checkpointed program, the checkpoint requests are ignored.
- Restart requires 5K of storage; therefore level 2 must be such that level 1 has 5K.

LOG statement

A LOG statement may be required at restart to reestablish the logging device. See *LOG Statement* under *Statement Descriptions* and *Loading Programs in a DPF Environment* for further information on the LOG statement.

STATEMENT EXAMPLES

This section shows an example that illustrates some of the uses of the OCL statements. The example consists of a series of jobs. The jobs involve three files: customer, inventory, and transaction. The customer file contains such information as customer names and addresses, total amounts of charges over a period, and total amounts of payments over the same period. The inventory file contains such information as item numbers and descriptions, prices of the items, and the numbers of items in stock. The transaction file contains such information as orders for items, refund orders for items returned, and customer payments. The transaction file is used to update the inventory and customer files.

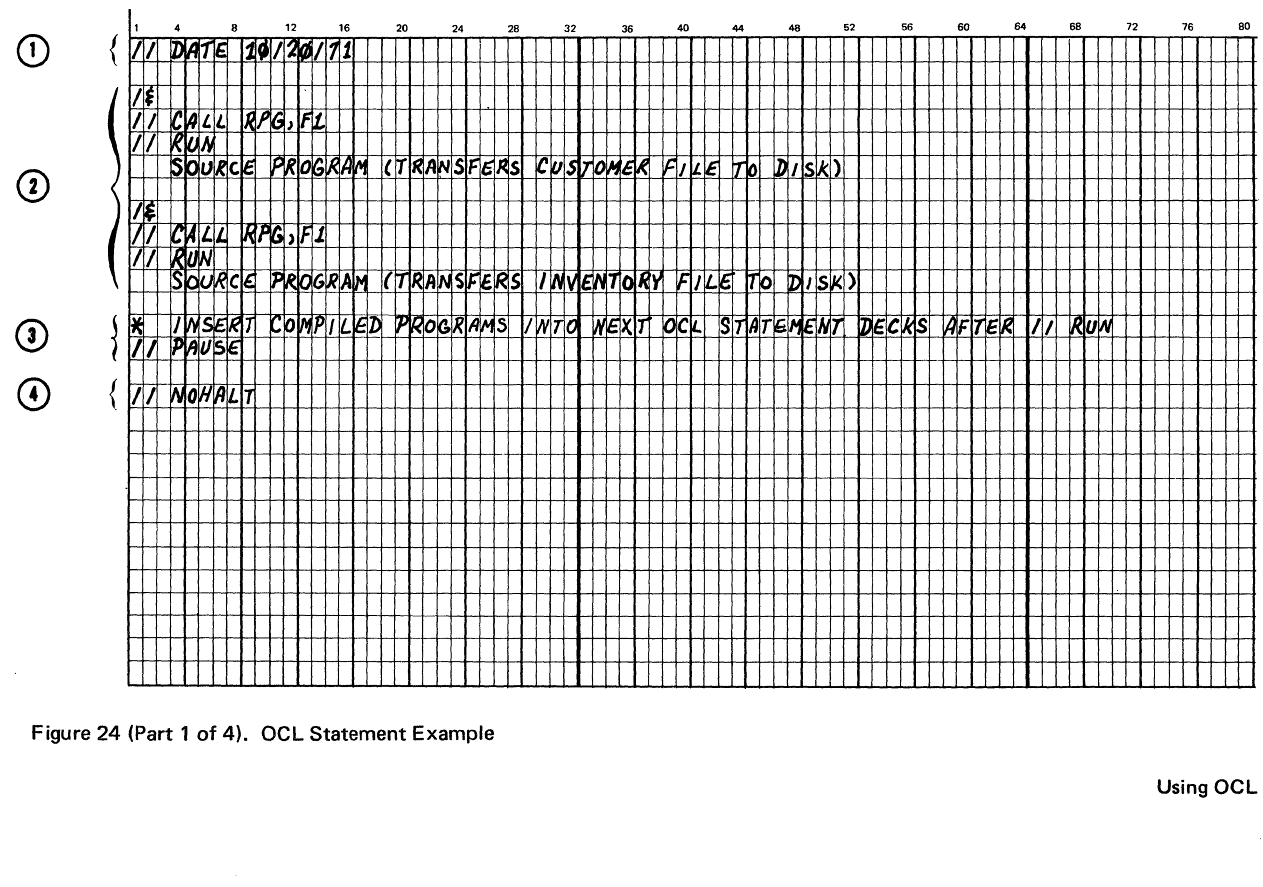
Example

The OCL statements for the jobs are shown in Figure 24. Sets of statements in the figure are numbered. The explanations corresponding to those numbers are given in the following section.

Explanation

1. The DATE statement supplies the system date, 10/20/71. It must be read by the system before the first LOAD or CALL statement after initial program load.
2. Two programs are being compiled: one that transfers the customer file from cards to disk; and one that transfers the inventory file from cards to disk. The OCL statements for the RPG II Compiler are in a procedure called RPG. A CALL statement, therefore, is used to instruct the system to read the procedure each time the compiler is to be run. The procedure is located on the fixed disk on drive one.

   The RPG II source programs following each set of CALL and RUN statements are input to the compiler. Like all input, each source program must be followed by a /* card. However, to be safe, /& statements were used before each LOAD and CALL statement in case the /* cards had not been placed after the source programs.
3. In the next two jobs, the object programs just compiled will be run. The comment and PAUSE statements are to remind the operator to place the object- program cards after the corresponding sets of OCL statements.
4. The system stops, temporarily, after each of the preceding compilations, giving the operator time to ensure that the compilations were successful. However, there is no need for the system to stop after the next few jobs. A NOHALT statement, therefore, is given at this point.



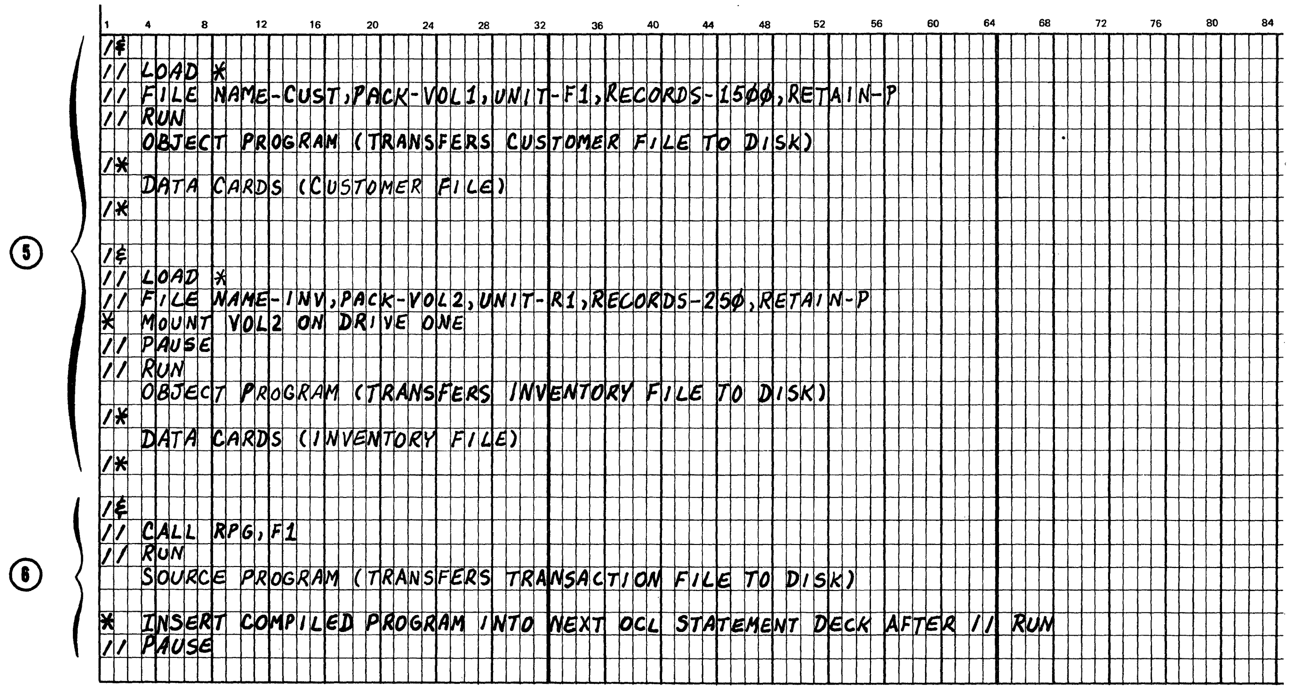
```
// DATE 10/20/71

/&
// CALL RPG,F1
// RUN
   SOURCE PROGRAM (TRANSFERS CUSTOMER FILE TO DISK)

/&
// CALL RPG,F1
// RUN
   SOURCE PROGRAM (TRANSFERS INVENTORY FILE TO DISK)
*  INSERT COMPILED PROGRAMS INTO NEXT OCL STATEMENT DECKS AFTER // RUN
// PAUSE

// NOHALT
```

Figure 24 (Part 1 of 4). OCL Statement Example

5. The two object programs previously compiled are being run to transfer the customer and inventory files, respectively, to disk.

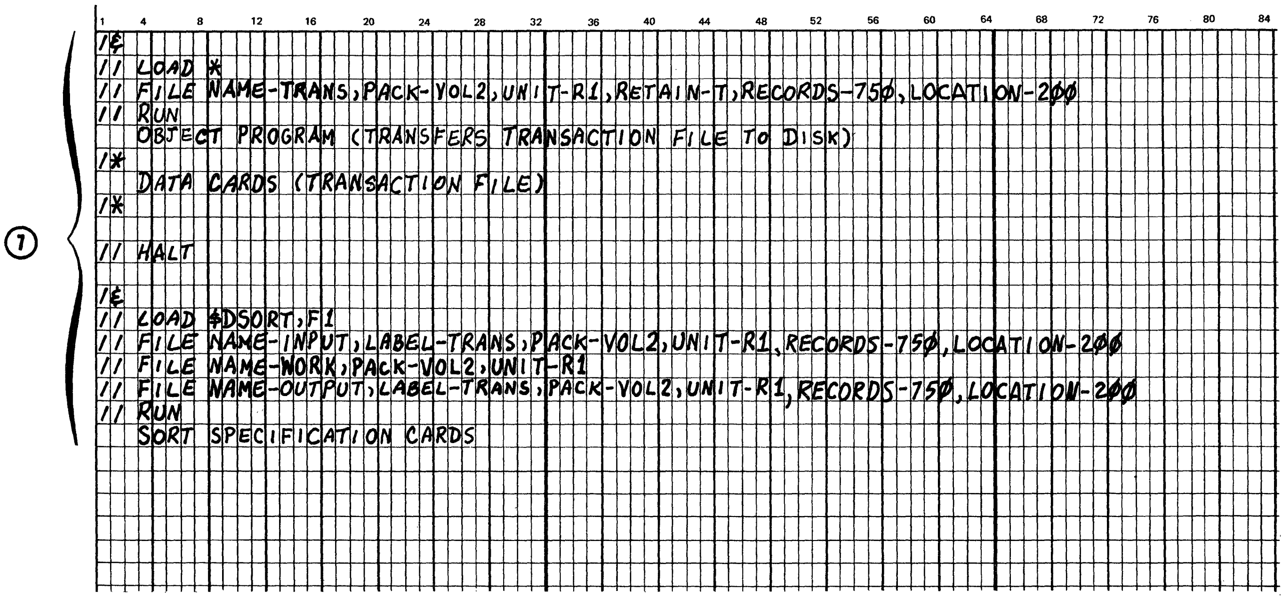
   In each case, a disk file is being created. Both files are permanent. The name that will identify the customer file on disk is CUST; the inventory-file name is INV. The date for both files will be 10/20/71.

   The cards containing the records to be transferred to disk are being read from the same device as the OCL statements. In each case, the cards must immediately follow the program that reads them. If the programs had been loaded from disk, the cards would have followed the RUN statement in each case.



```
/&
// LOAD *
// FILE NAME-CUST,PACK-VOL1,UNIT-F1,RECORDS-15ØØ,RETAIN-P
// RUN
   OBJECT PROGRAM (TRANSFERS CUSTOMER FILE TO DISK)
/*
   DATA CARDS (CUSTOMER FILE)
/*

/&
// LOAD *
// FILE NAME-INV,PACK-VOL2,UNIT-R1,RECORDS-25Ø,RETAIN-P
*  MOUNT VOL2 ON DRIVE ONE
// PAUSE
// RUN
   OBJECT PROGRAM (TRANSFERS INVENTORY FILE TO DISK)
/*
   DATA CARDS (INVENTORY FILE)
/*
/&
// CALL RPG,F1
// RUN
   SOURCE PROGRAM (TRANSFERS TRANSACTION FILE TO DISK)

*  INSERT COMPILED PROGRAM INTO NEXT OCL STATEMENT DECK AFTER // RUN
// PAUSE
```

Figure 24 (Part 2 of 4). OCL Statement Example



```
/&
// LOAD *
// FILE NAME-TRANS,PACK-VOL2,UNIT-R1,RETAIN-T,RECORDS-75Ø,LOCATION-2ØØ
// RUN
   OBJECT PROGRAM (TRANSFERS TRANSACTION FILE TO DISK)
/*
   DATA CARDS (TRANSACTION FILE)
/*

// HALT

/&
// LOAD $DSORT,F1
// FILE NAME-INPUT,LABEL-TRANS,PACK-VOL2,UNIT-R1,RECORDS-75Ø,LOCATION-2ØØ
// FILE NAME-WORK,PACK-VOL2,UNIT-R1
// FILE NAME-OUTPUT,LABEL-TRANS,PACK-VOL2,UNIT-R1,RECORDS-75Ø,LOCATION-2ØØ
// RUN
   SORT SPECIFICATION CARDS
```

Figure 24 (Part 3 of 4). OCL Statement Example

6. A program that transfers a transaction file, TRANS, from cards to disk is being compiled. Because the resulting object-program cards are to be placed with the next set of OCL statements, comment and PAUSE statements are used to remind the operator.

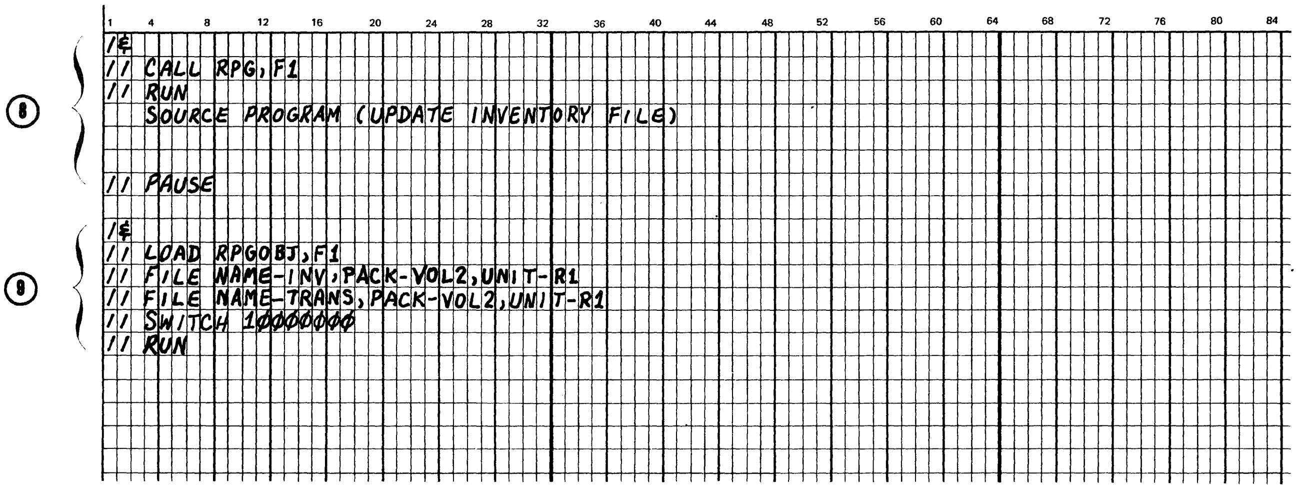
7. The transaction file is first transferred from cards to disk, and then sorted on disk by the Disk Sort program. A HALT statement precedes the sort job so that the system will stop after the sort job. This gives the operator a chance to check any diagnostic messages to ensure that the sort was successful. The HALT statement remains in effect for the remaining jobs.

   The INPUT and OUTPUT files are the same. The transaction file is read, sorted, and then written back on the same area of disk.

   The sort specification cards following the RUN statement are input to the Disk Sort program. Like all input, the last card must be a /* card.

8. The program that updates the inventory file with information from the transaction file is compiled. Comment and PAUSE statements again remind the operator to include the object-program cards with the next set of OCL statements.

9. The program just compiled is run to update the inventory file. This program can also print the transaction-file records. The printed output file, however, is conditioned by external indicator U1. Because the SWITCH statement sets U1 on, the transaction records will be printed. If the SWITCH statement had not been used, the indicator would have remained off and the records would not have been printed (external indicators are all initialized off at IPL time).



8

```
/&
// CALL RPG,F1
// RUN
   SOURCE PROGRAM (UPDATE INVENTORY FILE)


// PAUSE
```

9

```
/&
// LOAD RPGOBJ,F1
// FILE NAME-INV,PACK-VOL2,UNIT-R1
// FILE NAME-TRANS,PACK-VOL2,UNIT-R1
// SWITCH 1ØØØØØØØ
// RUN
```

Figure 24 (Part 4 of 4). OCL Statement Example

PART II. DISK UTILITY PROGRAMS

INTRODUCTION TO DISK UTILITY PROGRAMS

The Disk System includes a group of disk resident utility programs. These programs do a variety of necessary jobs, from preparing disks for use to maintaining the system libraries. 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
- 5445 Data Interchange Utility

The information for every program is divided into five sections:

- Control statement summary
- Parameter summary
- Parameter descriptions
- OCL (operation control language) considerations
- Examples

TO WRITE UTILITY CONTROL STATEMENTS

To write utility control statements (see *Control Statements),* use the sections in the following way:

1. Look at the *Control Statement Summary* to determine which control statements and parameters apply to the program uses 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 statements and utility programs for specific jobs.

5. To find information concerning the use of utility programs on disk refer to *OCL Considerations* for the necessary OCL statements.

Control Statements

All of the programs require utility control statements, which you must supply. These statements give the program information concerning the output you want the program to produce or the way in which you want the program to perform its function. The programs read these statements from the system input device. They must be the first input read by the programs.

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. 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.

Coding Rules

The rules for constructing control statements are as follows:

1. *Statement identifier.* // followed by a blank should precede the statement identifier. Do not use blanks within the identifier.

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

3. *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.

4. *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).

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

The following is an example of a control statement:

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

The statement identifier is COPY. The parameter keywords are FROM, LIBRARY, NAME, and TO. The information you supply is F1, O, System, and R1.

End Control Statement

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

SPECIAL MEANING OF CAPITAL LETTERS, NUMBERS, AND SPECIAL CHARACTERS

Capitalized words and letters, numbers, and special characters have special meanings in OCL and utility control statement descriptions.

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 that can be used are listed in the parameter summaries for the control statements.

Braces ($\{\ \}$) sometimes appear in parameters shown in control statement summaries and parameter summaries. 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- $\left\{\begin{matrix}\text{T}\\\text{P}\end{matrix}\right\}$ means you can use either RETAIN-T or RETAIN-P.

DISK INITIALIZATION PROGRAM

All disks must be initialized before use. Disks that have been initialized need not be re-initialized unless you want to erase their contents and rename them.

The Disk Initialization program prepares disks for use. It does this by:

- Writing track and sector addresses on the disk.
- Checking for defective tracks, a process called surface analysis.
- Assigning alternate tracks to any defective tracks found.
- Writing a name on each disk to identify the disk.
- Formatting the volume table of contents.

The process is called initialization. The program can initialize up to five disks during the same program run.

There are three types of initialization: primary, secondary, and clear. Primary is used to initialize any disk to disk drive capacity. Secondary is used only when using the 5444 disk and only when the drive capacity of your system is increased and you have programs and data on your disks that you want to keep. Clear is used to unconditionally initialize a disk.

CAUTION

Clear will destroy any files or libraries that were previously on disk.

The control statements you supply for the Disk Initialization program depend on the type of initialization and the number of disks you are initializing.

CONTROL STATEMENT SUMMARY

Type of Initialization — *Control Statements* ①

Primary②:

New Disks

```
                                                        ⑥
// UIN TYPE-PRIMARY③,UNIT-{ code  } ,VERIFY-number ,CAP- { HALF }
                          { 'codes'}                      { FULL }
                                                ⑦
// VOL PACK-name,ID-characters,NAME360-characters

// END
```

Disk already in use (reinitialize)

```
                                                                          ⑥
// UIN TYPE-PRIMARY,UNIT-{ code  } ,VERIFY-number,ERASE-{ NO  } ,CAP- { HALF }
                         { 'codes'}                      { YES }       { FULL }
                                                ⑦
// VOL PACK-name,ID-characters,NAME360-characters

// END
```

Secondary④:

Disk already in use

```
// UIN TYPE-SECONDARY,UNIT-{ code  } ,VERIFY-number
                           { 'codes'}
// END
```

Clear⑤:

```
                                                    ⑥
// UIN TYPE-CLEAR,UNIT-{ code  } ,VERIFY-number,CAP- { HALF }
                       { 'codes'}                     { FULL }
                                                ⑦
// VOL PACK-name,ID-characters,NAME360-characters

// END
```

① Control statements are required in the order they are listed: UIN, VOL, END or UIN, END.

② 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.

③ If the TYPE parameter is omitted, TYPE-PRIMARY is assumed.

④ VOL statements are not required for secondary initialization because the disks are already named.

⑤ If the TYPE parameter CLEAR is selected, ERASE-YES is assumed.

⑥ CAP-FULL should not be used on a half capacity system and can only be used on the 5444 disk.

⑦ NAME360 can only be used on the 5445 disk.

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 (5444 disk only). Applies only to disks that were initialized on drives of less capacity than the drives you are now 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 disturbed.	
TYPE-CLEAR	Clear initialization. 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.	
UNIT-code	Disk location (one disk).	Possible codes: R1, F1, R2, F2 D1, D2
UNIT-'code,code'	Disk location (two disks).	
UNIT-'code,code,code'	Disk location (three disks).	
UNIT-'code,code,code,code'	Disk location (four disks).	
UNIT-'code,code,code,code, code'	Disk location (five 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 (1)	Initialize a disk to half capacity even if on a full capacity drive (5444 disk only).	
CAP-FULL (1)	Initialize a disk to full capacity (5444 disk only).	

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 (2). 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 (2). Its length must not exceed ten characters. If you omit this parameter no additional identification is written on the disk.
NAME360-characters	Additional identification for 5445 disk. The name will be placed in the System/360 format 1 DSCB. Can contain any of the standard System/3 characters except apostrophes, leading or embedded blanks, and embedded commas (2) . Its length must not exceed 44 characters. If you omit this parameter the program defaults to SYSTEM/3.DATA.

(1) The CAP keyword forces ERASE-YES. Pack is initialized to capacity of the drive if this keyword is omitted.

(2) This is due to their delimiter function.

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.

Disk Drive Capacity

Disk drives of different data-storage capacities are available for the System/3 Model 10 Disk System. The 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.

Primary Initialization

Primary initialization applies to new disks, or disks you have used but want to initialize again. The program initializes 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 the files using File Delete and the libraries using the allocate function of Library Maintenance.

Secondary Initialization (5444 Disk Only)

Secondary initialization applies to disks that were initialized on drives of less capacity than the 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 re-initialized.

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

UNIT Parameter (UIN)

The UNIT parameter (UNIT-code) tells the location of the disks you want to initialize. The program can initialize up to five 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'.
4. For four disks, use UNIT-'code,code,code,code'.
5. For five disks, use UNIT-'code,code,code,code, code'.

The codes indicate the locations of the disks:

Code	*Meaning*
R1	Removable disk on 5444 drive one
F1	Fixed disk on 5444 drive one
R2	Removable disk on 5444 drive two
F2	Fixed disk on 5444 drive two
D1	Removable disk on 5445 drive one
D2	Removable disk on 5445 drive two

For primary 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 one and the second VOL statement to the removable disk on drive two. (No VOL statements are required for secondary initialization. The disk is already named.)

You cannot initialize the pack from which you loaded the Disk Initialization program or the system pack.

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. The greater the number specified in the VERIFY parameter the longer it takes to initialize the disk.

Surface Analysis

Surface analysis is a procedure for testing the condition of tracks. It consists of writing test data on tracks, then reading the data to ensure it was recorded properly.

In judging whether or not tracks are defective, the program does surface analysis the number of times you specify in the VERIFY parameter. If you omit the VERIFY parameter, surface analysis is done once. Tracks that cause reading or writing errors any time during surface analysis are considered defective. Defective tracks can be assigned alternates. The 5444 has six alternate tracks available; the 5445 has 60. If the program finds more than 6 or 60 defective tracks respectively it considers the disk unusable and stops initializing it.

The program also considers the disk unusable if either track 0 or 1 is defective. Tracks 0 and 1 are used only by the system and cannot have alternates assigned to them. For the 5445 the program also considers the disk unusable if any tracks in cylinder 0 are defective.

Alternate Track Assignment

Alternate track assignment is the process of assigning an alternate track to a defective track. If the Disk Initialization program finds a defective track during surface analysis, it assigns an alternate track to the defective track. The alternate is, in effect, a substitute for the defective track. Any time a program attempts to use the defective track, it will automatically use the alternate instead. Each 5444 disk has six alternate tracks (tracks 2-7). Each 5445 disk has 60 alternate tracks (tracks 4000-4059).

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

ERASE Parameter (UIN)

The ERASE parameter concerns alternate track assignment. It applies only to disks that have already been initialized and used, but which you are re-initializing 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.

Defective tracks are not retested if the ERASE parameter is omitted.

CAP Parameter (UIN)

The CAP parameter (5444 disk only) determines pack size when the pack 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. The use of the CAP keyword forces ERASE-YES.

PACK Parameter (VOL)

The PACK parameter (PACK-name) applies to primary initialization only. During primary 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, leading or embedded blanks, and embedded commas (due to their delimiter function). (See Appendix A for a list of standard System/3 characters.) 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, the program halts and prints a message. 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 initialization only. It enables you to include a maximum of ten characters, in addition to the disk name, to further identify a disk. The characters can be any combination of standard System/3 characters (Appendix A) except apostrophes, leading or embedded blanks, and embedded commas (due to their delimiter function). 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.

NAME360 Parameter (VOL)

The NAME360 parameter (NAME360-name) is used to specify a filename for data interchange with System/360-System/370. System/360-System/370 can use data on a System/3 disk pack by treating the pack like a file. System/3 gives a default filename of SYSTEM/3.DATA. The NAME360 parameter can be used if you would like to code a filename of your own.

NAME360 can contain any of the standard System/3 characters except apostrophes, leading or embedded blanks and embedded commas. Its length must not exceed 44 characters.

OCL CONSIDERATIONS

The following OCL statements are needed to load the Disk Initialization program.

```
// LOAD $INIT, code
// RUN
```

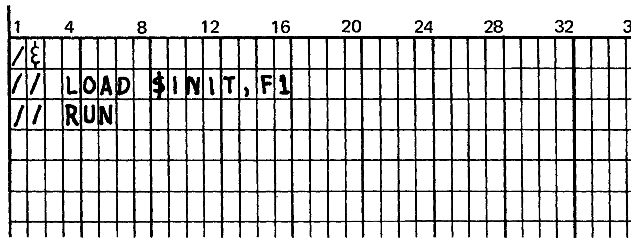
The code you supply depends on the location of the disk containing the Disk Initialization program. The codes are as follows:

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

EXAMPLES

Primary Initialization of Two Disks

Figures 25 and 26 are examples of the OCL statements and utility control statements needed for the primary initialization of two disks.

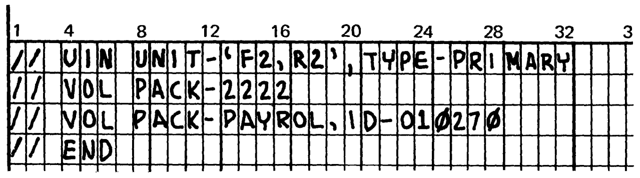


```
1  4   8   12  16  20  24  28  32  3
/&
// LOAD $INIT,F1
// RUN
```

Explanation:

- Disk Initialization program is loaded from the fixed disk on drive one.

Figure 25. OCL Load Sequence for Disk Initialization



```
1  4   8   12  16  20  24  28  32  3
// UIN UNIT-'F2,R2',TYPE-PRIMARY
// VOL PACK-2222
// VOL PACK-PAYROL,ID-01Ø27Ø
// END
```

Explanation:

- The two disks on drive two 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).

Figure 26. Utility Control Statements for Primary Initialization of Two Disks

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, F2, D1, or D2) 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: 1. Cylinder zero is defective. 2. More than 6 5444 tracks or 60 5445 tracks are defective. 3. Possible disk hardware error exists. 4. The program attempted to initialize the disk ten times without success. After this message is printed, halt 33 will occur. XX indicates the unit (R1, R2, F1, F2, D1, or D2) on which the initialization is terminated.
ALTERNATE TRACKS ASSIGNED PRIMARY TRACK XXX ALTERNATE TRACK XXX	These two messages are printed when a primary track is defective and an alternate track is assigned to it. 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 33 occurs.

ALTERNATE TRACK ASSIGNMENT PROGRAM

The Alternate Track Assignment program assigns alternate tracks to disk tracks that become defective after they are initialized. An tlternate track is a track that can be assigned to replace another track. When the program assigns an alternate, it transfers the contents of the defective track to the alternate. The 5444 has 6 alternate tracks, the 5445 has 60. An alternate track can replace any track except 0 and 1 on the 5444 or 0-19 of cylinder 0 on the 5445.

The program has three uses. The control statements you must supply depend on the program use.

The program uses and the situations to which they apply are as follows:

Program Use	**Situation**
Conditional assignment. Program tests the condition of a track and assigns an alternate to it if it is defective. (This is the normal use.)	Any time a disk track causes reading or writing errors during a job, the system halts with a code indicating that a disk error has occurred. You would now run the Alternate Track Assignment program to do conditional assignment.
Unconditional Assignment (1) Program assumes the track is defective and assigns an alternate to it without testing its condition.	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.
Cancel prior assignment. (1) Program cancels an alternate track assignment to free the alternate for use with another track.	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. 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.

(1) Whenever you request an unconditional assignment or cancel prior assignment, any pending suspected defective tracks are checked (conditional assignment).

CONTROL STATEMENT SUMMARY

Use	*Control Statements* (1)
Conditional Assignment	// ALT(2)PACK-name,UNIT-code,VERIFY-number // END
Unconditional Assignment	// ALT(2)PACK-name,UNIT-code,ASSIGN- {track / 'tracks'} ,VERIFY-number(3) // END
Cancel Prior Assignment	// ALT(2)PACK-name,UNIT-code,UNASSIGN- {track / 'tracks'} ,VERIFY-number(3) // END

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

(2) There can be only 6 ALT statements per job.

(3) The VERIFY parameter applies to the automatic conditional assignment that follows the unconditional request. (See Program Use and Situation.)

PARAMETER SUMMARY: ALT (ALTERNATE) STATEMENT

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 8-205 or 8-405 (for 5444) 20-3999 (for 5445) to identify tracks. Tracks 0-1 for the 5444 or 0-19 for the 5445 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. (1)	Use track numbers 8-405 (for 5444), or 20-3999 (for 5445) to which alternates are assigned.
UNASSIGN-'track,track,...'	Cancel two or more alternate track assignments (maximum is six). (1)	

(1) Before canceling an assignment, the program tests the condition of the track to which the alternate is assigned. The assignment is canceled if the test indicates that the track is not defective. If the test indicates that the track is defective, the program does not cancel the assignment unless the operator tells it to do so.

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. (See *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	*Meaning*
R1	Removable disk on 5444 drive one
F1	Fixed disk on 5444 drive one
R2	Removable disk on 5444 drive two
F2	Fixed disk on 5444 drive two
D1	Removable disk on 5445 drive one
D2	Removable disk on 5445 drive two

VERIFY Parameter

The VERIFY parameter (VERIFY-number) concerns conditional assignment. (See *Program Use* and *Situation* for unconditional and cancel prior assignments.) It 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.

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. Any time 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.
3. The system then halts with a halt code indicating a permanent disk I/O error. You can then run the Alternate Track Assignment program.

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.

Surface Analysis. Surface analysis is a procedure the program uses to test the condition of tracks. It 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.

Assignment of Alternate Tracks. If a track proves to be defective, the program assigns an alternate track. The alternate becomes, in effect, a substitute for the defective track. Any time a program attempts to use the defective track, it automatically uses the alternate instead.

The 5444 has 6 alternate tracks; the 5445 disk has 60. The program will not do conditional assignment if all alternate tracks are in use.

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 prints all track sectors containing data that caused reading errors. Characters that have no print symbol are printed as 2-digit hexadecimal numbers.

The following is an example:

```
ABCDE GH123 56...
      B     A
      6     4
```

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.

ASSIGN Parameter

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

For 5444, you can assign alternates to any tracks except 0-7, which are for system use only. For 5445 you can assign alternates to any tracks except 0-19 or 4000-4059; 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 (for 5444) or 20-3999 (for 5445) 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.

Unconditional Assignment

Unconditional assignment applies to tracks that occasionally cause read or write errors. Such tracks might not cause errors when tested by the Alternate Track Assignment program during conditional assignment. If they don't, the program will not assign alternate tracks to them. If you still want to assign alternates to these tracks, use unconditional assignment. In doing unconditional assignment, the program assigns alternates without first testing the condition of the tracks suspected of being defective.

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 (for 5444) or 20-3999 (for 5445) 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.

Cancel Prior Assignment

Cancelling an alternate track assignment consists of transferring the data from an alternate track back to the original track (the track to which the alternate is assigned), therefore, freeing the alternate from being the substitute for the original track.

Before transferring data back to the original track, the Alternate Track Assignment program tests the condition of the original track. If the test indicates that the track is defective, the program stops. Through the restart procedure you choose, you can tell the program to do one of four things (see *IBM System/3 Disk System Halt Procedure Guide,* GC21-7540):

1. Cancel the assignment and transfer the data back to the original track regardless of the condition of the original track.
2. Test the track again.
3. Leave the assignment as it is. If there are other tracks for which you are cancelling assignments, the program continues with those. Otherwise, it ends.
4. Cancel the job.

Cancelling assignments is not often done. It applies to cases where a defective track is found, but all six alternates are in use. To recover the data from the defective track, you might want to cancel an alternate track assignment to free the alternate track. Normally this involves copying, to another disk, a file or library entry that uses an alternate track, then freeing the alternate for use with the defective track you found.

OCL CONSIDERATIONS

The following OCL statements are needed to load the Alternate Track Assignment program.

```
// LOAD $ALT,code
// RUN
```

The code you supply depends on the location of the disk containing the Alternate Track Assignment program. 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

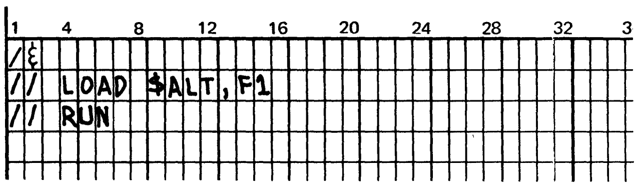
EXAMPLES

Conditional Assignment

Figures 27 and 28 are examples of the OCL statements and utility control statements needed for a conditional assignment as described in the following situation.

Situation

The sytem cancels a job if a defective track is found on the removable disk on drive one. (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.

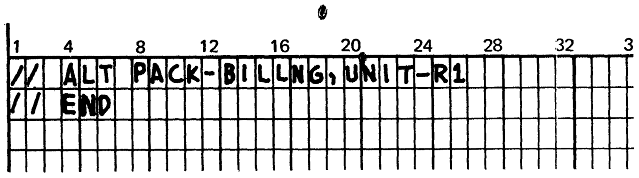


```
/&
// LOAD $ALT,F1
// RUN
```

Explanation:

- Alternate Track Assignment program is loaded from the fixed disk on drive one.

Figure 27. OCL Load Sequence for Alternate Track Assignment



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

Explanation:

- The name of the disk (BILLNG) and its location (removable disk on drive one) 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 tracks.

Figure 28. Utility Control Statements for a Conditional Assignment

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.
RECORD WITH DATA ERROR	This message is printed when the Alternate Track Assignment program found an error when transferring data. The record 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

The Alternate Track Rebuild program enables you to correct data that could not be transferred correctly to an alternate track. One or more alternate tracks can be corrected during a program run. You must supply the control statements and data used to correct the errors.

In writing control statements for this program, you will need the information printed by the Alternate Track Assignment program when it assigned the alternate track. The printed information tells you the name of the disk and numbers of the track and sectors suspected of containing incorrect data. It also includes the data from these sectors, which you can use to locate incorrect data. On the 5445, fixed record refers to a physical 256-byte record, similar to the sector on the 5444

CONTROL STATEMENT SUMMARY (1)

```
// REBUILD PACK-name,UNIT-code,TRACK-location, LENGTH-number,DISP-position

Substitute data

// END
```

(1) 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
data                          for positions 1-78
// END

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

PARAMETER AND SUBSTITUTE DATA SUMMARY

REBUILD Statement

PACK-name	Name of the disk.
UNIT-code	Location of the disk. Possible codes are R1, F1, R2, F2, D1, D2.
TRACK-location	*5444 Disk Unit*—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 9). *5445 Disk Unit*—Number of track and fixed record containing incorrect data. Number is printed by Alternate Track Assignment program. Track number must be four digits; fixed Record number must be two digits. (TRACK-011109 means track 111, fixed record 9).
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.)

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 see if 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	*Meaning*
R1	Removable disk on 5444 drive one
F1	Fixed disk on 5444 drive one
R2	Removable disk on 5444 drive two
F2	Fixed disk on 5444 drive two
D1	Removable disk on 5445 drive one
D2	Removable disk on 5445 drive two

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.

For the 5444 disk, 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.

For the 5445 disk, the possible track numbers are 0020-3999. Always use four digits. The possible fixed record numbers are 01-19. Always use two digits. The track number must precede the fixed record number. For example, the parameter TRACK-111019 means track 1110, record 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 or fixed record. 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 or fixed record.

Length applies to characters that occupy consecutive positions in the sector or fixed record. If the characters you want to replace do not occupy consecutive positions, you must either replace all intervening characters or use more than one REBUILD statement. For example, to replace characters 10-11 and 24-25 in a sector or fixed record, 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 or fixed record. The position of the first character is 1; the position of the second character is 2, and so on. The maximum position you can specify 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 code the substitute characters that apply to that statement. The characters must be in hexadecimal form. Appendix A shows the hexadecimal codes for the System/3 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.

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

Note: If the LENGTH parameter of the REBUILD statement exceeds 38, at least two substitute data cards are required. Each substitute data card, except the last one, must be completely filled with data and must have a comma in column 95 and a blank in column 96.

OCL CONSIDERATIONS

The following OCL statements are needed to load the Alternate Track Rebuild program.

```
// LOAD $BUILD, code
// RUN
```

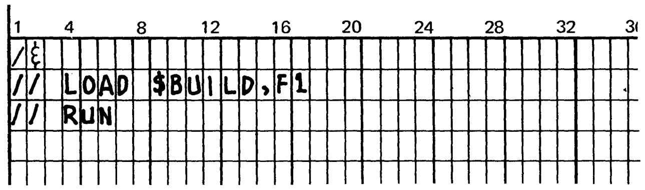
The code you supply depends on the location of the disk containing the Alternate Track Rebuild program. The codes are as follows:

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

EXAMPLES

Correcting Characters on an Alternate Track

Figures 29 and 30 are examples of the OCL and utility control statements needed for correcting characters on an alternate track.



```
/&
// LOAD $BUILD,F1
// RUN
```

Explanation:

- Alternate Track Rebuild program is loaded from the fixed disk on drive one.

Figure 29. OCL Load Sequence for Alternate Track Rebuild

```
1  4   8   12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
// REBUILD PACK-BILLNG,UNIT-R1,TRACK-Ø2ØØØ,LENGTH-4,DISP-12Ø
C7C8,C9F1
// END
```

Explanation:

- The name of the removable disk (BILLNG) and its location (drive one) are indicated in the PACK and UNIT parameters in the REBUILD statement.
- The sector containing the incorrect characters is sector 0 of the alternate track assigned to track 20 (TRACK-02000). The character in position 120 is the first character being replaced (DISP-120).
- The characters in positions 120 through 123 in sector 0 are being replaced (LENGTH-4).
- The substitute characters follow the REBUILD statement. They are G (C7), H (C8), I (C9), and 1 (F1).

Figure 30. Utility Control Statements for Correcting Characters on an Alternate Track

Situation

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

```
**SECTOR WITH DATA ERROR**

TRACK    1.......10........20........30........40........50........60........70........80......88
02000                    Z  ABCDEFGHI       JKLMNOPQR        STUVWXYZ    0123456789
         FFFFFF903B524677 DC         CCCCCCD         DDDDDDEE        EEEEEE          FFFFFF000000
         FEDCBAFBEDFEF705 F0         ABCDEF0         ABCDEF01        ABCDEF          ABCDEF000000

         0000000000000000000000000000002000100008300000000000000000000000000000000000000000000000
         00000000000000000000000000000024888C2100100000000000000000000000000000000000000000000000

         0000000000000000000000000000000000000000000000000000000000000000000000000000000C
         000000000000000000000000000000000000000000000000000000000000000000000000000000005A
```

55202A

It means that errors were detected in sector 0 of track 20. (Assume the name of the disk is BILLNG.)

In checking the characters printed by the program, you found that the characters in positions 120-123 in the sector are incorrect and you want the operator to run the Alternate Track Rebuild program to correct them.

FILE AND VOLUME LABEL DISPLAY PROGRAM

The File and Volume Label Display program has two uses:

1. Print the entire Volume Table of Contents (VTOC) from a disk.

2. Print only the VTOC information for certain data files.

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

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.

The control statements you supply for the program depend on the program use.

CONTROL STATEMENT SUMMARY

Uses	*Control Statement* (1)
Print entire VTOC	// DISPLAY UNIT-code, LABEL-VTOC // END
Print only file information from VTOC	// DISPLAY UNIT-code, LABEL { filename / 'filenames' } (2) // END

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

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

PARAMETER SUMMARY (DISPLAY STATEMENT)

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. (1)

(1) 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	*Meaning*
R1	Removable disk on 5444 drive one
F1	Fixed disk on 5444 drive one
R2	Removable disk on 5444 drive two
F2	Fixed disk on 5444 drive two
D1	Removable disk on 5445 drive one
D2	Removable disk on 5445 drive two

LABEL Parameter

The LABEL parameter indicates the information you want 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.

Entire Contents of VTOC

The parameter LABEL-VTOC means to print the entire contents of the VTOC. The meaning of the information the program prints is given in the following chart. Headings that are listed are the ones printed by the program to identify the information. Figures 31 and 32 are examples of VTOC printouts.

If the program needs more than one page to list the file information it prints the headings for the file information at the top of each new page.

```
PACK-111111           ID-ANDERSON

NO. OF ALTERNATE TRACKS AVAILABLE-2

TRACKS WITH ALTERNATE ASSIGNED-302,200

DEFECTIVE ALTERNATE TRACKS-3,5

DEVICE CAPACITY-400

LIBRARY EXTENT--  START  END  EXTENDED END
                   008   027       027

AVAILABLE SPACE ON PACK
   LOCATION    TRACKS
     028        367
     399        001
     401        001


PACK-111111    UNIT-R1     DATE 11/11/70
  FILE         FILE    KEEP FILE  REC    KEY   KEY    NEXT AVAIL   NEXT AVAIL     INDEX        DATA    VOL
  NAME         DATE    TYPE TYPE  LEN    LEN   LOC      RECORD        KEY       START END   START END  SEQ

COST         09/21/71   T    C    0128                405/11/129                             405  405   00
MASTER       ~~03/14/71~~   P    C    0128                404/11/129                             404  404   00
EMPLOYEE     12/07/70   P    I    0128   05    0005      ****      402/C1/129   402   402    403  403   02
UPDATE       ~~09/14/71~~   T    I    0128   05    0005   396/11/129   395/0C/185   395   395    396  396   00
PARTS        08/09/71   T    D    0128                   ****                                400  400   01
SERIAL       ~~08/16/71~~   T    C    0128                398/11/129                             398  398   00
ADDRESS      09/21/71   T    C    0080                397/06/065                             397  397   00
BACKUP       09/29/71   S    C    0128                399/11/129                             399  399   00
```

55201A

Figure 31. VTOC Printout Example

```
PACK-D1D1D1            ID-

NO. OF ALTERNATE TRACKS AVAILABLE-60

AVAILABLE SPACE ON PACK

   LOCATION   TRACKS

    001/00     3941

    199/00     0002


PACK-D1D1D1     UNIT-D1   DATE 09/29/71
  FILE     FILE   KEEP FILE REC  KEY KEY     NEXT AVAIL       NEXT AVAIL        INDEX           DATA       VOL
  NAME     DATE   TYPE TYPE LEN  LEN LOC       RECORD            KEY        START    END    START   END    SEQ

COST     09/21/71 T    D    0128              *****                                       199/10 199/19 00
MASTER   03/14/71 P    D    0096              *****                                       199/08 199/09 00
EMPLOYEE 12/07/70 P    I    0050 05  0006 199/03/01/101 199/02/01/019 199/02  199/02  199/03 199/07 00
PARTS    08/09/71 T    D    0256              *****                                       198/18 198/19 00
ADDRESS  09/21/71 T    C    0030          198/16/01/061                                   198/16 198/17 01
SERIAL   08/16/71 T    C    0100          198/01/01/201                                   198/01 198/15 00
UPDATE   09/14/71 S    I    0200 03  0007 199/01/02/145 199/00/01/015 199/00  199/00  199/01 199/01 00
```

Figure 32. VTOC Printout Example of 5445 Disk

MEANING OF VTOC INFORMATION

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) - 5444 disk only.
LIBRARY EXTENT	Boundary of libraries on the disk. (If the 5444 disk contains no libraries, these headings are not printed.)
START	Track on which library begins. If 5444 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 (5444 disk 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 (5444). First cylinder/track in available area (5445).
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	Program level 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:
	P = permanent
	T = temporary
	S = scratch

FILE TYPE	File type:
	I = indexed
	C = consecutive
	D = direct
	SC = split cylinder, consecutive
	SD = split cylinder, direct

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.

Heading	*Meaning*
NEXT AVAIL RECORD	Beginning location of next available record in file. For 5444 disk, location is track, sector, and position within sector. For 5445 disk, location is cylinder, track, fixed record, and position within record. EXAMPLE: 099/18/006 = track 99, sector 18, position 6.(1) 050/02/12/006 = cylinder 50, track 2, fixed record 12, position 6.(1)
NEXT AVAIL KEY	Indexed files only. Beginning location of next available record key in index portion of file. For 5444 disk, location is track, sector, and position within sector. For 5445 disk, location is cylinder, track, fixed record, and position within record. EXAMPLE: 090/10/006 = track 90, sector 10, position 6.(2) 052/03/10/006 = cylinder 52, track 3, fixed record 10, position 6.(2)
INDEX START END	Indexed files only. For 5444 disk, tracks on which index starts (START) and ends (END). For 5445 disk, cylinder/track on which index starts (START) and ends (END).
DATA START END	Disk area reserved for the file. START is the first 5444 track or 5445 cylinder/track of the area. END is the last 5444 track or 5445 cylinder/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.

(1) 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 ****.

(2) 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 ****.

File Information Only

The parameter LABEL-filename or LABEL-'filenames' means to print certain file information from the VTOC. For one file, use LABEL-filename; for two files, use LABEL-'filename,filename'; and so on. (Use the names that identify the files in the VTOC.) You can list 20 filenames for a program run. The statement length, however, is restricted to 96 characters.

The program prints the file information for each of the files you list. This is the information described for the headings PACK name and FILE LABEL in the chart, *Meaning of VTOC Information.*

If the program needs more than one page to list the file information, it prints headings for the file information at the top of each new page.

OCL CONSIDERATIONS

The following OCL statements are used to load the the File and Volume Label Display program.

```
// LOAD $LABEL,code
// RUN
```

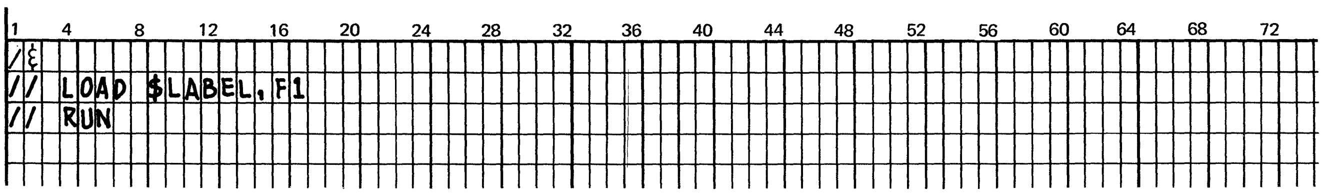
The code you supply depends on the location of the disk containing the utility program. The codes are as follows:

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

EXAMPLES

Printing VTOC Information for Two Files

Figures 33 and 34 are examples of the OCL statements and utility control statements needed to print VTOC information for two files.

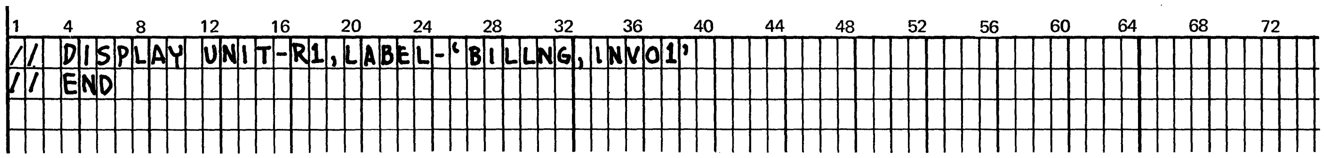


```
/&
// LOAD $LABEL,F1
// RUN
```

Explanation:

- The File and Volume Label Display program is loaded from the fixed disk on drive one.

Figure 33. OCL Load Sequence for File and Volume Label Display



```
// DISPLAY UNIT-R1,LABEL-'BILLNG,INVO1'
// END
```

Explanation:

- The files for which information is printed are named BILLNG and INVO1 (LABEL-'BILLNG,INVO1' in DISPLAY statement). They are located on the removable disk on drive one (UNIT-R1).

Figure 34. Utility Control Statements for Printing VTOC Information for Two Files

FILE DELETE PROGRAM

The File Delete program has three uses:

- Remove all files from a disk.
- Remove only the files you name.
- Scratch file references in the Volume Table of Contents (VTOC). Deleting files frees the space they occupy for use by new files.

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.

The control statements you supply for the File Delete program depend on the function to be performed.

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. 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. If a REMOVE statement is used, files are erased from the disk. No file is physically scratched or removed from the VTOC until end of job has occurred.

CONTROL STATEMENT SUMMARY

Use	*Control Statements* [1]
Scratch all files in the VTOC.	// SCRATCH PACK-name, UNIT-code, LABEL-VTOC // END
Scratch only one file in the VTOC.	// SCRATCH PACK-name, UNIT-code, LABEL-filename, DATE-date [2]
Scratch multiple files in the VTOC	// SCRATCH PACK-name, UNIT-code, LABEL- $\begin{Bmatrix}\text{filename}\\ \text{'filenames'}\end{Bmatrix}$
Remove all files from disk	// REMOVE PACK-name, UNIT-code, LABEL-VTOC, DATA- $\begin{Bmatrix}\text{NO}\\ \text{or}\\ \text{YES}\end{Bmatrix}$ // END
Remove only the files named from disk	// REMOVE PACK-name, UNIT-code, LABEL- $\begin{Bmatrix}\text{filename}\\ \text{'filenames'}\end{Bmatrix}$ DATE-date, DATA- $\begin{Bmatrix}\text{NO}\\ \text{or}\\ \text{YES}\end{Bmatrix}$ // END

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

[2] 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. At least one SCRATCH or REMOVE statement is required by the program. When deleting files, you can list as many filenames as the statement will hold. The statement length, however, cannot exceed 96 characters. If you want to delete more files than you can specify in one SCRATCH or REMOVE statement, use additional statements. The END statement must follow the last SCRATCH or REMOVE statement.

PARAMETER SUMMARY		
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. (1)
LABEL-'filename,filename,...	Scratch or remove only the files named in the VTOC.	
DATE-date (2)	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 (3)	

(1) These are the names you gave the files when you placed them on disk.

(2) If the pack has more than one file with the name you list in the LABEL parameter, they will all be deleted unless you use the DATE keyword and parameter to indicate a particular file. If the DATE keyword is used, only one filename can be given in the LABEL parameter for that control statement. (The DATE parameter must be in the same format as the system date.)

(3) If YES is used, then all files specified will be deleted from the VTOC and the disk. A message will be printed on the Syslog device for each file removed. YES is not allowed on a SCRATCH statement. NO is the default value. If NO is used, all files specified will be deleted from the VTOC but *not* deleted from the 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	*Meaning*
R1	Removable disk on 5444 drive one
F1	Fixed disk on 5444 drive one
R2	Removable disk on 5444 drive two
F2	Fixed disk on 5444 drive two
D1	Removable disk on 5445 drive one
D2	Removable disk on 5445 drive two

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.

Form	*Files Deleted*
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 40.

DATE Parameter

The DATE parameter can only be used with LABEL-filename. 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 6-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 they were specified when you placed the file on disk.

DATA Parameter

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

If NO is coded in this parameter, then the file specified will not be removed from the disk, but any reference to it in the VTOC will be removed. If neither YES or NO is specified, NO is used as the default condition.

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 Syslog device for each file removed from the disk in this format:

'DATA REMOVED FOR FILE___DATE 000000'

The DATA parameter may be used on a SCRATCH statement but only NO may be coded. If YES is coded on a SCRATCH statement, an error will occur.

OCL CONSIDERATIONS

The following OCL statements are needed to load the File Delete program:

```
// LOAD $DELET,code
// RUN
```

The code you supply depends on the location of the disk containing the utility program. 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

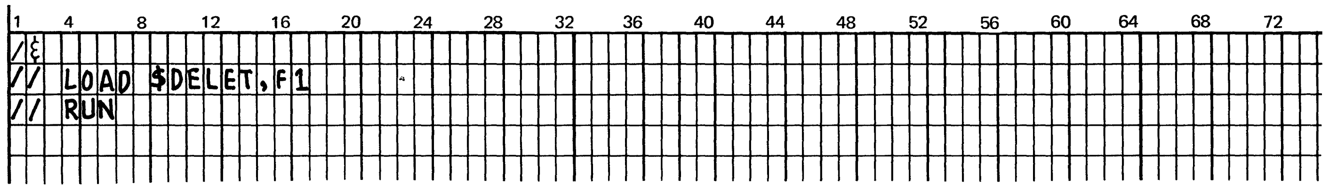
EXAMPLES

Deleting One of Several Files Having the Same Name

Figures 35, 36, and 37 are examples of the OCL statements and utility control statements needed to delete one of several files having the same name as described in the following situation.

Situation

Assume that three files on a removable disk have the same name: INVO1. The dates of these files are 6/16/70, 8/18/70, and 11/15/70. You want to delete the version dated 6/16/70.

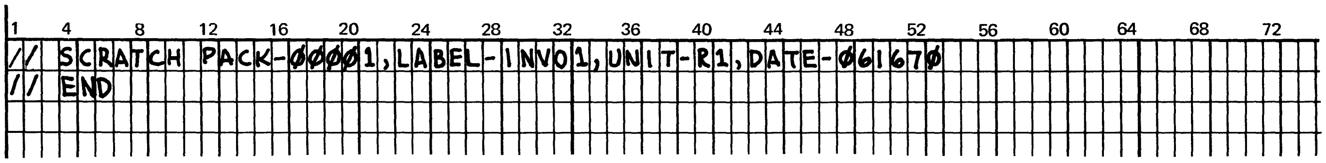


```
/&
// LOAD $DELET,F1
// RUN
```

Explanation:

- File Delete program is loaded from the fixed disk on drive one.

Figure 35. OCL Load Sequence for File Delete



```
// SCRATCH PACK-ØØØØ1,LABEL-INVO1,UNIT-R1,DATE-Ø6167Ø
// END
```

Explanation:

- Disk that contains the file being deleted is named 00001 (PACK-00001 in SCRATCH statement).
- Because two other files have the name INVO1, the date (061670) is needed to complete the identification of the file you want to delete (LABEL-INVO1 and DATE-061670).
- The removable disk containing the file to be deleted is on drive one (UNIT-R1).

Figure 36. Utility Control Statements to Delete One Version of a File

```
1  4   8   12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
// REMOVE PACK-ØØØØ1,LABEL-INVO1,UNIT-R1,DATE-Ø6167Ø,DATA-YES
```

Explanation:

- A REMOVE statement is used instead of a SCRATCH statement.
- Disk that contains the file being deleted is named 00001 (PACK-00001 in REMOVE 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 one (UNIT-R1).
- The YES specification in the DATA parameter will delete all data from the disk containing information on the specified file.

Figure 37. Utility Control Statement to Delete One Version of a File Using a REMOVE Statement

The Disk Copy/Dump program has three general uses. The control statements you must supply depend on the program use.

The program uses and most common reasons for them are as follows:

Program Use	Common Reasons
Copy entire contents of one disk to another.	Provide a reserve disk in case something happens to the original disk. Important disks, such as those containing your libraries and permanent data files, are normally the ones you would copy.
Copy a data file from one disk to another, or from one area to another on same disk.	Any of the following: ● Provide a reserve file in case something happens to the original file. ● Move a file to a larger disk area. ● Re-organize the data portion of an indexed file. (Data in the copy of the file is re-organized; the original file is unchanged.) ● Delete records from a file. (Records are omitted from the copy of the file; the original file remains unchanged.)
Print all or part of a data file.	Provide a printed copy of the records in a file, perhaps for use in checking the records for errors.

The OCL sequence used to load the program describes the disk file being copied or printed. If you are copying the file to disk, the file being created must also be described in the OCL sequence.

CONTROL STATEMENT SUMMARY

```
Uses [1]                 Control Statements [2]

Copy an Entire Disk      // COPYPACK FROM-code,TO-code

                         // END
                                   { OUTPTX- }      { DELETE- }                         [3]        { NO  }   [4]        { NO  }
Copy a Data File         // COPYFILE {  -or-   } DISK, {  -or-   } 'position,character',   REORG- { -or- } ,   WORK- { -or- }  [5]
                                   { OUTPUT- }      { OMIT-   }                                    { YES }              { YES }
                         // END
                                   { OUTPTX- }      { DELETE- }                         [3]            [4]        { NO  }
Copy and Print           // COPYFILE {  -or-   } BOTH, {  -or-   } 'position,character',   REORG-YES,   WORK- { -or- }  [5]
a Data File                        { OUTPUT- }      { OMIT-   }                                                   { YES }
                         // END
                                   { OUTPTX- }      { DELETE- }                         [3]            [4]        { NO  }
Copy a Data File,        // COPYFILE {  -or-   } BOTH, {  -or-   } 'position,character',   REORG-YES,   WORK- { -or- }  [5]
But Print Only a                   { OUTPUT- }      { OMIT-   }                                                   { YES }
Part of the File         // SELECT KEY,FROM-'key' [4]                  ─┐
                                  -or-                                  │
                         // SELECT KEY,FROM-'key',TO-'key' [4]          │
                                  -or-                                  │  One of these [6]
                         // SELECT RECORD,FROM-number                   │
                                  -or-                                  │
                         // SELECT RECORD,FROM-number,TO-number         │
                         // SELECT PKY,FROM-'key' [7]                   │
                                  -or-                                  │
                         // SELECT PKY,FROM-'key',TO-'key' [7]         ─┘

                         // END
                                   { OUTPTX- }
Print an Entire Data     // COPYFILE {  -or-   } PRINT
File                               { OUTPUT- }
                         // END
                                   { OUTPTX- }
Print Only a Part        // COPYFILE {  -or-   } PRINT                 ─┐
of a Data File                     { OUTPUT- }                          │
                         // SELECT KEY,FROM-'key' [4]                   │
                                  -or-                                  │
                         // SELECT KEY,FROM-'key',TO-'key' [4]          │
                                  -or-                                  │  One of these [6]
                         // SELECT RECORD,FROM-number                   │
                                  -or-                                  │
                         // SELECT RECORD,FROM-number,TO-number         │
                         // SELECT PKY,FROM-'key' [7]                   │
                                  -or-                                  │
                         // SELECT PKY,FROM-'key',TO-'key' [7]         ─┘
                         // END
```

[1] The program uses include the possible combinations of copying and printing files.

[2] For each use, the program requires the control statements in the order they are listed: COPYPACK, END; COPYFILE, END; and COPYFILE,SELECT,END.

[3] Needed only if you want to delete a certain type of record. DELETE cannot be used with direct files.

[4] Applies only to indexed files. When OUTPUT-BOTH is specified, REORG-YES is required.

[5] 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.

[6] Identifies the portion you want to print.

[7] Index files with packed keys.

PARAMETER SUMMARY

COPYPACK Statement

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

OUTPUT-DISK	Copy the file from one disk to another, or from one area to another on the same disk.[1]
OUTPUT-PRINT	Print the entire file or only part of the file.[1]
OUTPUT-BOTH	Copy the file from one disk to another, or from one area to another on the same disk.[1] Also print the entire file or only part of it.
OUTPTX- $\left\{\begin{matrix}\text{DISK}\\\text{PRINT}\\\text{BOTH}\end{matrix}\right\}$	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.[2] OMIT causes deleted records not to be printed. Character can be any of the System/3 characters except blank, apostrophe, or comma.[3] Position can be any position in the record (the first position is 1, second 2, and so on). The maximum position is 4096.
REORG-NO[4]	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[4]	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[5]	Required for copying a file from one area to another on a removable disk on drive one. It means: do not use a work area on the fixed disk on drive one.
WORK-YES[5]	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

$\left\{\begin{matrix}\text{KEY}\\\text{PKY}\end{matrix}\right\}$,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.
$\left\{\begin{matrix}\text{KEY}\\\text{PKY}\end{matrix}\right\}$,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).[5]

[1] 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.

[2] Program prints the records it deletes.

[3] This is due to their delimiter function.

[4] REORG-NO is assumed if you omit the REORG parameter. When OUTPUT-BOTH is used for indexed files, REORG-YES is required.

[5] WORK-NO is assumed if you omit the WORK parameter.

[6] To print only one record, make the FROM and TO record keys the same.

PARAMETER DESCRIPTIONS

FROM and TO Parameters (COPYPACK)

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 unit.

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	*Meaning*
R1	Removable disk on 5444 drive one
F1	Fixed disk on 5444 drive one
R2	Removable disk on 5444 drive two
F2	Fixed disk on 5444 drive two
D1	Removable disk on 5445 drive one
D2	Removable disk on 5445 drive two

Copying Entire Disk

When copying a disk, the Disk Copy/Dump program transfers the contents of the disk to another disk. The content 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 contain 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 one when using the 5444 disk. (The system pack and the pack from which the Disk Copy/Dump program is loaded must be F1.)

To do this, the program uses available space on the fixed disk on drive one (5444 disk). 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 one. 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 is 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 Disk Copy/Dump program, or you can restore the disk by reinitializing.

After a successful copy, the copy program prints a message:

```
COPYPACK IS COMPLETE
```

Note: If you copy a disk containing an active checkpoint, that checkpoint will exist on both the FROM and TO disks. When one of the two active checkpoints is utilized to restart the checkpointed program, care must be taken to ensure that the job is not restarted a second time. To ensure that this will not occur, it is recommended that you perform IPL and load Restart ($$RSTR) from the pack containing the second active checkpoint If you then select the controlled cancel option when the Hbnn halt occurs (nn is the last requested checkpoint number), the checkpoint will be deactivated.

OUTPUT Parameter (COPYFILE)

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 OUTPTX parameter can be used to display printed output in hexadecimal values.

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.

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.

The OCL load sequence for the Disk Copy/Dump program indicates (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 one. (See description of the WORK parameter for more information.) (The system pack and the pack from which the Disk Copy/Dump program is loaded must be F1.)

In copying a file, the program can omit records. (See the description of the DELETE parameter 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 the description of the REORG parameter for more information.)

Printing Files

The program can print all or part of a data file. To print only part, the program needs a SELECT control statement. (See the description of the SELECT control statement 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. Appendix A lists the hexadecimal numbers for characters in the standard character set.

The following is an example of the way the program prints hexadecimal numbers using OUTPTX:

```
ABCDE GHIJ12345

CCCCCBCCCDFFFFF4444444
1234567891123450000000
```

The hexadecimal number B6 represents a character that has no print symbol.

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

```
(number) RECORDS PRINTED
```

DELETE Parameter (COPYFILE)

In copying a data file, the Disk Copy/Dump program can omit records of one type. The DELETE parameter identifies the type of record. 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'. *Character* is the character that identifies the records. *Position* is the position of the character in the records. For example, with the parameter DELETE-'100,X', all records with an X 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 DELETE.

The OMIT keyword can be used instead of DELETE. The deleted records are not printed if OMIT is used.

REORG (Reorganize) Parameter (COPYFILE)

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 parameter.

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 when you are both copying and printing an indexed file (OUTPUT-BOTH). However, the REORG parameter does not apply to copying temporary entries.

WORK Parameter (COPYFILE)

The WORK parameter applies to copying a data file from (1) one removable disk to another using the same disk drive (WORK-YES), or (2) one area to another on a removable disk on drive one (WORK-NO). It tells the program whether or not to use a work area on the fixed disk on drive one.

The parameter WORK-YES means to use a work area. WORK-NO means not to use a work area. WORK-NO is assumed if you omit the WORK parameter.

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.

If you are copying on 5445 drive one, the work area will be on R1. R1 must contain a minimum of 198 contiguous unused tracks. It is recommended, however, that R1 contain no files or libraries as the number of pack changes on D1 will decrease with an increase in work area space. You cannot copy split cylinder files from D1 to D1 using WORK-YES.

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 one. 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 one.

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 one, use the WORK-NO parameter. This keeps the program from using a work area on the fixed disk when it transfers the file from one area to the other.

SELECT KEY and SELECT PKY Parameters (SELECT)

The SELECT KEY and SELECT PKY parameters apply to printing part of an indexed file. The SELECT PKY parameter applies to printing part of an index file which contains packed keys. 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 keys between those two in the file index identify 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.

You can use fewer characters in the FROM or TO parameter than are contained in the actual keys; when keys are packed, however, you must use the same number of characters as contained in the actual keys. If you use fewer characters, the program ignores the remaining characters in the record key. The number of characters used in the FROM and TO parameters need not be the same.

For example, assume that the following area 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 in the FROM parameter.

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 keys beginning with the characters A8.

SELECT RECORD Parameters (SELECT)

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.

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. If you want to print only one record, use the same number in the FROM and TO parameters.

COPYING MULTIVOLUME FILES

When copying multivolume 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 either condition is not satisfied, a halt occurs.

Maintaining Proper Volume Sequence Numbers

To maintain proper volume sequence numbers when copying a multivolume 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 (volume 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 the 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.

Maintaining Correct Relative Record Numbers

To maintain correct relative record numbers when copying one volume of a multivolume direct file, the size of the output volume must be the same as the size of the input volume. (If you want to increase the size of a file, you must copy the entire file.) If 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, to maintain the correct relative record numbers, output and input volume extents must be equal if you are copying only one volume of a multivolume direct file.

Direct File Attributes

If you copy an entire multivolume direct file in one run, the output file will be given consecutive attributes in the Volume Table of Contents (VTOC). However, this does not effect file processing. A file with either consecutive or direct attributes can be accessed by a consecutive or direct access method. If only one volume is copied, the direct attribute will be maintained.

Copying Multivolume Indexed Files

If you want to copy a multivolume file, REORG-YES must be given in the FILE statement. Since an unordered load to a multivolume indexed load is not permitted, a REORG-NO will cause a halt if an out of sequence record is encountered. If you would prefer not to reorganize the file, it must be copied one volume at a time. When copying one volume at a time, the HIKEY on the output volume must be the same as the HIKEY on the input volume. Making the HIKEYs the same will ensure that both the input and output volumes are the same length and no records will be lost. When copying one volume of a multivolume index file, either REORG-YES or REORG-NO may be specified.

OCL CONSIDERATIONS

The following OCL statements are needed to load the Disk Copy/Dump program, if you are using the program to copy an entire disk.

```
// LOAD $COPY,code
// RUN
```

The code you supply depends on the location of the disk containing the Disk Copy/Dump program. 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

If you are copying or printing files you must (1) describe the disk files being copied or printed and (2) describe the file being created. To do this, the following OCL statements are needed in the load sequence:

```
// LOAD $COPY,code

// FILE NAME-COPYIN,UNIT-code, PACK-diskname, LABEL-filename

// FILE NAME-COPYO, UNIT-code, PACK-diskname, LABEL-filename,
```

$$// \left\{ \begin{matrix} \text{TRACKS-number} \\ \text{RECORDS-number} \end{matrix} \right\} \text{,RETAIN-code}$$

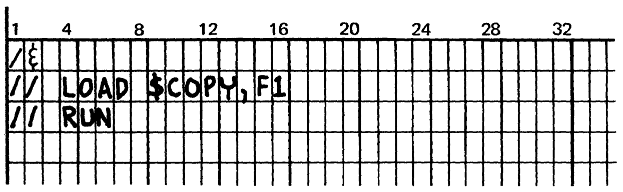
```
// RUN
```

Statement Entry	*Considerations*
// LOAD	– –
$COPY	Name of Disk Copy/Dump program.
code	Location of disk containing Disk Copy/Dump program. Can be R1, R2, F1, F2.
// FILE	– –
NAME-COPYIN	Name Disk Copy/Dump program uses to refer to file to be copied (input file).
UNIT-code	Location of disk containing file to be copied. Can be R1, R2, F1 F2, D1, D2.
PACK-diskname	Name of disk containing file to be copied.
LABEL-filename	Name by which file to be copied is identified on disk.

Statement Entry	*Considerations*
// FILE	– –
NAME-COPYO	Name Disk Copy/Dump program uses to refer to output file being created.
UNIT-code	Location of disk on which output file is to be created. Can be R1, R2, F1, F2, D1, D2.
PACK-diskname	Name of disk on which output file is to be identified on disk.
LABEL-filename	Name by which output file is to be identified on disk.
TRACKS-number RECORDS-number	Size of output file expressed either as number of records (RECORDS) or number of disk tracks (TRACKS).
RETAIN-code	Designation (temporary, permanent, or scratch) of output file. Can be T, P, or S.
// RUN	– –

EXAMPLES

Figures 38-43 are three examples of the OCL statements and utility control statements needed to (1) copy an entire disk, (2) copy a file from one disk to another and (3) print part of a file. Each of the three examples has two figures.

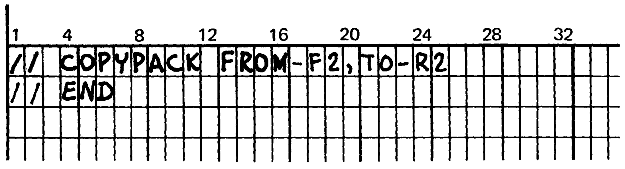


```
/&
// LOAD $COPY,F1
// RUN
```

Explanation:

- The Disk Copy/Dump program is loaded from the fixed disk on drive one.

Figure 38. OCL Load Sequence for Copying an Entire Disk

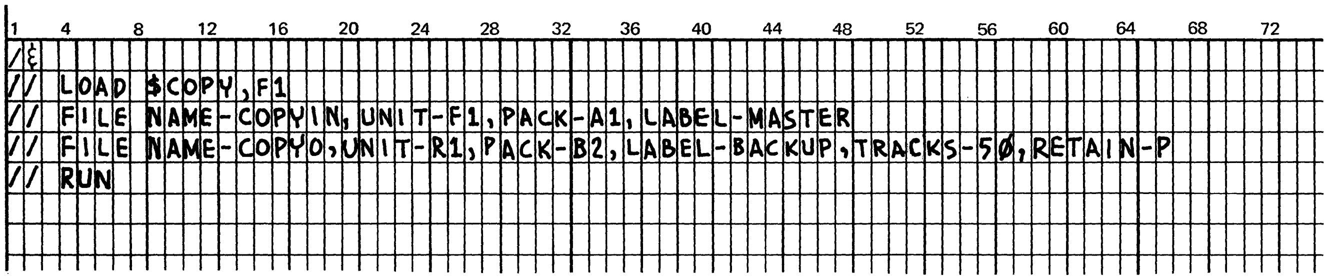


```
// COPYPACK FROM-F2,TO-R2
// END
```

Explanation:

- The contents of the fixed disk on drive two (FROM-F2 in COPYPACK statement) is copied onto the removable disk on drive two (TO-R2).

Figure 39. Utility Control Statements for Copying an Entire Disk



```
/&
// LOAD $COPY,F1
// FILE NAME-COPYIN,UNIT-F1,PACK-A1,LABEL-MASTER
// FILE NAME-COPYO,UNIT-R1,PACK-B2,LABEL-BACKUP,TRACKS-5Ø,RETAIN-P
// RUN
```

Explanation:

- Disk Copy/Dump program is loaded from fixed disk on drive one.
- 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 one (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 one (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 (TRACKS-50).

Figure 40. OCL Load Sequence for Copying a File from One Disk to Another

```
// COPYFILE OUTPUT-DISK
// END
```

Explanation:

- 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.

Figure 41. Utility Control Statements for Copying a File from One Disk to Another

```
/&
// LOAD $COPY,F1
// FILE NAME-COPYIN,UNIT-R1,PACK-B2,LABEL-BACKUP
// RUN
```

Explanation:

- Disk Copy/Dump program is loaded from the fixed disk on drive one.

- 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 one (UNIT-R1). Its name is B2 (PACK-B2).

Figure 42. OCL Load Sequence for Printing Part of a File

```
// COPYFILE OUTPUT-PRINT
// SELECT KEY,FROM-'ADAMS',TO-'BAKER'
// END
```

Explanation:

- 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).

Figure 43. Utility Control Statements for Printing Part of a File

LIBRARY MAINTENANCE PROGRAM

The Library Maintenance program has four functions:

Function	*Meaning*
Allocate	Create (reserve space for), delete, re-organize, and change the sizes of libraries.
Copy	Place entries in, and display the contents of, libraries.
Delete	Delete library entries.
Rename	Change the names of library entries.

The control statements you must supply depend on the function you are using.

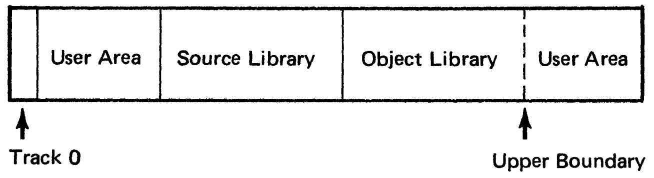
LIBRARY DESCRIPTION

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.

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 executable object programs.) *Routines* are programs and subroutines that need further translation before being loaded for execution. (They are sometimes called nonexecutable object programs.)

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

Object Library

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. But 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 a permanent entry is deleted and replaced with a permanent entry using fewer sectors. The Library Maintenance program scans the library to see what sectors are available. The entry is then placed into the gap that has the fewest sectors over and above the number required by the new entry. If the entry is the same size, no sectors are lost, but all temporary entries are deleted.

If the number of unusable sectors becomes excessive, the library should be reorganized. In reorganizing entries, the Library Maintenance program deletes temporary entries and shifts entries so that gaps do not appear between them. This makes more sectors available for use.

Source Library

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, all duplicate characters and blanks are removed from entries. Later, if the entries are printed or punched, the duplicate characters and blanks are re-inserted.

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 first character of a directory name must be an alphabetic character. Maximum length is six characters. 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 four functions of the Library Maintenance programs are described separately. Every description contains the following:

1. List of specific uses.

2. Control statement summary indicating the form of 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.

Following the function descriptions are:

1. OCL considerations

2. Examples

ALLOCATE FUNCTION

ALLOCATE USES

- Create (reserve space for) libraries.
- Change the sizes of libraries.
- Delete libraries.
- Reorganize libraries.

ALLOCATE CONTROL STATEMENT SUMMARY

// ALLOCATE TO-code,SOURCE- $\left\{\begin{matrix}\text{number}\\ \text{R}\end{matrix}\right\}$,OBJECT- $\left\{\begin{matrix}\text{number}\\ \text{R}\end{matrix}\right\}$,SYSTEM- $\left\{\begin{matrix}\text{NO}\\ \text{YES}\end{matrix}\right\}$,DIRSIZE-number,WORK-code

	Use(1)	*Parameter Needed*(2)
Source Library	Create	TO-code,SOURCE-number,WORK-code(3)
	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
	Delete	TO-code,OBJECT-0
	Reorganize	TO-code,OBJECT-R,WORK-code

(1) 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).

(2) 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.

(3) The WORK parameter is needed only if the disk contains an object library that you are not deleting.

Library Maintenance Allocate Restrictions

This program has restrictions and operating conditions that the user must be aware of when maintaining libraries.

Limit of Four Allocations

The system control program allows no more than four allocations of disk space per job. Each ALLOCATE statement that requires additional space (create, increase size) counts as one allocation. The WORK parameter also counts as one allocation (see *WORK Parameter*). For example, creating a source library on a disk that already contains an object library would require two allocations (a work area is needed), but deleting an object library would require no allocations.

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 PARAMETER SUMMARY

TO-code	Location of disk you are using. Possible codes are R1, F1, R2, and F2.
SOURCE-number (no source library)	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	Assign one track to object library directory. Object library directory will not be large enough to contain system program entries.
SYSTEM-YES	Assign three tracks to object library directory. Object library directory will be large enough to contain system program entries.
WORK-code	Location of disk containing space the program can use as a work area. Possible codes are R1, F1, R2, or F2.

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 librarian or system is loaded.

Codes for the possible locations 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

SOURCE and OBJECT Parameters

These parameters identify library uses:

Parameter	*Use*
SOURCE-number OBJECT-number (number is not zero)	• If the disk contains no library, parameter means create a library. Number is the number of tracks you want to assign to the library. • If the disk contains a library, 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 to 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 *Scheduler Work Area Size.*

The following charts show the results of coding the SYSTEM parameter for different allocate users.

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 coded.

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 disk allocation is used. This is included in the limit of four allocations per job (see *Limit of Four Allocations).*

Size of the Work Area

The work area must be large enough to hold the entire 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 (disk contains an object library).	Source library and object library.
Reorganize source library (disk doesn't contain an object library).	Source library.
Change object library size.	Object library.
Reorganize object library.	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)

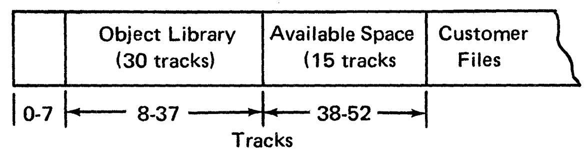
Source Library Size

- 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 (Figure 44). To do this, it 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.

Disk Space Before Creating Source Library



Disk Space After Creating Source Library

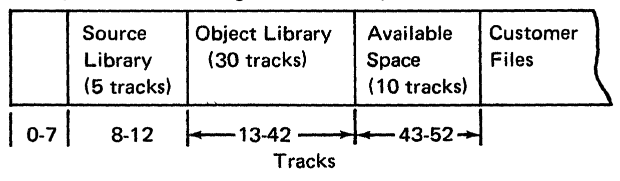

Source Library (5 tracks)
Object Library (30 tracks)
Available Space (10 tracks)
Customer Files
0-7
8-12
13-42
43-52
Tracks

Figure 44. Moving Object Library to Insert Source 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.

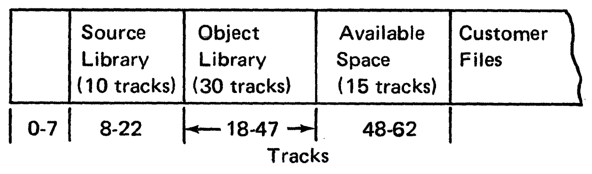
Changing the Size of a Source Library

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 (Figure 45).
- 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
8-22
18-47
48-62
Tracks

Disk After Five Tracks Are Added to Source Library

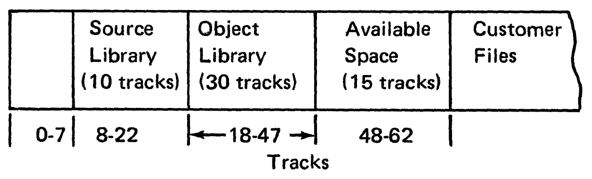

Source Library (10 tracks)
Object Library (30 tracks)
Available Space (15 tracks)
Customer Files
0-7
8-22
18-47
48-62
Tracks

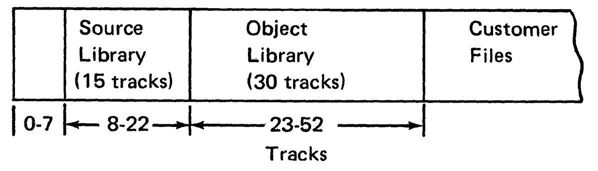
Figure 45. Increasing Source Library Size

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. (Figure 46)

- 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

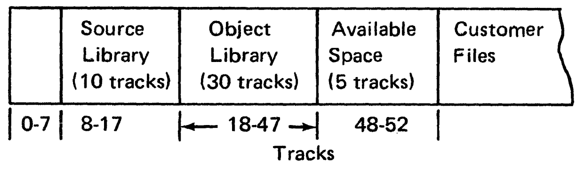
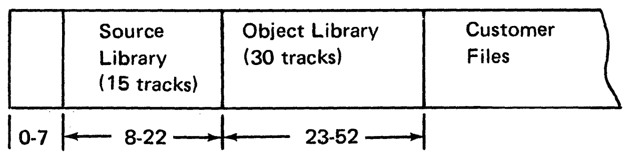

Source Library (10 tracks)
Object Library (30 tracks)
Available Space (5 tracks)
Customer Files
0-7
8-17
18-47
48-52
Tracks

Figure 46. Decreasing Source Library Size

Deleting a Source Library (SOURCE-O)

The program makes the disk area occupied by the source library available for other use (disk files). (Figure 47)

Disk Before Source Library Deleted



Disk After Source Library Deleted

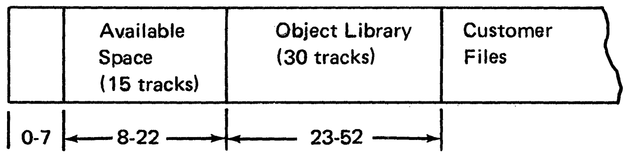


Figure 47. Deleting Source Library

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 eara 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: 30 tracks, including the directory tracks, if the object library is to contain a minimum system; otherwise, the minimum is three tracks including the directory tracks. A minimum system is made up of those system programs necessary to load and run programs. It does not include system programs necessary to generate and maintain a system.

- 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 DPF (Dual Programming Feature) and/or the inquiry feature is on the system. For non-DPF systems, two tracks are needed; for DPF systems, four tracks are needed. The inquiry and checkpoint/restart features require 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*
12K	4
16K	5
24K	6
32K	7
48K	10
64K	13

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 Library (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.***) To do this, it needs a work area. (See *WORK parameter.)*

Deleting an Object Library (OBJECT-O)

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. A work area is needed. (See *WORK parameter.)*

COPY FUNCTION

COPY USES	
Reader-to-Disk	• Add or replace a library entry. The reader is the system input device, which can be either the keyboard or a card reader.
Disk-to-Disk	• 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. • Copy minimum system.
Disk-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 system directory. • Print system directory only.
Disk-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.
Disk-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.

COPY CONTROL STATEMENT SUMMARY: READER-TO-DISK

Add or Replace a Library Entry

```
                                 S                                   T
// COPY FROM-READER,LIBRARY-   { P } ,NAME-name,TO-code,RETAIN-    { P }
                                 O                                   R
                                 R
```

Library Entry

// CEND ◀ Must always follow the source or object statement being placed into the source or object libraries.

COPY CONTROL STATEMENT SUMMARY: DISK-TO-DISK

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.ALL
                              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 either 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, NEWNAME cannot be DIR, ALL, or SYSTEM.

COPY CONTROL STATEMENT SUMMARY: DISK-TO-PRINTER-AND/OR-CARD

Print and/or Punch One Library Entry (or Entries with the Same Name from All Libraries)

$$\text{// COPY FROM-code,LIBRARY-}\left\{\begin{array}{l}\text{S}\\\text{P}\\\text{O}\\\text{R}\\\text{ALL}\end{array}\right\}\text{,NAME-name,TO,}\left\{\begin{array}{l}\text{PUNCH}\\\text{PRINT}\\\text{PRTPCH}\end{array}\right\}$$

Print and/or Punch Temporary and Permanent Library Entries that Have Names Beginning with Certain Characters

$$\text{// COPY FROM-code,LIBRARY-}\left\{\begin{array}{l}\text{S}\\\text{P}\\\text{O}\\\text{R}\\\text{ALL}\end{array}\right\}\text{,NAME-characters,ALL,TO-}\left\{\begin{array}{l}\text{PUNCH}\\\text{PRINT}\\\text{PRTPCH}\end{array}\right\}$$

Print and/or Punch All Temporary and Permanent Library Entries of a Certain Type

$$\text{// COPY FROM-code,LIBRARY-}\left\{\begin{array}{l}\text{S}\\\text{P}\\\text{O}\\\text{R}\end{array}\right\}\text{,NAME-ALL,TO-}\left\{\begin{array}{l}\text{PUNCH}\\\text{PRINT}\\\text{PRTPCH}\end{array}\right\}$$

Print Directory Entries for Library Entries of a Certain Type

$$\text{// COPY FROM-code,LIBRARY-}\left\{\begin{array}{l}\text{S}\\\text{P}\\\text{O}\\\text{R}\end{array}\right\}\text{,NAME-DIR,TO-PRINT}$$

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

$$\text{// COPY FROM-code,LIBRARY-}\left\{\begin{array}{l}\text{S}\\\text{P}\\\text{O}\\\text{R}\\\text{ALL}\end{array}\right\}\text{,NAME-DIR,TO-PRINT,OMIT-}\left\{\begin{array}{l}\text{name}\\\text{CHARACTERS.ALL}\end{array}\right\}$$

COPY PARAMETERS

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:

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-$\left\{\begin{matrix}S\\P\\O\\R\end{matrix}\right\}$

Type of library entries involved in copy use. Possible codes are:

Code	*Meaning*
S	Source statements (source library)
P	OCL procedures (source library)
O	Object programs (object library)
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:

Information	*Meaning*
name	Name of the library entry involved.
characters.ALL	Only those entries beginning with the indicated characters. The names 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.)
ALL	All entries. (The type indicated in LIBRARY parameter.)

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: *Code* — *Meaning* T — Temporary P or R — Permanent *Replacing Existing Library Entry.* RETAIN gives designation of the TO entry and tells program whether to halt before replacing entry: *Code* — *Meaning* T — Temporary designation. Halt before replacing entry. P — Permanent designation. Do not halt before replacing entry. R — Permanent designation. Do not halt before replacing entry. *Printing or Punching Entries.* The RETAIN parameter is ignored.
TO-code	Location of disk that is to contain the copies of the entries: *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
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.

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 Figures 48-50)

- 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 Figures 48-50)

- 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 a 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.

Using the Copy Function

Reader-to-Disk

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 primary hopper of the MFCU. The operator can, however, change the system input device by using the OCL READER statement.

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.

Disk-to-Disk

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), both the source and object libraries are reorganized and temporary entries become permanent entries in the new 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, the disk you specify in the TO parameter must not contain any entries.
- 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.

Disk-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 *Limit of Four Allocations.*)

Printed or Punched Library Entries

- Blanks and duplicate characters are reinserted into source statements and procedures to make them redatable.

- Object programs and routines are printed and punched as they exist in the library.

Printout of Directory Entries

- Source library directory (Figure 48)

- Object library directory (Figure 49)

- System directory (Figure 50)

PRINTOUT

```
SOURCE DIRECTORY FROM XX VOL. ID XXXXXX MM/DD/YY

                        ADDRESS
TYPE  NAME     FIRST@   LAST@    ATTRI  #SECTORS
X     XXXXXX   XXX-XX   XXX-XX   X        XXX
```

Explanation:

Heading	*Meaning*

TYPE	S = source statements
	P = procedure

NAME	Name of library entry (up to six characters)
ADDRESS (FIRST and LAST)	Addresses of first and last sectors that contain the library entry. Addresses are expressed by track and sector numbers. EXAMPLE: 008-03 means track 8, sector 3.

ATTRI	T = temporary
	P = permanent

#SECTORS	Total number of sectors used for the library entry.

Figure 48. Source Library Directory Printout

PRINTOUT

```
OBJECT DIRECTORY FROM XX VOL. ID XXXXX MM/DD/YY

              DSK     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     XXXXX
```

EXPLANATION:

Heading	*Meaning*
TYPE	A { P=permanent, T=temporary } Attribute L { O=object, R=routine } Library
NAME	Name of library entry (up to six characters)
DSK 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; RLD is information used in loading the program for execution. For routines, this number is the category of the routine. This number is used by the Overlay Linkage Editor for determining overlays.

Figure 49. Object Library Directory Printout (Part 1 of 2)

PRINTOUT (Continued)

Heading	*Meaning*	
LINK ADDR	Object programs only. Assigned core address of this library entry.	
RLD DISP	Object programs only. It indicates the 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 POINT	Object programs only. Main storage address where program execution begins before relocations.	
CORE SEC	Core size, given in sectors, required to run the program.	
ATTR	Byte 1:	
	Bit 0=1	Permanent Entry
	0	Temporary Entry
	Bit 1=1	Inquiry. This program requires that the Inquiry key be pressed to start processing.
	Bit 2=1	Inquiry Invoking. This program runs in program level 1 and can be rolled out to allow an Inquiry program to run.
	Bit 3=1	Dedicated. In a DPF system, this program must run with the other program level inactive.
	Bit 4=1	Source Required. This program requires the allocation of the $WORK and $SOURCE files. $SOURCE must be filled either from SYSIN or a source library.
	Bit 5=1	Deferred Mount. This program accepts mounting of packs during its execution.
	Bit 6=1	PTF Applied. A program temporary fix (PTF) has been applied to this program.
	Bit 7=1	RPG Object/Overlay Program
	Byte 2:	
	Bit 0=1	SYSIN Dedication. The SYSIN device must be dedicated to this program. The device is released at end of job.
	Bit 1=1	Checkpoint/Restart Program
	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.
	Bit 3	Reserved
	Bit 4=1	Macro Processor Allowed. This program can be preceded by the macro processor. If the source required attribute is present and the UPSI condition of X'80' is satisfied, the $SOURCE file is opened as input instead of output.
	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.
	Bit 6	Reserved
	Bit 7	Reserved
LEVEL	Release level of system programs. For user programs this can be assigned by the Overlay Linkage Editor.	
TOT SEC	Total number of disk sectors occupied by the library entry.	

Figure 49. Object Library Directory Printout (Part 2 of 2)

SYSTEM DIRECTORY

SOURCE LIBRARY SECTION

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 SECTION

Object Directory Location	TTT-SS
End of Directory	TTT-SS
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 Available 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	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

TTT-SS-DDD means track, sector, and displacement. Displacement is the number of characters from the beginning of the sector. XXX means number of sectors. YYY means number of tracks.

Figure 50. System Directory Printout

DELETE FUNCTION

DELETE USES

- Delete a temporary or permanent entry from a library (or entries with the same name from all libraries).
- Delete temporary or permanent
- Delete a temporary or permanent entry from a library (or entries with the same name from all libraries).
- Delete temporary or permanent library entries that have names beginning with certain characters.
- Delete all temporary or permanent library entries of a certain type.

DELETE RESTRICTIONS

- System modules cannot be deleted from the active system pack (the pack the system was loaded from at IPL time).
- The object library cannot be deleted from the pack that the system or the Library Maintenance program was loaded from.
- 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.

DELETE 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- <     > ,NAME-ALL,RETAIN- <     >
                              | O |                  ( P )
                              ( R )
```

DELETE PARAMETERS

FROM-$\left\{\begin{matrix}\text{R1}\\\text{F1}\\\text{R2}\\\text{F2}\end{matrix}\right\}$

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-$\left\{\begin{matrix}\text{S}\\\text{P}\\\text{O}\\\text{R}\\\text{ALL}\end{matrix}\right\}$

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-$\left\{\begin{matrix}\text{name}\\\text{characters.ALL}\\\text{ALL}\end{matrix}\right\}$

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-$\left\{\begin{matrix}\text{T}\\\text{P}\end{matrix}\right\}$

Designation of entries being deleted:

Code	*Meaning*
T	Temporary
P	Permanent

RENAME FUNCTION

RENAME USE

- Change the name of a library entry.
- Change the name of library entries that have names beginning with certain characters.

RENAME CONTROL STATEMENT SUMMARY

// RENAME FROM-code,LIBRARY-$\left\{\begin{matrix}\text{S}\\ \text{P}\\ \text{O}\\ \text{R}\end{matrix}\right\}$,NAME-name,NEWNAME-name

// RENAME FROM-code,LIBRARY-$\left\{\begin{matrix}\text{S}\\ \text{P}\\ \text{O}\\ \text{R}\end{matrix}\right\}$,NAME-characters.ALL,NEWNAME-characters

RENAME PARAMETERS		
FROM-code	Location of disk that contains the entry you are renaming. 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}	Type of library entry you are renaming. Possible codes are:	
	Code	*Meaning*
	S	Source statements (source library)
	P	Procedures (source library)
	O	Object programs (object library)
	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:	
	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.
	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

The following OCL statements are needed to load the Library Maintenance utility program.

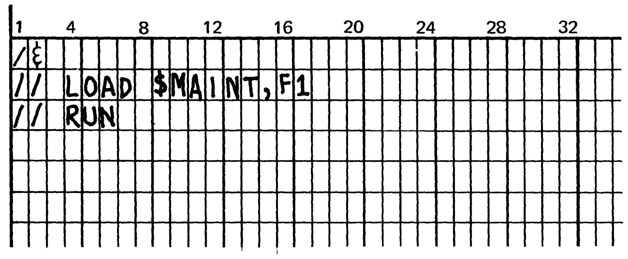
```
// LOAD $MAINT,code

// RUN
```

The code you supply depends on the location of the disk containing the Library Maintenance program. 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

EXAMPLES

Figures 51-62 illustrate the functions of the Library Maintenance utility program. Figure 51 is an example of the OCL needed to load the utility program. The other figures are examples of the control statement necessary to carry out the specified function.


1
4
8
12
16
20
24
28
32

```
/&
// LOAD $MAINT,F1
// RUN
```

Explanation:

- Library Maintenance program is loaded from the fixed disk on drive one

Figure 51. OCL Load Sequence for Library Maintenance

```
1  4   8   12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
// ALLOCATE TO-R1,SOURCE-12,OBJECT-45,SYSTEM-YES
// END
```

Explanation:

- Libraries are being created on the removable disk on drive one (TO-R1 in ALLOCATE statement).
- Source library space is 12 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.
- Directory will be three tracks.

Figure 52. Allocate Example. Creating Both Source and Object Libraries on a Disk

```
1  4   8   12  16  20  24  28  32  36
// ALLOCATE TO-R1,SOURCE-15,WORK-F1
// END
```

Explanation:

- Source library is located on the removable disk on drive one (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 one (WORK-F1).

Figure 53. Allocate Example: Changing the Size of a Source Library

```
1  4   8   12  16  20  24  28  32
// ALLOCATE TO-R1,OBJECT-Ø
// END
```

Explanation:

- Object library is located on the removable disk on drive one (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, the program also deletes the work area.

Figure 54. Allocate Example: Deleting the Object Library from a Disk

```
1  4   8   12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
// COPY FROM-F1,LIBRARY-O,NAME-SYSTEM,TO-R1
// END
```

Explanation:

- System programs are in the object library on the fixed disk on drive one (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 one (TO-R1).

Figure 55. Copy Example: Copying Minimum System from One Disk to Another

```
1  4   8   12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
// COPY FROM-R1,LIBRARY-ALL,NAME-DIR,TO-PRINT
// END
```

Explanation:

- All library directories and the system directory on the removable disk on drive one 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 indicate that the program is to be printing directories.

Figure 56. Copy Example: Printing Library Directories

```
1  4   8   12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
// COPY FROM-R1,LIBRARY-O,NAME-ACCT,TO-F1,RETAIN-R
// END
```

Explanation:

- 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 one.

- TO-F1 tells the program to copy the object program to the fixed disk on drive one. 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.

Figure 57. Copy Example: Copying Object Program to F1

```
1  4   8   12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
// DELETE FROM-R1,LIBRARY-S,NAME-PAYROL
// END
```

Explanation:

- The program deletes a set of source statements (LIBRARY-S in DELETE statement) named PAYROL (NAME-PAYROL) from the removable disk on drive one (FROM-R1) that has a temporary attribute.

Figure 58. Delete Example: Deleting an Entry from a Library

```
1  4   8   12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
// DELETE FROM-R1,LIBRARY-ALL,NAME-INV.ALL
// END
```

Explanation:

- The entries being deleted are on the removable disk on drive one (FROM-R1 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), with temporary attributes.

Figure 59. Delete Example: Deleting All Entries with Names that Begin with Certain Characters

```
1  4   8   12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
// DELETE FROM-R1,LIBRARY-P,NAME-ALL,RETAIN-T
// END
```

Explanation:

- The entries being deleted are on the removable disk on drive one (FROM-R1 in DELETE statement).
- All temporary procedures are being deleted from the source library (LIBRARY-P,NAME-ALL).

Figure 60. Delete Example: Deleting All Library Entries of One Type

```
1  4   8   12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
// RENAME FROM-R1,LIBRARY-S,NAME-ACCT,NEWNAME-ACCT1
// END
```

Explanation:

- The removable disk on drive one 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).

Figure 61. Rename Example: Renaming a Set of Source Statements in a Source Library

```
1.  // LOAD $MAINT,F1
    // RUN
2.  // ALLOCATE TO-R1,OBJECT-Ø,SOURCE-Ø
3.  // ALLOCATE TO-R1,OBJECT-40,SOURCE-12,SYSTEM-YES,DIRSIZE-7
4.  // COPY FROM-F1,TO-R1,LIBRARY-ALL,NAME-ALL
    // END

              Reload System (IPL) from R1

5.  // LOAD $MAINT,R1
    // RUN
6.  // ALLOCATE TO-F1,OBJECT-Ø,SOURCE-Ø
7.  // ALLOCATE TO-F1,OBJECT-40,SOURCE-12,SYSTEM-YES,DIRSIZE-7
8.  // COPY FROM-R1,TO-F1,LIBRARY-ALL,NAME-ALL
    // END
              Reload System (IPL) from F1
```

Explanation:

1. The system and $MAINT are both loaded from F1.
2. The libraries on R1 are deallocated (if present).
3. New library space is allocated on R1.
4. The libraries are copied from F1 to R1. The libraries are reorganized as they are copied. Temporary entries become permanent when copied (see *Disk-to-Disk Considerations, Output*).
5. The system and $MAINT are now loaded from R1.
6. The libraries on F1 are deallocated.
7. New library space is allocated on F1.
8. The libraries are copied back to F1. The pack on R1 could be used as a back-up pack. It contains the same libraries as F1.

Figure 61. Reorganizing the System Pack

All IBM 2316 disk packs initialized on System/3 5445 Disk Drive by $INIT have a System/360-System/370 formatted volume table of contents (VTOC). The System/360-System/370 VTOC is not used by System/3. When it is necessary to exchange data between System/3 and System/360-System/370 on a 2316 disk pack, the IBM System/3 5445 Data Interchange Utility can be used (see Appendix C for an alternate method). The utility must be run going to and returning from System/360-System/370.

When the utility program is run against a 2316 disk pack, the contents of the System/3 VTOC are mapped to the System/360-System/370 VTOC. If data is to be returned to the System/3 via the utility without reinitialization, then restrictions on the use of the pack on System/360-System/370 must be observed. Any deviations from these restrictions can result in the format of the pack being altered beyond the capacility of the utility to return the pack to normal System/3 format. This can result in errors in the utility run returning the pack or unrecoverable errors on the pack while processing it on System/3.

Following is a list of the methods of processing data files on the interchange pack by OS or DOS:

Functions (sequential processing only)	*Disposition*	*Type*	*Open*
Reading with OS using BSAM or QSAM	OLD	FBS	INPUT
Reading with DOS using SAM-GET	––	––	INPUT
Update in place with OS using BSAM or QSAM	OLD	FBS	UPDATE
Update in place with DOS using SAM-GET/PUT	––	UPDATE	INPUT

CAUTION:

Only the above disposition and open types may be used.

The update-in-place function can be used on a data set written on System/3 filled with dummy records. Since duplicate file names are not allowed on System/360-System/370, the System/3 file names will be qualified with the file date. An example would be PAYROLL. D711026. PAYROLL would be the file name on System/3 and the file was created on October 26, 1971.

Files to be processed by QSAM must have a logical record length that is an even submultiple of 256.

No files may be allocated or deleted on System/360 or System/370.

Any System/3 P or T file on the pack is mapped into the System/360-System/370 VTOC. Multivolume files are not supported and their interchange results in a System/360-System/370 entry that appears like a single volume file. Split cylinder files will have a System/360-System/370 format one but it is not usable due to basic differences in split file philosophy between the systems. If the System/3 file type is either consecutive or indexed but not split, then a System/360-System/370 end-of-file mark is written in the file area at the end of data. When the utility is run to return the pack to System/3, the end of file marks are removed and the System/360-System/370 VTOC entries are deleted.

The utility must always be run last when going to System/360-System/370 and first when returning to System/3. Any deviation from this procedure can result in loss of data on the pack.

The attributes of all System/360-System/370 VTOC entries assigned by the utility are as follows:

Name of file	–	name. DYYMMDD
Creation date	–	00000
Expiration date	–	99365 (date protected)
Volume sequence number	–	0001
Record/block format	–	FIXED BLOCK STANDARD (FBS)
Organization	–	sequential (regardless of S/3 type)
System Code	–	"IBM DSM/3"
Block length	–	256 bytes
Logical record length	–	same as S/3 length
Extent type	–	single

CONTROL STATEMENT SUMMARY

System/3 to System/360-System/370 Conversion

```
// NEWVTOC UNIT-{D1},PACK-name
               {D2}
// END
```

System/360-System/370 to System/3 Conversion

```
// UPDATE UNIT-{D1},PACK-name
              {D2}
// END
```

PARAMETER SUMMARY

PACK-name	Name of the disk.
UNIT-code	Location of the disk. Possible codes are D1 and D2.

PARAMETER DESCRIPTIONS

PACK Parameter

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

The 5445 Data Interchange 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 pack being transferred. Codes for the possible locations are as follows:

Code	*Meaning*
D1	Removable disk on 5445 drive one
D2	Removable disk on 5445 drive two

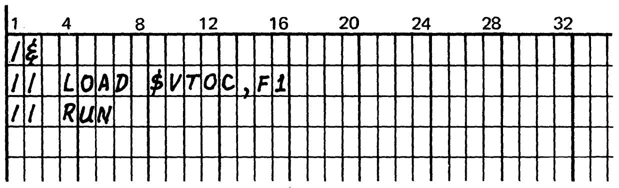
OCL CONSIDERATIONS

The following OCL statements are needed to load the 5445 Data Interchange Utility program:

```
// LOAD $VTOC, code
// RUN
```

The code you supply depends on the location of the disk containing the utility program. 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



```
1  4   8   12  16  20  24  28  32
/&
// LOAD $VTOC,F1
// RUN
```

Explanation:

- 5445 Data Interchange Utility is loaded from the fixed disk on drive one.

APPENDIX A. IBM SYSTEM/3 STANDARD CHARACTER SET

Character	Hexadecimal Equivalent
Blank	40
¢	4A
.	4B
<	4C
(	4D
+	4E
|	4F
&	50
!	5A
$	5B
*	5C
)	5D
;	5E
¬	5F
- (minus)	60
/	61
,	6B
%	6C
— (underscore)	6D
>	6E
?	6F
:	7A
#	7B
@	7C
' (apostrophe)	7D
=	7E
"	7F
A	C1
B	C2
C	C3
D	C4
E	C5
F	C6
G	C7
H	C8
I	C9
}	D0
J	D1
K	D2
L	D3
M	D4
N	D5
O	D6
P	D7
Q	D8
R	D9
S	E2
T	E3
U	E4
V	E5
W	E6
X	E7
Y	E8
Z	E9
0	F0
1	F1
2	F2
3	F3
4	F4
5	F5
6	F6
7	F7
8	F8
9	F9

APPENDIX B. CONVERSION

RECORDS TO TRACKS CONVERSION

Determining the Number of Sequential or Direct File Tracks

The following two steps should be followed to determine the number of tracks in a sequential or direct file. (Round results to the nearest whole number.)

1. number of records x record length = number of characters

2. $\dfrac{\text{number of characters}}{\text{number of characters per track [1]}}$ = number of tracks

Determining the Number of Indexed File Tracks

The following two steps should be used to determine the number of data tracks in an indexed file:

1. number of records x record length = number of characters

2. $\dfrac{\text{number of characters in a sector [2]}}{\text{number of characters per track [1]}}$ = number of data tracks

The following four steps should then be followed to determine the number of index tracks in an indexed file:

1. key field length + (3 for 5444 or 4 for 5445) = index entry length

2. $\dfrac{\text{number of characters in a sector [2]}}{\text{index entry length}}$ = number of entries per sector

3. $\dfrac{\text{number of records}}{\text{number of entries per sector}}$ = number of sectors

4. $\dfrac{\text{number of sectors}}{\text{number of sectors per track [3]}}$ = number of index tracks

The total number of tracks in an indexed file can then be determined by adding the number of data tracks to the number of index tracks.

---

[1] 6144 for the 5444
5120 for the 5445

[2] 256 (For the 5445, a sector is referred to as a fixed record.)

[3] 24 for the 5444
20 for the 5445

CYLINDER/TRACK TO TRACK NUMBER CONVERSION

To convert cylinder/track to track number, multiply cylinder number by the number of tracks on each cylinder and add track.

EXAMPLE: 5/3 = cylinder/track
$5 \times 20^{*} + 3 = 103$
103 = track number

TRACK NUMBER TO CYLINDER/TRACK CONVERSION

To convert track number to cylinder/track, divide track number by the number of tracks on a cylinder. The quotient is the cylinder and the remainder is track.

EXAMPLE: 103 = track number
$103 \div 20^{*} = 5$ (remainder 3)
5/3 is the cylinder/track

* 20 = number of tracks on a cylinder

APPENDIX C. SYSTEM/360-SYSTEM/370 DISK FILE COMPATIBILITY

This appendix is intended for the user who intends to exchange data between System/3 and System/360-System/370 without using the IBM System/3 5445 Data Interchange Utility Program. The access method limitations listed in the utility program section of this manual should be followed.

Disk files created on the 5445 can be read and updated using System/360-System/370. Disk files can also be created using System/360-System/370 and subsequently read or updated with a System/3 Model 10 Disk System.

The volume label and volume table of contents (VTOC) identify the information contained on the disk pack. The volume label identifies the volume and points to the System/360-System/370 VTOC. The System/360-System/370 VTOC contains one label record which describes the complete pack as one System/360-System/370 file. The System/3 VTOC resides in a fixed location within this System/360-System/370 file and can be examined by the System/360-System/370 program.

See *IBM System/3 Disk Systems System Control Program Logic Manual,* SY21-0502, for a description of the System/3 VTOC and volume label.

System/3 to System/360-System/370

The System/3 Disk Initialization Program writes a volume label in the System/360-System/370 format on every disk pack. The System/3 disk format consists of 256-byte physical records. This record length may be altered for System/360-System/370 VTOC records.

Any of the access methods previously listed may be used by System/3 when creating a file to be used by System/360-System/370. The logical records in a particular System/3 file can be accessed by System/360-System/370 by means of a user program using the Sequential Access Method if the user program:

- Locates the file label in the System/3 VTOC for the desired file.
- Uses the start of data information and record length information from the System/3 VTOC to perform the accessing and logical deblocking.
- Uses the end-of-file information from the System/3 VTOC.

System/360-System/370 to System/3

Volumes created on System/360-System/370 can be processed on System/3 if System/360-System/370 provides a System/3 VTOC entry and writes 256-byte physical records. A System/3 user program or utility can then read and unblock the file according to the information in the System/3 VTOC.

CAUTION

If the System/3 VTOC provided by System/360-System/370 is not exactly the same as the System/3 format, unexpected results (destroyed data files or unrelated halts) may occur.

Index

READER'S COMMENT FORM

IBM System/3 GC21-7512-4
Model 10 Disk System
Operation Control
Language and Disk Utilities
Reference Manual

YOUR COMMENTS, PLEASE...

Your comments concerning this publication will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Note: Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.