CONTROL DATA
CORPORATION

CONTROL DATA® 6000 COMPUTER SYSTEMS

SCOPE REFERENCE MANUAL
6000 VERSION 3.3

REVISION RECORD	
REVISION	DESCRIPTION
A	This document is printed in conjunction with the release of version 3.3 of the
(9-4-70)	SCOPE Operating System. Features added to the Reference Manual since the
	previous version are indicated by a bar in the outside margin or by a dot next
	to the page number if an entire page is affected.
B	Changes and additions are indicated by a bar in the outside margin or by a dot
(6-30-71)	next to the page number if the entire page is affected. Additions include
	Recover, Permanent File macros and the LABEL macro; and print files
	conventions. Pages affected include front matter; 2-1 thru 2-40; 3-1 thru 3-33;
	4-1 thru 4-32; 5-1 thru 5-23; 6-1 thru 6-16; 7-7, 7-8; 8-1 thru 8-16; 10-1 thru
	10-28; A-1 thru A-3; B-1 thru B-21; C-5, C-6; D-3, D-4, D-15 thru D-22,
	D-27 thru D-36, D-39 thru D-47; F-1 thru F-2; Index-1 thru Index-13.
	Comment Sheet.
C	Changes and additions are indicated by a bar in the outside margin or by a dot
(7-9-71)	next to the page number if the entire page is affected. Additions include diag-
	nostic messages for Tape Reliability, a new feature. Minor corrections have
	been made to the text. Pages affected include front matter, 6-15; 7-1, 7-7;
	8-9; 10-4, 10-9, 10-25; A-1 thru A-4; D-1, D-12, D-19, D-20, D-26 thru D-29,
	D-43; Comment Sheet.
Publication No. 60305200	

Additional copies of this manual may be obtained from the nearest Control Data Corporation sales office.

Address comments concerning this manual to:

CONTROL DATA CORPORATION
Software Documentation
215 MOFFETT PARK DRIVE
SUNNYVALE, CALIFORNIA 94086

or use Comment Sheet in the back of this manual

New features, as well as changes, deletions, and additions to information in this manual are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

REVISION RECORD (Cont'd)

REVISION	DESCRIPTION
D	Changes include REQ parameters, CPC operation, OPEN/WRITE,
(9-10-71)	OPEN/WRITE/NR; clarifications of RECOVR, FET macros, DISPOSE,
	RETURN, MEMORY, CLOCK and COPY items. Page affected include front
	matter; 2-21, 2-22, 2-39; 3-9, 3-19, 3-20; 4-3, 4-6, 4-7 thru 4-9, 4-14
	4-30, 4-31; 8-2; D-25, D-28, D-39, D-40; comment sheet
Publication No. 60305200	

PREFACE

This manual describes the SCOPE 3.3 Operating System for the CONTROL DATA® 6000 Series computers. It has been written with the Applications Programmer in mind, yet it contains information of value to the Systems Programmer as well.

Other documents of interest to programmers using SCOPE 3.3 are:

SCOPE 3.3 OPERATOR'S GUIDE	Pub. No. 60306400
SCOPE 3.3 PRODUCT SET INSTALLATION HANDBOOK	Pub. No. 60305100
SCOPE 3.3 SCOPE INDEXED SEQUENTIAL	Pub. No. 60305400
SCOPE 3.3 USER'S GUIDE	Pub. No. 60252700

Suggestions and criticisms of this manual, its form and content, may be submitted on the Comment Sheet in the back of the manual or sent directly to Software Documentation, Control Data Corporation, 215 Moffett Park Drive, Sunnyvale, California, 94086.

CONTENTS

INTRODUCTION 1

SCOPE (Supervisory Control Of Program Execution) is a group of programs and subprograms that comprise the operating system for the CONTROL DATA 6000 series computers. Input, compilation, assembly, loading, execution, and output of all programs submitted to the computer, as well as the allocation of resources for these programs are monitored and controlled by SCOPE. This file-oriented operating system resides primarily on random access mass storage devices. It uses the versatility and efficiency of the computer hardware to direct the multiprogramming of up to seven user programs (jobs). Jobs written in the COMPASS assembly language and a variety of compiler languages, or calling for many different utility systems, can be processed under SCOPE. The product set contains: COMPASS, FORTRAN, FORTRAN Extended, COBOL, SORT/MERGE, PERT/TIME, EXPORT/IMPORT, 1700 MSOS Import H/S, SIMSCRIPT, APT, ALGOL, SIMULA, INTERCOM, and BASIC.

6000 SERIES HARDWARE OVERVIEW

The minimum hardware requirements of a 6000 series computer system consist of: the computer (including 32,768 words of central memory storage); any combination of disks, disk packs, or drums to provide 24 million characters of mass storage; a card reader, card punch, and printer (with controllers); and two 7-track magnetic tape units. Larger systems can be obtained by including optional equipment such as: additional central memory; extended core storage (ECS), additional card readers, punches, printers, and tape units. Graphic plotters and microfilm recorders are also available.

The 6000 series computer is composed of central memory, one or two highspeed central processors, seven to ten peripheral processors, and a display console. Central memory holds all active jobs. Up to seven jobs can be active simultaneously, sharing the central processor registers at scheduled intervals. The central processor thus serves as the computer's calculator. Each job in central memory gains access to the central processor alternately with the other jobs until execution is complete.

The peripheral processors are used to input jobs to the computer, load jobs from mass storage into central memory, transfer input or output between mass storage or peripheral devices and active jobs, and dispose of output from completed jobs. They relieve the central processor of all input/output tasks, enabling it to devote full time to program execution. Each peripheral processor contains its own memory and is actually an individual computer that operates independently of the other processors.

The display console includes two CRT screens which display information about the system and the jobs in process. The console also includes a keyboard through which the operator can enter requests to modify stored programs and display information about jobs in or awaiting execution.

Further information about the computer hardware can be found in the 6000 Series Computer Systems Reference Manual. Details about the movement of information between the central processor registers and central memory are presented in this manual and also in the 6000 Series COMPASS Reference Manual.

MAIN CONCEPTS OF SCOPE

SYSTEM OVERVIEW

All SCOPE routines and subroutines are recorded in a file on a mass storage disk or drum. Copies of routines used most frequently also reside permanently in memory where they can be executed without delay. Others are called into memory from mass storage only when needed, to ensure that a maximum amount of memory remains free for user jobs.

The portion of SCOPE residing in central memory consists of system tables and pointer words used for communication between user jobs and SCOPE routines, and for linking the peripheral processor memories with each other and with central memory. Also, some frequently used routines that can be called into peripheral processor memory on a transient basis are stored in central memory, because they can be loaded from that area much faster than from mass storage. Central memory areas reserved for SCOPE cannot be overwritten by user jobs.

One peripheral processor (PP0) holds only one routine, called Monitor, that oversees and controls all SCOPE functions. These functions include allocation of hardware and files to a user job, as well as coordination of the activities of all other PP's.

Another peripheral processor (PP1) is devoted exclusively to a routine that drives the system display console and input keyboard. This routine interprets and processes all requests typed by the operator and displays all messages from SCOPE to the operator.

Each remaining pool PP contains a resident routine that services requests from Monitor, loads and executes programs as assigned by Monitor, and provides a communications interface between Monitor and the program loaded. The programs which PPs load and execute include routines for handling input/output activities associated with all phases of job execution, such as: reading a job from a card reader onto mass storage or from mass storage into central memory, sending output to mass storage or printer, or reading and writing information on magnetic tapes.

Each resident routine in a pool PP contains pointer words that refer to a communication area in central memory. The PP resident routine contains a subroutine that uses these pointers to continually examine the communication area for requests from Monitor, which is linked to all PP's and jobs through these and other central memory communication areas.

BASIC FUNCTIONS

When a job enters the computer, SCOPE processes the job, assigns the hardware resources required, and ensures that input, output, and system files needed by the job are made available to it. These three functions are interrelated and mutually dependent; resource and file management take place through the entire course of job processing.

Job processing consists of the following tasks: loading the job into the computer, assigning it system resources (central memory storage, magnetic tapes, etc.), executing the job, terminating the job, and processing its output.

When the user's program and associated data is in card form, the job consists of these cards preceded by a group of control cards. Control cards request such functions as assembly or compilation, loading, and execution. They also request equipment or files required by the job, and the amount of time and central memory storage or ECS it will need. In addition, processing priority with respect to other jobs and special instructions to the computer operator can be specified on these cards.

When the user's programs reside on mass storage or on tape, the job may consist only of control cards that direct the loading and execution of these programs and ensure that the hardware and files needed for their execution are available.

Job processing is initiated by loading the job into the card reader. From this point on, control of the job is assumed by SCOPE, which assigns a PP that transfers the job from the card reader onto mass storage. The job is then in the input queue, awaiting availability of system resources. When the required resources are available and the priority of the job permits, SCOPE designates another PP to read the job into central memory. In accordance with the control card requests, SCOPE assigns any additional files or hardware required to the job, also using PP's for these purposes. These items can include tape or disk files or ECS. The job begins execution, sharing access to the central processor with other jobs in execution. As execution progresses, peripheral processors are assigned to handle any additional input or output files and their associated hardware.

Jobs in central memory awaiting use of the central processor for the first time, or after interruption of execution, are scheduled by priority. When the job using the central processor is interrupted, perhaps to await completion of input/output, the job with the next highest priority is given the central processor. When the interrupted job is again ready for the central processor, it will be scheduled according to its priority.

When a job is completed or when it is terminated because of a fatal error, SCOPE frees the central memory areas, releases the files assigned to the job, and assigns a PP to send any output to mass storage. The job is then in the output queue. When the peripheral devices requested for the output are available, SCOPE writes the output from the output queue onto these devices. Typical output devices are the printer, card punch, tape unit or graphic plotter. If no device is specified, output is assigned automatically to the printer. Following the output from the job itself, the dayfile containing chronological information, diagnostic messages, and accounting data about the job is output.

MULTIPROGRAMMING

Multiprogramming means that more than one job can be in process in central memory at one time. At any given instant, only one job can be using the central processor. However, several can be performing input/output. In fact, a job can have more than one input or output operation in progress simultaneously.

CENTRAL MEMORY USAGE

Each job in progress occupies a contiguous, uninterrupted block of words in central memory. References to all addresses within each block are made in relation to the reference address (RA) which is the first address in the block. The length of the block is the field length (FL) of the job. If a user's job references a location outside its field length, the hardware senses this error; SCOPE terminates the job, thereby protecting all other jobs and system programs in central memory from being accidentally overwritten. For this reason, the user can consider his job as a program running alone in a computer with a central memory the size of his field length.

Two distinct areas of central memory are reserved for portions of SCOPE. These areas cannot be addressed by user jobs during normal execution. The low core area (the beginning addresses of central memory) contains the central memory resident routines of SCOPE, the systems tables, pointer words, communication areas used to link the peripheral processor and central memories, and subroutines used often by both central memory and the peripheral processors. The high core area comprises the last (highest numbered) addresses in central memory. It holds information about files on mass storage and is referenced during data transfer to and from such files. The relationship of low and high core to the remainder of central memory is shown in figure 1-1. As shown, the first address is at the extreme low end of central memory and the last address is at the extreme upper end.

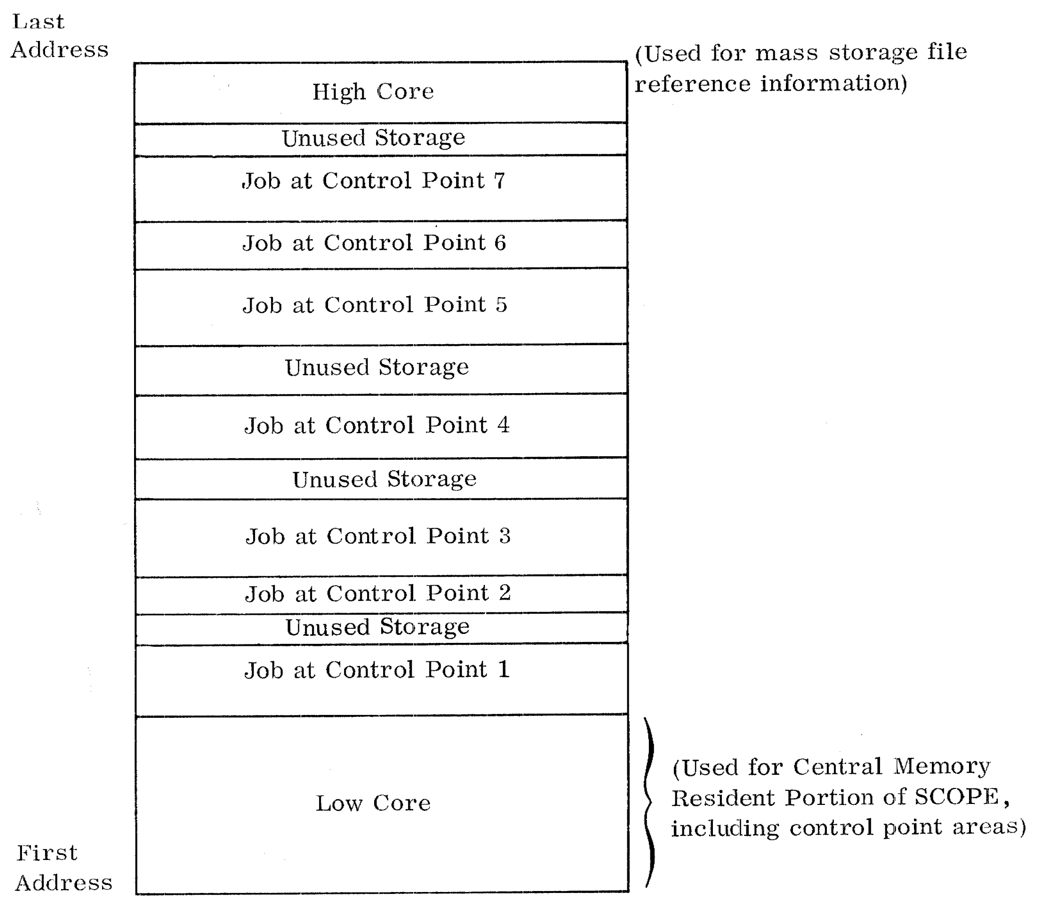


Figure 1-1. Central Memory Allocation

CONTROL POINTS

Every job in central memory is related to a SCOPE control point. The control point is an area of central memory partitioned off by SCOPE for a job. Each control point interrelates the following elements common to a particular job: the central memory field length allotted; other hardware and files used by the job; a block in low core, called the control point area, that contains reference information about the job; and the number associated with the job on the console displays. The control point area contains such information as the job name, processing time accumulated, input/output equipment assigned to it, related control statements, and the job's exchange jump package.

Control points are numbered from 1 to n; seven is the maximum number of jobs that can be run simultaneously. At most installations, n equals 7; however, since this value is an installation option, it may differ at some sites. In addition to the control points available to user jobs, two others numbered 0 and n + 1 are reserved for SCOPE functions. Control point 0 is used to identify all hardware and software resources which are not presently allocated to user jobs or which are used only by SCOPE. Control point n + 1 is used by SCOPE for system utility programs.

When jobs are input from a card reader or tape, or when job output is transferred to the printer or card punch, the SCOPE loader package that handles these functions must reside at one of the control points 1 through n. In addition, a control point is often reserved for jobs being input from remote processing facilities. If one or both of these cases prevail, and x is the number of control points reserved for these functions, then n-x is the number of user jobs that can be running simultaneously.

CENTRAL MEMORY ALLOCATION

The position of central memory storage allocated to each job is related to the control point number to which the job is assigned. At most installations, this assignment is made and maintained in numerical order. Thus, the job at control point 2 will always follow that at control point 1 in memory, and the job at control point 2 will remain behind that at control point 3, as shown in figure 1-1.

Although they remain in the same order, jobs are moved up and down in storage to make room for incoming jobs through a process called dynamic relocation. This process takes place continuously as central memory is required or released. For example, jobs may be running at all control points except control point 2 and a new job is assigned to control point 2. If sufficient contiguous storage is not available for the new job, SCOPE will relocate any other jobs necessary to make sufficient contiguous storage. Each job will be moved as a continuous block, and only its reference address (RA) will be changed accordingly within the appropriate SCOPE reference tables. It may be necessary to relocate the jobs at control points 1 and 3, or only one of them. If the job at control point 3 is relocated, it also may be necessary to move one or more of the jobs following it; but the order of the jobs within central memory remains the same. When a job is moved in storage, Monitor suspends all user program activity at the control point, waits for all PPs assigned to the control point to pause, initiates the exchange jump package to save dynamic information about the job during the move, and starts the SCOPE routine that moves the job. When the move is complete, the RA of the job is modified and job activity is resumed.

JOB ROLLOUT

In certain cases, active jobs can be moved (rolled out) from central memory to mass storage or ECS while maintaining their control points and all information necessary to resume processing. Job roll-out is initiated by the operator under time-shared conditions when a job with a very high priority needs more storage than is available. A job may be rolled out also when it must await the availability of unusual hardware resources, such as many peripheral devices, or a large amount of mass storage. If the job cannot proceed until such equipment becomes available, the operator can withdraw it and process other jobs until the equipment is ready. Although overall turnaround time is prolonged when a job is rolled out, the total processing time of the job is not affected. Further, the turnaround time for other jobs is greatly enhanced.

JOB PRIORITIES

The user can request one of several priority levels for his job or leave the level assignment to SCOPE. Once the job is read into the system, SCOPE cannot change the priority level but can alter the priority of a job with respect to other jobs of the same level. Generally, within a level, the highest priorities are assigned to jobs that have been in the input queue the longest. Among jobs with equal priority, the job with the greatest field length is generally processed first if the resources required for that job are available.

JOB/SCOPE COMMUNICATION

Since no instructions within a job can directly access memory locations outside its field length, an area must be reserved within the job's FL that SCOPE checks periodically to maintain communication with the job. This area is comprised of the first 100(octal) locations in the job's FL, location RA+0 through RA+77(octal). The user program actually begins at location RA+100(octal). RA+0 is reserved for use by the hardware; if an arithmetic error occurs, the type of error and the location of the error are stored in RA+0. Requests from the job to SCOPE are stored in RA+1. These can be requests for a PP to perform input/output, a signal to SCOPE that processing is complete, or a request to terminate the job because of an error. RA+2 through RA+77 holds information about the SCOPE control card that is directing the current processing, and information for the pertinent loader.

EXCHANGE JUMPS

A job gains or relinquishes access to the central processor each time an exchange jump instruction is issued by the SCOPE Monitor routine. This is done when a job has used the central processor for the maximum interval allowed by SCOPE. It also is done in response to any event that affects the relationship of the job to others in the overall SCOPE priority system. Each time an exchange jump is executed, any job using the central processor is interrupted, and the job with the next highest priority achieves use of the central processor.

The exchange jump is made possible through a block of information within the control point area assigned to each job. The control point area, which resides in low core, contains a 16-word exchange package, which comprises the information used directly in exchange jumps: the most recent contents of all central processor registers, the RA and FL, in central memory and in ECS, of the job, and the address of the next instruction to be executed (the program address).

Suppose that job A has the next highest priority for access to the central processor and job B, which is currently active, is to be interrupted. The Monitor program performs the following steps to deactivate job B and activate job A. First, Monitor executes a central exchange jump which replaces job B's active exchange package with the SCOPE idle package and saves job B's exchange package in the 16-word control point area for job B. Then Monitor performs another central exchange jump which replaces the active SCOPE idle package with the exchange package of job A and saves the idle exchange package in the 16-word control point area for job A. When an exchange jump is executed, the issuing of instructions within the active job halts and the instructions already issued are executed. The program address of this job is set to the address of the next instruction to be issued when it later resumes access to the central processor.

ACCOUNTING (JOB AND SYSTEM DAYFILES)

On mass storage, SCOPE maintains a chronological accounting of each job run, called the job dayfile, which is automatically printed at job termination. It contains a copy of all control cards processed, equipment assignments, diagnostic messages, central and peripheral processor time used, and the date and time of day associated with each processing event relative to the job. The job dayfile is maintained entirely by the system and is not accessible by user jobs.

SCOPE also maintains a system dayfile; it is a record of every processing event that takes place in the system, and relates to all jobs in the system. It contains all information in the job dayfiles plus other relevant system messages. The system dayfile can be printed, punched on cards, or copied onto tape at the operator's request.

OPERATOR/SCOPE COMMUNICATION

The computer operator communicates with SCOPE through the keyboard of the display console. SCOPE, in turn, transmits messages to the console's twin screens. Several displays can be requested, and certain kinds of messages are associated with each. The displays most often requested are job status, input/output queues, and dayfile displays. The job status display shows the progress of all jobs in process, their priority level, the most current program message, and other information. The input/output queues display presents the status of jobs not yet entered for execution and of executed jobs with output pending access to a peripheral device. In the dayfile display, the latest messages to the system dayfile are presented. Displays, operator requests and other operator messages may be found in the SCOPE 3.3 Operator's Guide.

JOB PROCESSING 2

Job processing is controlled through the use of SCOPE control cards. These cards identify programs and data files, and direct the sequence of program execution. SCOPE control cards are always found within the first logical record of a job, the control card record; they determine all operations on subsequent logical records.

JOB FLOW

The manner in which control cards handle a user program is illustrated by following a sample job as it is processed. For example, consider a job to assemble and execute a program written in COMPASS, with the output to be for use by the job printed on a line printer. A tape is also provided.

This job would be structured as follows:

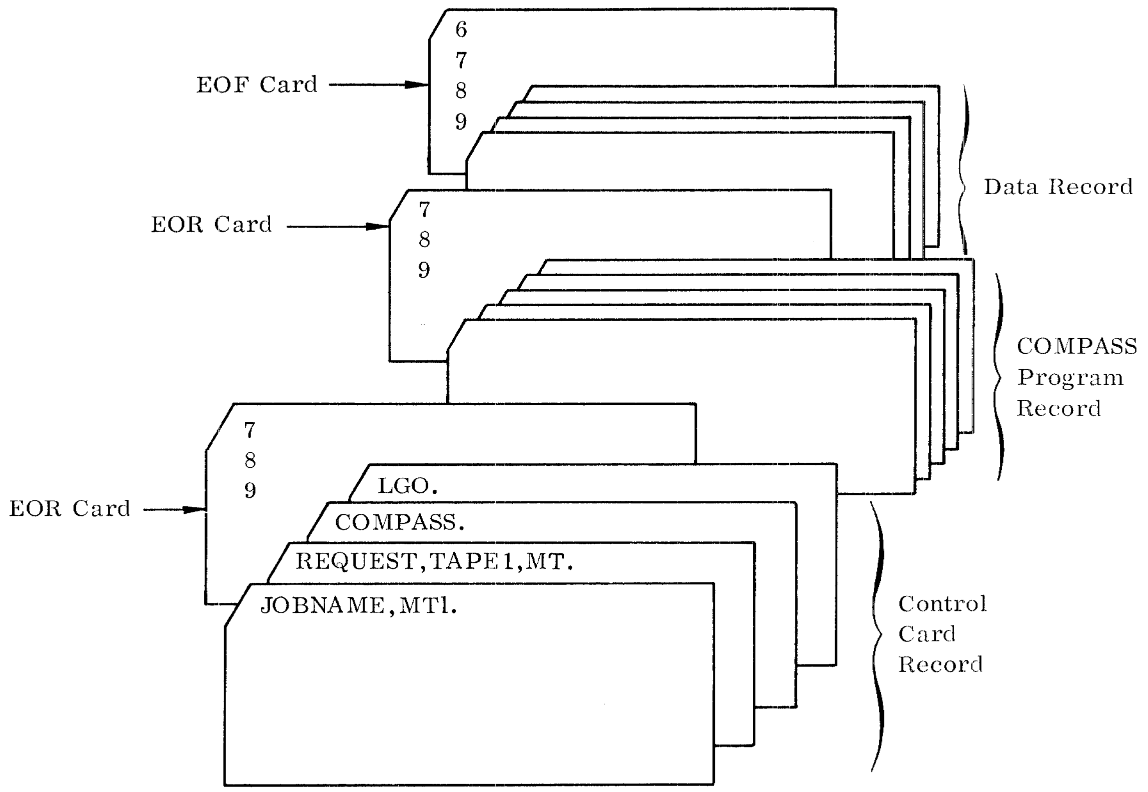
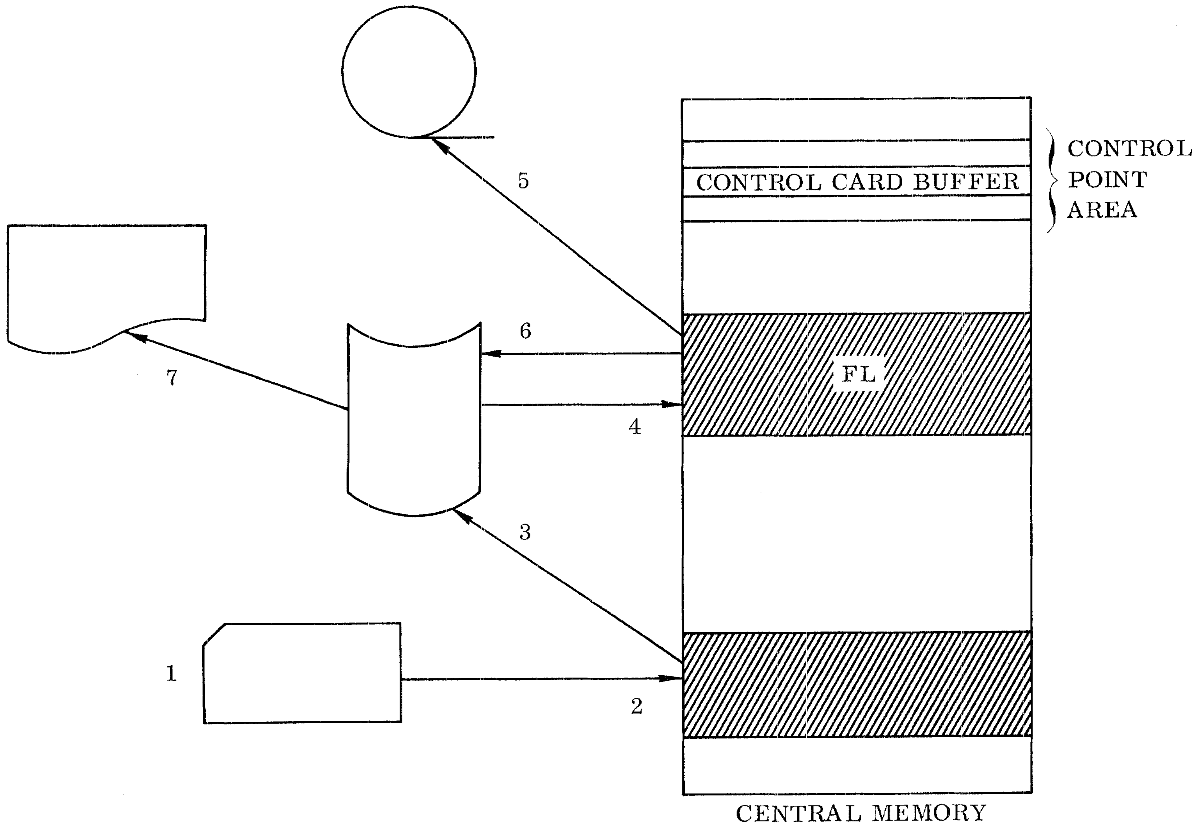


Figure 2-1. Sample COMPASS Job

Although this structure is typical of many small jobs, it is only one of many possible deck arrangements. Other sample structures are presented elsewhere in this manual. All jobs, however, share certain common features: each begins with a card that identifies the job and terminates with an end-of-file (EOF) card, formed by a 6/7/8/9 punch in column 1. Thus, the job itself actually forms a file.

In any job, the control cards, every user program, and every set of data cards each comprise one logical record. Each logical record, except the last in a job, is terminated by an end-of-record (EOR) card, formed by a 7/8/9 punch in column 1. The last logical record is terminated by an EOF card, which acts as both an EOR and an EOF indicator. Although only this card is required, some programmers terminate their jobs with both an EOR and an EOF card as a matter of preference.

When the sample job is input through the card reader, SCOPE calls a PP routine to translate the job card, check the validity of the entries on this card, and assign a priority to the job. Next the PP copies the job through a central memory input/output buffer onto mass storage. At this point, SCOPE identifies the job by its file name JOBNAME (from the job card), by the file type (input—meaning that the job is now part of the input queue), and by its assignment to control point 0. These three elements—file name, file type, and control point—are used to keep track of all jobs and other files in the system.



1	Job read into card reader	5	Some output to a scratch tape
2	Job read through buffer onto disk	6	Job assigned to output queue
3	Job in mass storage input queue	7	Output to printer
4	Job assigned control point; goes into execution		

Figure 2-2. Job Flow

When control point x is free, if sufficient central memory is available and the priority of the job permits, the job is assigned to that control point and assigned a field length in central memory. Then the file name is changed to INPUT, the file type to local (meaning the job is local to a user control point), and the control point number to x (the number of the control point).

SCOPE saves the original name of the job for later use. The file INPUT is positioned at the beginning of the second record (the user's program). The control cards are read into a buffer within the related control point area in low core, and are ready for execution. Each control point will be assigned a file named OUTPUT to collect job output and a file named DFILEx to contain the job dayfile. In the name DFILEx, x is the job's control point number.

Since the job card was processed when the job was read into the input queue, job control is advanced to the second card in the control card record—the REQUEST card. This card directs the computer operator to make a reel of tape available to the job for the file named TAPE1 (type local, control point x). When the operator mounts the tape and logically attaches it to the job by a keyboard entry, the job advances to the next control card.

The next card is the COMPASS control card, which directs assembly of the user's program. To accomplish this, SCOPE requests the loader to load the COMPASS assembler into the field length. Control passes to COMPASS, which assembles the next logical record (the user's program) on the file INPUT onto a mass storage file called LGO (type local, control point x). (For assembly or compilation, the user can designate files other than INPUT as an input file and files other than LGO as output by entries on the COMPASS control card; but unless such alternative files are named on the assembly or compilation card—the COMPASS card in this case—INPUT and LGO are used by default.) COMPASS also writes a source-language listing of the program onto the file named OUTPUT. Control then proceeds to the next control card, LGO.

The LGO card directs the program execution. The loader loads the LGO file containing the user's program in object code into central memory and writes a map of this program onto the file OUTPUT; library subprograms required are also loaded. Control then passes to the user's program for execution, input data is read from the third record on the INPUT file (the user's data) and output is written on TAPE1 and OUTPUT.

As each control card is executed, it is copied onto DFILEx. The last card to be processed in the INPUT file is the EOR card, signalling the end of the job. SCOPE writes the central and peripheral processor running times on DFILEx and copies this file onto OUTPUT, which is then detached from the control point. The name OUTPUT is changed to JOBNAME (the original job name), the file type to output, and the control point to 0. TAPE1 is also released so that the tape unit may be available for another job. INPUT, LGO, and DFILEx are cleared and released from SCOPE control. All equipment associated with the job is released from control point x and assigned to control point 0, where it can be requested by other jobs. The control point area and field length in central memory are also made available for other jobs. When a printer is available, JOBNAME, containing the assembly-language program listing, load map, output, and dayfile, is printed.

PROCESSING WITH CONTROL CARDS

Control cards contain one or two fields; the first, which is required, contains a flag word that requests action. Certain flag words are reserved for SCOPE and cannot be declared as the names of user programs. The second field is optional; it contains one or more parameters to be used with the flag word.

If only the flag word appears, it must be terminated with a period or a right parenthesis. If a parameter is used, it must be separated from the flag word by a blank or by any character other than an alphanumeric character (A-Z and 0-9) or an asterisk. Multiple parameters must be separated from each other by any characters other than alphanumeric, blank, or asterisk. After the last parameter, a period or a right parenthesis must terminate the field. All blanks in the parameter field are ignored when the card is processed. Comments consisting of any characters can be written to the right of the terminator.

As an example, consider a control card to request that a file named TAPE2, on tape reel number 4326, be assigned to a job. The flag word is REQUEST, and TAPE2 and MT (for standard-density magnetic tape) are parameters. MOUNT TAPE 4326 is a comment to the computer operator to mount this tape. This card could be written in any of the following formats:

```
REQUEST,TAPE2,MT.MOUNT TAPE 4326
REQUEST(TAPE2,MT) MOUNT TAPE 4326
REQUEST TAPE2,MT. MOUNT TAPE 4326
```

The flag word on each control card processed names a program or subprogram that is called into memory and executed. When the flag word is read, the PP advancing the job searches a table of all active files in central memory for a file name that matches the flag word and is attached to the control point of the job. If no such name is found in the file name table, the PP searches its own memory for a program name identical to the flag word. If this name is not found, the SCOPE library is searched. If it does not contain the file name, the job is terminated; if the file is located, it's contents will be loaded into memory and executed. Control will then advance to the next control card.

Since the user file tables are searched before the SCOPE library, a user could substitute his own program or subprogram for execution in place of a library program by storing it as a file. For example, if a user wanted his job assembled under his own version of COMPASS, he could request assignment of the file containing this version to the control point of his job. When the COMPASS control card was processed, the name of this file would appear in the central memory table forestalling a search of the SCOPE library for standard COMPASS. The user's version of COMPASS would be loaded and executed.

JOB IDENTIFICATION AND CONTROL

IDENTIFYING JOBS
(Job Card)

Job identification, allocation of certain resources, and establishment of processing priority level is accomplished with the job card. This card is always the first in the job deck. Its format is:

n,Tt,CMfl,ECfl,Pp,Dym,MTk,TPk,NTk,CPp.

In this and other formats described, capital letters indicate constants and small letters stand for variables.

REQUIRED PARAMETERS

The flag word, n, is the name the user assigns to the job to uniquely identify it to SCOPE. Only this entry is required on the job card. It can range from one to seven characters. Any combination of numbers or letters can be used; the first character must be a letter. The job name must begin in column 1 and if no parameters appear, it must terminate with a period or a right parenthesis.

SCOPE automatically modifies the name of every job by replacing the sixth and seventh characters with a number that differs for each job. This ensures unique identification if a job is entered with a name identical to that of another job already in process. For example, if two jobs are named JOBNAME, one may be processed as JOBNA23 and the other as JOBNA34. If a job name contains less than five characters, SCOPE fills with zeros all unused characters through the fifth and adds unique sixth and seventh characters.

OPTIONAL PARAMETERS

Parameters may be included on the job card to specify certain resources, priority levels, or processing time limitations. If these parameters are omitted, SCOPE automatically assigns system default values established when SCOPE is installed. These parameters can be listed in any order following the job name.

SCOPE ignores any unknown parameters that appear on the job card. However, when improper characters are used as variables with valid parameters, the job will be terminated. For example, on the following job card, CMABC and MTZOT would cause the job to abort.

```
JOB,T100,P17,CMABC,MTZOT.
```

SCOPE interprets all numbers on job cards as octal values, unless otherwise noted in this manual, or unless this procedure is redefined by the systems analyst when he installs SCOPE.

The optional parameters follow:

Tt

t is an octal value for the time in seconds which the user estimates his job will require the central processor. This value must include the time required for assembly or compilation; it does not include time during which the job is in the input queue or in central memory but not using the central processor. If the job access to the central processor exceeds the value specified by t, the job will be terminated prematurely. t may be any octal number not to exceed five digits. An infinite time can be specified by 77777; the job will proceed until completed even if it exceeds the installation default value for t. Users should be cautious in assigning 77777 as a time limit; however, certain kinds of program errors, such as an infinite loop can result in great waste in such cases.

To determine the value of t, estimate the minutes of central processor time a job will need and multiply this value by 100. The result will approximate the central processor octal time value in seconds. For example, when a time limit of 4 minutes is estimated, t can be derived as follows:

t = 4 min. x 100 = 400 seconds (octal).

The job card could be written in any of these formats:

jobname,T400.

jobname(T400)

jobname T400.

CMfl

fl is the field length in number of central memory words that the user estimates his job will require. The user must include space for the SCOPE loader routines needed to load his job into central memory, as well as for the assembler or compiler requirements. If this parameter is included, the job will not be assigned central memory until the specified storage is available. Most COMPASS or compiler language jobs require at least 43000 words; when in doubt, the user should specify this value or omit the parameter to elect the default value. The highest permissible value is defined by an installation parameter. SCOPE will round any value upward by a multiple of 100. Thus, a job requesting 40110 words of storage would be assigned 40200 words. Typical control card entry is:

```
jobname,T300,CM43000.
```

ECfl	fl is the amount (octal) of ECS the user anticipates his job will need, in multiples of 1000-word blocks. Generally, this value is specified only for jobs that require large amounts of data. The highest value permitted is defined by an installation parameter. No ECS will be assigned if the parameter is omitted. This job card would ensure 4000 words of ECS:

```
jobname,CM43000,EC4.
```

Pp	p is the processing priority level requested for a job. The lowest executable priority level is 1. The highest value permitted depends on an installation parameter, but it can never exceed octal 7777. This value places the job ahead of all other jobs of different levels in the system. For a job of very low priority, the job card might be punched as follows:

```
JOB,CM43000,P1.
```

Dym	This parameter is used only in conjunction with job dependency in a string of interdependent jobs. y is the dependency identifier (two alphabetic characters) assigned by user to the entire string. m is the dependency count (number) of jobs (0-77 octal) upon which this particular job depends. Examples showing how the D parameter is used are presented in the discussion of the TRANSF card.
MTk NTk	When tape is required by a job, one of these parameters must be selected. MT specifies seven-track tape and NT nine-track. k is the maximum number of seven or nine-track tape units that a job will require at any one time. The value of k can range from 0 to 77(octal), but cannot exceed the total number of tape units at the computer site. If more tape units are required at any time during job execution than are specified by k, the job will be terminated. A job can use more than a total of k tape units only if their use is not simultaneous. For example, if k is 3 and tape units A, B, and C (with seven- track tapes) are assigned to the job, and tape unit C is later unloaded and released from the job, tape unit D can be requested for the job. This makes a total of 4 tape units used during the entire job. For this job the card might appear as:

```
TAPEJOB,MT3.
```

If both seven and nine-track tapes are used, MTk and NTk should both be noted.

TPk	TP has the same meaning as MT. But if both TP and MT appear on the job card, only the MT parameter will be used to determine the maximum seven-track tape units needed.

CPp This parameter should appear only when the job is to be processed on a 6700 computer. p indicates which central processor a user requests for the job. When p is B, the 6400 processor will be assigned. When p is A, the faster 6600 processor is used. If p is not specified, SCOPE will assign either processor. This example specifies use of the 6600 central processor:

```
M2,CM43000,CPA.
```

The CP parameter is used on the 6500 configuration for diagnostic routines only.

After the terminator following the last parameter, general comments or messages to the operator can appear on the job card (as on any control card).

Examples of job cards:

```
J2,T4,CM43000,EC2,P1,DAB3,MT5,CPA.  THIS IS A VERSION 3.3 JOB.

T023 EC1,MT1(CM43000)

STAR(T3)

OSCAR.COBOL V3 USED

S3R2,MT1. FIRST RUN.

DOGCAT,CM50000.
```

LIMITING MASS STORAGE (LIMIT CARD)

Normally, a job being processed will be assigned as much mass storage space as required, if available. Sometimes, a user may want to limit the maximum mass storage that can be assigned. This may be done, for example, when a heavy job load is imposed on the computer and a user wants to be certain that his job does not degrade overall throughput by using too much mass storage.

Mass storage is limited by using the LIMIT card:

LIMIT,n.

n is a decimal number, indicating the number of blocks of 4096 60-bit words to which mass storage should be limited for this job. A user wanting to limit his job to 40960 words of mass storage would set n to 10, since 10 x 4096 = 40960. His LIMIT card would appear:

```
LIMIT,10.
```

If the job requires mass storage in excess of the specified limit at any time, the job is terminated.

EQUIPMENT AND FILE ASSIGNMENT

REQUESTING EQUIPMENT FOR A JOB (REQUEST CARD)

Before a file can be referenced by a job, the device on which a file resides or on which it is to be written must be assigned to the job. SCOPE makes this automatic assignment for files input from punched cards or output to a printer or card punch. All card input is written on the mass storage file INPUT, printer output on the mass storage file OUTPUT, and card output on the mass storage files PUNCH (for Hollerith cards) or PUNCHB (for binary cards). When card input is referenced by a job, INPUT is read automatically. When the job is terminated, OUTPUT is printed, PUNCH is punched on Hollerith cards, and PUNCHB is punched on binary cards.

In addition, all local files created while a job is in progress are written automatically to mass storage unless another device is specified by the user. Thus, if a job created a file named SAMMY and did not specify a storage device, SAMMY would be written on mass storage.

Equipment must be requested by the user for input files from devices other than mass storage, card punch or printer. For instance, tapes containing referenced files must be requested in the control card record with REQUEST cards. Disk pack mounts must be requested by RPACK cards. Since control cards are processed in order of appearance, the REQUEST card for a particular file must precede the control card that executes the program referencing that file; otherwise, the file will be assumed to be on mass storage when it is referenced. When the REQUEST card is encountered, the job will be terminated with the message DUPLICATE FILE NAME.

When the REQUEST card is processed, job processing halts. The file equipment requested by the job is assigned automatically or by operator action. In the latter case, the equipment request is shown on the display screen to the computer operator. The operator makes the device physically available by mounting a tape or disk pack, or turning on a device. He then makes the device logically available by entering a command on the console keyboard; this assigns the device to the control point of the requesting job. Processing is then resumed, and the device is recognized as the source or destination of the associated file.

Format of the REQUEST card is:

REQUEST,lfn,dt,eq.

REQUIRED PARAMETERS

The flag word, REQUEST, and the lfn parameter are required on the REQUEST card. As with the job card and all other control cards, any of the standard separators and terminators can be used. If a parameter is listed more than once or is in error, a diagnostic message will be printed and the job will be terminated.

lfn — The logical file name is the name by which the user refers to this file in his program. Logical file names can be from 1 to 7 characters, beginning with a letter. When lfn is the only parameter listed, the operator can choose the device to be assigned to the job for the file. For example, when any of the following cards is processed, the operator can assign disk, drum, tape, or any other available device for the file:

```
REQUEST,FILE1.

REQUEST(FILE1)
```

OPTIONAL PARAMETERS

Files are assigned to the devices named by optional parameters. They can be assigned to a device type (for example, to any tape unit or disk available) with the dt parameter or to a specific device such as a particular tape unit or line printer) with the eq parameter (see page 2-15).

dt	The dt parameter designates the type of device on which a file is recorded or on which it is to be written.

ALLOCATABLE DEVICES (MASS STORAGE)

For allocatable devices (disks, disk packs, drums, and ECS), the following codes can be used for dt:

Device	dt
6603-I disk	AA
6638 disk	AB
6603-II disk	AC
3637/865 drum	AD
814 disk	AF
3553-1/821 disk file	AL
3553-1/841 multiple disk drive	AM
3234/854 disk pack controller	AP
ECS	AX

The following example would request the operator to relate FILE1 to any 6638 disk available:

```
REQUEST,FILE1,AB.
```

When the dt parameter for a device is preceded by an asterisk, operator action is not required and SCOPE assigns the equipment automatically. The following card would request SCOPE to automatically assign a 6638 disk for FILE1.

```
REQUEST(FILE1,*AB)
```

The allocation style may be indicated by a suffix to the dt parameter of two octal digits. This allocation style can be used to assign the file to a subset of the device specified. If more than one RBR exists for a device and each has a unique allocation style, this parameter can be used to assign a file to the RBR with the desired record block size.

Example:

If the 6603 has two 1024-bit RBR's, the first with allocation style 02 and the second with allocation style 01, then a file can be written on inner zone only (50 PRU/RB) by requesting AA01 and on the outer zone only (64 PRU/RB) by requesting AA02.

Less specific requests can be issued for mass storage devices by using the following codes in the dt parameter. The previous rules for automatic SCOPE assignment (request with asterisk) and recording technique (request with numerical suffix) apply:

Device	**dt**
Any mass storage device for a local (temporary) file	A*
Any mass storage device for a file subsequently to be cataloged as a permanent file.	*PF

For example, the following card would request the operator to relate FILE1 to any mass storage device available for a local file:

```
REQUEST,FILE1,A*.
```

The OV parameter may be specified on the REQUEST card for mass storage files. This parameter indicates that special processing is to occur when mass storage is not available to meet all the requirements specified when the file was created. When the OV parameter is present and no mass storage space is available on the specified device or the specified equipment type or allocation style is not available, the system will assign the overflow of the file to any mass storage device available. Also, a device-capacity- exceeded status is returned to the user when the following conditions hold: when OV is present and mass storage of any type is not available and the FST for the file contains the address of an FET in which the EP bit is set. Files assigned to a permanent file device cannot overflow to a non-permanent file device regardless of the specification of the OV parameter.

NON-ALLOCATABLE DEVICES (TAPES)

For magnetic tapes, dt can be MT for standard 7-track SCOPE, S, or L tapes, and NT for 9-track tapes. As with allocatable device codes, automatic assignment not requiring operator intervention can be requested by prefixing the code with an asterisk. The following card requests automatic assignment of standard magnetic tape for FILE1, for scratch purposes.

```
REQUEST,FILE1,*MT.
```

When a file requires more than one tape for processing, dt may be prefixed with 2. The operator will assign two tape units for the file, and tapes will be used in the order assigned. When the end of the reel is reached on the first tape unit, processing continues with the tape on the second unit, while the first tape is rewound and unloaded. The operator mounts another tape on the first unit, so that it can be processed when the tape on the second unit reaches the end of the reel. Alternation of units continues until the file is no longer referenced. Since this process requires operator action, the asterisk for automatic assignment cannot be used when more than one tape unit is required. This example requests the operator to assign all tapes necessary for FILE1.

```
REQUEST,FILE1,2MT.
```

Within the dt field, other characteristics of the tape can be requested by including additional parameters with MT or NT. Parameters can appear in any order; standard separators and terminators are required. Examples appear at the close of this discussion.

7-Track Tape (MT)

Density

The density (bits per inch) for recording information on 7-track tape can be specified by a parameter used in place of MT. Automatic assignment (asterisk) or multitape files (2 prefix), can be requested by prefixes to MT or to the codes below.

Density (bpi)	**dt**
200	LO
556	HI
800	HY

If MT is declared, unlabeled tapes are read or written according to an installation parameter. ANSI-labeled input tape is read at the density specified in its volume header label. ANSI-labeled output tape is written at an installation declared density.

SPECIAL USAGE (CK OR MF)

A tape containing more than one file is a multifile tape and must be so defined by including MF within the dt field. Such a tape cannot be assigned automatically; if the field is prefixed with an asterisk, the prefix will be ignored.

If a tape is to be used for a checkpoint dump (described in Section 9), CK must be included within the dt field to designate the tape for this purpose.

Data Formats (S or L)

A tape in the standard SCOPE tape format need not be designated by a REQUEST card parameter. However, if the tape is written in either S or L format, S or L must be specified within the dt field.

Label Formats (U or Y)

If the tape is unlabeled, no parameter is needed. For tapes with labels in the standard ANSI format, the dt field must contain U. If labels are written in other formats, Y is required in the dt field.

Initial Use (E or N)

For labeled tapes, the initial use of the tape can be specified by entering E (existing) for input and N (new) for output. With labeled tapes, automatic assignment (asterisk) cannot be requested; such a request will result in the assignment of an unlabeled scratch tape. The following chart indicates label processing which takes place when E or N parameters are used in combination with U or Y parameters.

Label Format Type / Initial Use Specification	U	Y	Omitted
E	ANSI format label checked	Custom format label checked	ANSI format label checked
N	ANSI format label written	Custom format label written	ANSI format label written
Omitted	ANSI format label written	Custom format label written	Unlabeled input or output tape

Two additional parameters, IU and SV, can be used to control the disposition of the tapes at the end of the job when IU is specified, the unload request or function will be ignored for the tape referenced on this REQUEST card. When SV is specified, the tape referenced will be unloaded and saved at job termination. The tape unit will be turned off logically; before it is reassigned, this unit must be turned on again.

Examples of REQUEST Cards for 7-Track Tape

The following card requests an operator to assign an output tape for a multifile group with the logical file set name MANYF. All files in the set will be written at 800 bpi in S tape format:

```
REQUEST(MANYF,MF,S,N,HY)
```

This card requests an input tape file B2 written in standard installation density. The tape has labels, written in ANSI format.

```
REQUEST,B2,MT,U,E.
```

9-TRACK TAPE (NT)

Density

The density in which information is to be recorded on a 9-track tape can be specified with parameters used with, or in place of, NT. Automatic assignment (asterisk prefix) or multitape files (2 prefix) can be requested by prefixing these parameters, but in such cases, NT must be omitted:

Density (bpi)	**dt**
1600	PE
800	HD

If NT is declared, but PE and HD are absent, unlabeled and labeled tapes are written according to an installation parameter. For this type of tape, density setting is effective only when tapes are written; the density at which 9-track tapes are read is a function of the hardware.

Special Usage

The parameters for declaring a multifile tape for 7-track tapes also apply to 9-track tapes; however 9-track tapes cannot be used as checkpoint tapes.

Label Formats

The parameters for denoting label formats for 7-track tapes also apply to 9-track tapes.

Initial Use

The parameters for declaring the initial use of labeled 7-track tapes also apply to labeled 9-track tapes.

Coded Data

On 9-track tapes, data can be read or written in the display code used by Control Data or in binary code. To convert between Control Data code and ANSI, the parameter US should be included on the REQUEST card. To convert between Control Data code and EBCDIC, EB should be included on the REQUEST card. If neither US or EB is included, code will be converted in accordance with an installation parameter.

As with 7-track tapes, special termination procedures can be requested by adding the IU or SV parameters to the REQUEST card.

Examples of REQUEST Cards for 9-Track Tapes

This card requests that an ANSI-labeled 9-track input tape containing FILE1 be assigned to the job; the label will be checked when FILE1 is first referenced by the user program. The EBCDIC character code in which the tape is written will be converted to Control Data character code when it is read.

```
    REQUEST(FILE1,NT,U,E,EB)
```

The following card requests that an unlabeled 9-track output tape of 1600 bpi density be assigned for FILE2.

```
    REQUEST,FILE2,PE.
```

UNIT-RECORD DEVICES

If a file is input from a card reader or output to a printer or card punch, devices are assigned automatically by SCOPE; the REQUEST card is not necessary. However, if a special type or model of card reader, printer, card punch, paper-tape reader or punch, terminal keyboard, hard copy or microfilm recorder is to be used, this device must be requested. Such equipment can be requested with the following codes. Devices for which codes are defined but software is not supported are denoted with an asterisk.

Device	Code
*Paper tape reader	TR
*Paper tape punch	TP
501, 512, or 505 line printer	LP
501 or 505 line printer only	L1
512 line printer only	L2
405 Card Reader	CR
Remote Terminal Keyboard	KB
200-User Terminal Card Reader or Teletype Paper Tape Reader	CR
200-User Terminal Line Printer or Teletype Paper Tape Punch	LP
415 Card Punch	CP
6612 Keyboard/Display Console	DS
*252-2 Graphic Console	GC
*253-2 Hard Copy Recorder	HC
*254-2 Microfilm Recorder	FM
*Plotter	PL

*Codes are defined but supporting software is not provided under SCOPE.

REWINDING FILES (REWIND CARD)

In most cases, when a file is requested for a job, that file is positioned automatically at its beginning of information. However, because of variations in installation parameters and procedures, automatic positioning may not always occur with every file requested. Therefore, it is best to follow the REQUEST card with a REWIND card to ensure that the file will be positioned at its beginning when first referenced.

The REWIND card has the following format:

REWIND,lfn. or *REWIND,lfn1,lfn2,lfn3,...lfnn.*

lfn is the name of the file to be repositioned. More than one lfn, separated by commas, may appear on one REWIND card.

The tape containing file MAX is requested for the following job; the file is repositioned, loaded into central memory and executed.

```
Cards                  Function

jobname,MT1.           Names job.
REQUEST,MAX,MT.        Requests tape containing object program file MAX.
MAX.                   Loads and executes MAX.
REWIND,MAX.            Rewinds MAX.
MAX.                   Loads and executes MAX a second time.
7/8/9
data deck
7/8/9
second data deck
6/7/8/9                Signals EOF and end of job.
```

In the above example and the examples that follow, some control cards are referenced before they are fully explained in the text. In such cases, a brief notation of the card's purpose is presented beside the card.

REQUESTING PRIVATE DISK PACKS (RPACK CARD)

With private disk packs users can record files on high speed non-allocatable mass storage devices. These files can be accessed only by the job to which they are assigned. The restriction to files local to a job results from dedication of the private pack to the control point of that job. The private pack cannot be used for common or scratch files. When a private pack is assigned to a job, the names of the logical files it contains are written into central memory along with information needed to access them. When the job terminates, this information is rewritten onto the pack before it is logically unloaded. Then, the pack can be removed from the drive with the files intact, for later use.

CREATING FILES ON NEW PACKS

Files are created on new, unused private packs in a three-phase operation. First, the private pack is requested for the user's job. Next, the names of the files are associated with the private pack. Finally, the files are written on the private pack.

To request that a private pack be assigned to a job, the RPACK card is used, in the following format:

RPACK,pname,N.

On this card, pname is the pack name to be assigned to the entire pack; it is not to be confused with the logical file name associated with each file to be written on the pack. However, the pack name and the name of one of its files can be identical. The sole function of the pack name is to identify and reserve this pack for the job. Another parameter, N, indicates that this is a new pack, as yet containing no files.

As an example, this card assigns the name MYPACK to a new private pack:

```
RPACK,MYPACK,N.
```

When the RPACK card is processed, SCOPE writes a label on the pack; the label contains pname and other information used by the system. This label is used to identify the pack and list the files it contains. At this time, also, a message displayed to the computer operator directs him to assign a visual pack number or identifier to the pack. He does this by typing a six-character identifier at the console keyboard. To direct the operator in assigning a special number, COMMENT cards (detailed later in this section) can be placed before the RPACK card in the control card record.

Next, the names of files to be written on the pack must be assigned to the pack. This, in essence, reserves areas on the pack for these files. The REQUEST card is used in the following format:

REQUEST,lfn,PK,pname.

On this card, lfn is the logical file name and pname is the pack name defined on the RPACK card. Assignment is automatic and requires no operator action. Prefixes and suffixes previously defined for the REQUEST card, however, cannot be used with PK. The following card associates the file named MYFILE with the private pack MYPACK.

```
REQUEST,MYFILE,PK,MYPACK.
```

The above entries are the only ones required in the control card record. The new files will be created and updated on the private pack by commands within the user program record.

As a complete example, the following FORTRAN job creates two files, TAPE1 and TAPE2, residing on a private disk pack named MYPACK having an identification number of N1122.

Card	Function
JOB1,T1000.	Names job.
RUN(S)	Compiles program.
COMMENT. OPERATOR SHOULD ASSIGN BLANK	
COMMENT. LABELED PRIVATE PACK AND HE SHOULD	
COMMENT. TYPE IN A VSN OF N1122 IN ANSWER	
COMMENT. TO THE FOLLOWING RPACK REQUEST.	
RPACK,MYPACK,N. VSN N1122	Tells operator to assign new pack named MYPACK to job.
REQUEST,TAPE1,PK,MYPACK.	Assigns file TAPE1 to MYPACK.
REQUEST,TAPE2,PK,MYPACK.	Assigns file TAPE2 to MYPACK.
LGO.	Loads and executes program.
7/8/9	Signals EOR.
(FORTRAN program to create files TAPE1 and TAPE2)	
6/7/8/9	Signals EOF.

USING EXISTING PACKS

A parallel method is used in requesting and referencing files on existing packs. First, the private pack containing the files must be requested with an RPACK card of this format:

RPACK,pname,E.

On this card, pname is the name assigned to the pack and E indicates an existing pack. When this card is processed, the operator assigns a pack already labeled with pname to the job. If pname in the label does not correspond to pname on the RPACK card, the operator is informed; job processing is delayed until he assigns the proper pack.

At the user's option, another parameter, vsno, can be included as follows:

RPACK,pname,E,vsno.

This is the volume serial number assigned to the pack. If this parameter in the label does not match its counterpart on the RPACK card, processing is delayed until the proper pack is assigned.

When the proper pack is assigned, its files are made available to the job and other files can be added to it by REQUEST cards.

The following example shows a FORTRAN job that accesses two existing files, TAPE1 and TAPE2, previously created by the job just illustrated, and writes a new file, TAPE3, on the same pack.

No REQUEST cards are needed to associate TAPE1 and TAPE2 with the pack, because these files already reside on the pack.

Card	Function
JOB2,T1000.	Names job.
RUN(S)	Compiles program.
COMMENT. THE OPERATOR MUST ASSIGN	
COMMENT. PRIVATE PACK WITH VSNO	
COMMENT. OF N1122 TO FOLLOWING RPACK	
COMMENT. REQUESTED.	
RPACK,MYPACK,E,N1122.	Tells operator to assign pack named MYPACK to job.
REQUEST,TAPE3,PK,MYPACK.	Assigns new file,TAPE3, to MYP-ACK.
LGO.	Executes program.
7/8/9	Signals EOR.
(FORTRAN program to read files named TAPE1 and TAPE2 and create a third file TAPE3.)	
6/7/8/9	Signals EOF.

DISPOSING OF FILES BEFORE END OF JOB (DISPOSE CARD)

Under normal operating conditions, files are released from a job only after all control cards have been processed and the job is released from central memory and terminated. However, a user can request that local files on allocatable devices (public packs, disks, drums, or ECS) be released for termination processing before the job is completed. This could be done, for instance, when output is ready for the printer but additional programs in a job remain to be run.

To release files for termination processing, the DISPOSE card is used in either format:

DISPOSE(lfn,x=ky) *DISPOSE(lfn,*x=ky)*

On this card, lfn is the name of the logical file to be released. This must be a temporary local file residing on an allocatable device. If lfn is a permanent file, the DISPOSE card is ignored. When only the lfn parameter is used, the file referenced is released from the job and the system; it can then be overwritten by other jobs. The following card will release the file named EXFILE from the system:

```
DISPOSE(EXFILE)
```

The x parameter represents the disposition requested. It is used when a disposed file is to be printed, punched on cards, or output on some other device as indicated by the following codes.

File Disposition	x Code
Printed on any available printer	PR
Printed on 501 or 505 printer	P1
Printed on 512 printer	P2
Punched on formatted binary cards	PB
Punched on Hollerith cards	PU
Punched on field-free binary cards (using all 80-columns for data)	P8
Printed on microfilm recorder	FR
Plotted on microfilm recorder	FL
Plotted on any available plotter	PT
Printed on hardcopying device	HR
Plotted on hardcopying device	HL

Codes FR, FL, PT, HR and HL are defined but not supported in SCOPE 3.3

The presence of * in the x parameter indicates the file is to be disposed at end-of-job. In that case, DISPOSE will ignore the =ky part of the parameter and the disposition code specified by x will be put into the FNT. If no FNT entry exists when the DISPOSE card is encountered, one will be created.

This example shows the x parameter on a card that requests early disposition of the file named ZOOK for printing on any available printer:

```
DISPOSE(ZOOK,PR)
```

Further characteristics of the output device can be specified by expanding the x parameter to x =ky. For example, when k is C, the identifier y must be formed by two alphanumeric characters, defined by installation, to request certain hardware characteristics. The following job containing two FORTRAN programs illustrates how these identifiers are used.

The first program produces a group of checks on a file called PAYROLL. The second program produces other unrelated data. To permit the payroll checks to be printed on a 512 printer while the second program is in process, the dispose card could be used as indicated.

Card	Function
`JOBPAY.`	Names job.
`RUN(S)`	Compiles first program.
`LGO.`	Loads and executes first program.
`DISPOSE(PAYROLL,P2=CPR)`	Prints PAYROLL on 512 printer.
`REWIND(LGO)`	Rewinds compiler output file.
`RUN(S)`	Compiles second program.
`LGO.`	Executes second program.
`7/8/9`	Signals EOR.
(First FORTRAN Program)	
`7/8/9`	Signals EOR.
(Second FORTRAN Program)	
`6/7/8/9`	Signals EOF.

On the DISPOSE card, P2 specifies the 512 printer, C determines that special characteristics are required of this printer, and PR is a parameter that the installation interprets to mean that printing is to be done on special payroll forms. A message to the operator will inform him that printing will be delayed until the forms requested by the PR identifier are placed in the printer. After the operator loads the forms in the printer, he enters a command at the console to begin printing.

The ky parameters can also be used to indicate other procedures. For example, when k is I (for an INTERCOM terminal) or E (for an EXPORT/IMPORT terminal), y designates a specific remote site to which output is to be directed.

RELEASING FILES FROM JOBS (RETURN CARD)

Normally, all files assigned to a job are retained by that job until it terminates. However, any file, permanent or temporary, common or otherwise, allocatable or non-allocatable, can be released from the job prior to termination with the RETURN card. When a RETURN card appears, a CLOSE,RETURN is performed on each file named, unless it has a special disposition code set. In that case, the special disposition is honored. RETURN also causes a decrease in the number of logical tape units reserved by the job. A magnetic tape assigned to a member file of a multifile set is not released.

When the RETURN card is processed, sequential files are rewound, files on magnetic tape are unloaded, files on private packs are locked and made unavailable to the job. Common files are released from the users job but remain in the system for access by other jobs. The card format is:

RETURN,lfn1,lfn2,...lfnn.

On this card, lfn is the name of the file released; one or more files can be referenced on one card.

The following card would release the files MUTT and JEFF from the job.

```
RETURN,MUTT,JEFF.
```

UNLOADING FILES (UNLOAD CARD)

Like the RETURN card, the UNLOAD card also can be used to release files from the job before termination. When an UNLOAD card is encountered, a CLOSE.UNLOAD is performed on each file named. Tape units are returned to the control point pool and may be used for another job. In contrast to the RETURN card, the number of units logically reserved by this job is not decreased. Otherwise, an UNLOAD is equivalent to a RETURN.

The format of the UNLOAD card is:

UNLOAD,lfn1,lfn2,...lfnn.

REMOVING FILES FROM PRIVATE PACKS (REMOVE CARD)

A file can be removed from a private disk pack by first issuing the RPACK request to assign the pack to the job and then issuing the REMOVE request. When the REMOVE request is processed, all space on the disk occupied by that named file is released and the file is dropped automatically from the system by SCOPE. No operator action is required.

The REMOVE card is punched in the following format:

REMOVE,lfn,pname.

REMOVE and lfn are required entries; lfn is the file to be removed from the pack.

The pname parameter, for pack name, is optional; when it is used, SCOPE compares pname on the REMOVE card with pname on the header label on the private pack. If pname is the same on both items, the file will be removed; if it is not, a message is issued to the operator.

The following sample job will remove the file named TAPE2 from the private disk pack named MYPACK, which is assigned a visual identifier of N1122.

```
JOB3,T1000.             Names job
RPACK,MYPACK,E,N1122.   Requests pack named MYPACK for job.
REMOVE,TAPE2,MYPACK.    Removes file TAPE 2 from MYPACK.
6/7/8/9                 Signals EOF.
```

CREATING COMMON FILES (COMMON CARD)

Normally, when a user's job terminates, all files not designated for output or recorded on a private disk pack are released from the system. However, with the COMMON card, a user can request that any files associated with his job remain in the computer for later access by other jobs. The format of the COMMON card is:

COMMON,lfn.

Here, lfn is the name of the logical file declared common.

Common files are created by inserting the COMMON card in the control record at any point after the file is first referenced. For example, the following COBOL job creates a file called TASK during execution. The user wants to ensure that TASK is left in the system for reference by a later job. He inserts the COMMON card after the card that executes the COBOL program:

Card	Function
`JOB4.`	`Names job.`
`COBOL.`	`Compiles COBOL program.`
`LGO.`	`Loads and executes COBOL program.`
`COMMON,TASK.`	`Leaves TASK in mass storage as a common file.`
`7/8/9`	
`(COBOL program)`	
`6/7/8/9`	

A file made common by a job is assigned to the job until the job terminates. It is then assigned to SCOPE until it is referenced by another job.

When a job references a file declared common by a previous job, the COMMON card must be used before that reference in order to assign the file to the present job. If the common file is not in use by another job, it will be assigned to the requesting job immediately. Otherwise, processing of the requesting job will be suspended until the common file is free. When the common file is assigned, it stays with the requesting job until the job terminates.

In the following job, an existing common file named MONEY is requested by the job, loaded into central memory, and executed.

Card	Function
`JOB5.`	`Names job.`
`COMMON,MONEY.`	`Requests file MONEY for job.`
`LOAD(MONEY)`	`Loads MONEY.`
`EXECUTE.`	`Executes MONEY.`
`6/7/8/9`	`Signals EOF.`

When a non-existent common file is requested, job processing is suspended until such a file is created by a job and becomes available. Processing then continues.

A file on magnetic tape or private disk pack cannot be declared common. Such a declaration has no effect, and a diagnostic message is issued.

Common files are destroyed during deadstarting of the computer following a shutdown. However, information can be saved even through a deadstart if it is written on a permanent file (Section 5).

RELEASING A COMMON FILE (RELEASE CARD)

Any common file can be released from common status with the RELEASE request. When this request is processed, the file referenced is dropped from common status and is assigned to the requesting job as a local file. When the job is terminated, this file is destroyed along with all other temporary local files.

The RELEASE request is written in the following format:

RELEASE,lfn.

lfn is the name of the file to be released.

In the following example, an existing common file called CASH is requested for the job, loaded, executed, and released from the systems:

Card	**Function**
`JOB6.`	`Names job.`
`COMMON,CASH.`	`Requests file CASH for job.`
`LOAD(CASH)`	`Loads CASH.`
`EXECUTE.`	`Executes CASH.`
`RELEASE,CASH.`	`Releases CASH from common status.`
`6/7/8/9`	`Signals EOF.`

TAPE JOB PROCESSING

JOB INITIATION

A job's tape requirements are among the criteria used by SCOPE to schedule a job to a control point. The maximum number of tape units needed at any one time by a job is given on the job card. SCOPE will not schedule a job until the system tape pool contains enough available tape units to satisfy tape requirements; other scheduling criteria must also be satisfied.

When a tape job is brought to a control point, the required number of tape units are reserved logically to the job by decreasing the system tape pool count and increasing the control point tape pool count. Specific tape units are not assigned. The time and type of assignment is a user option.

Tape requirements can be minimized by careful planning of job phases. The job card parameter specifies the maximum number of tape units that may be assigned at any one time, not the maximum number of tape requests. Additional tape units may be requested after the job has used its maximum allotment, provided the job releases to the control point tape pool, via UNLOAD, one unit for each additional unit requested. If total tape requirements decrease after the first phases of the job, units should be returned to the system tape pool via RETURN.

TAPE ASSIGNMENT

Use of the LABEL card is recommended for all tape requests except unlabeled input tapes. The LABEL options selected by the user depend on the intended usage of the tape file. Use of the LABEL card permits SCOPE to assign the unit automatically regardless of selected options. Refer to section 8 for a description of the LABEL card.

SCRATCH TAPE FILES

Scratch tape files should be requested as follows:

LABEL, logical file name, W,X=IU,...

An available scratch tape will be assigned automatically and blank-labeled. Use of inhibit unload (X=IU) is optional but recommended, since it will ensure that the tape is not unloaded when the unit is released but is available immediately for automatic assignment to the next job requesting a scratch tape, without requiring an operator type-in.

Other LABEL card parameters may appear if other than the default values for data format, density, etc. are desired.

SAVE OUTPUT TAPE FILES

Save output tape files should be requested as follows:

LABEL,logical file name,W,L =file label name, T= retention, X=SV,...

An available scratch tape will be assigned automatically and labeled according to parameters on the LABEL card. The save parameter is optional but recommended, since it will ensure, when the unit is released, that the tape is unloaded and the operator notified that the tape should be saved.

Other LABEL card parameters may appear if other than the default values for data format, edition number, etc. are desired.

A specific output tape may be assigned by including the VSN parameter on the LABEL card.

LABEL, logical file name,L=file label name,X=SV,VSN=visual reel number,...

INPUT TAPE FILES

Input tape files should be requested as follows:

LABEL,logical file name,R, L=file label name,...

The VSN parameter may be included to restrict automatic assignment to a specific volume.

If the tape can be located by VSN on any unit, that unit will be assigned automatically; otherwise an available unit is assigned automatically and the operator is directed to mount the tape file on that unit.

Other LABEL card parameters may appear if the tape was created with other than default values for data format, density, etc. Enough label field parameters should appear to ensure that the correct tape is located. Any tape label field value will be considered correct if the value for that field is not explicitly declared on the LABEL card.

OTHER TAPE FILES

A REQUEST card/function may be used to request manual or automatic assignment of a tape unit. Manual assignment of an unlabeled input tape is necessary.

A user may request automatic assignment of a labeled input tape by using a REQUEST card/function, but this practice is not recommended. However in this case, the first reference to the file must be an OPEN with automatic recall and FET label fields must be specified.

JOB TERMINATION

When a job has finished processing a tape file, the tape unit should be released. If the number of units reserved to the job can be decreased, the unit should be returned to the system tape pool (RETURN card). Otherwise, the unit may be returned to the control point tape pool, thereby maintaining the number of units logically reserved to the job (UNLOAD card). In either case, the physical unit will become available for assignment to any control point.

When a tape unit is released to either pool or when the job terminates, a trailer label will be written, if appropriate, and the tape will be positioned according to the following criteria:

If inhibit-unload was declared, the tape will be rewound.

If save was declared, the tape will be rewound/unloaded and the operator requested to save the tape.

If no declaration was made and the tape is released prior to job termination, the tape will be rewound/unloaded.

If no declaration was made and the tape is released as a result of job termination, the tape will not be positioned.

PROGRAM EXECUTION

A user can call a program for execution by using a control card on which is punched the name of the file containing the program. When such a card is processed, SCOPE searches a table containing the names of all user-created files assigned to the job. When the specified file name is found, the file is rewound and the program on the file is loaded into central memory and executed. If the file is not located, however, SCOPE searches the system library for a SCOPE file with that file name. Again, if the file is found, the program on that file is loaded and executed. If the file cannot be located, or if the file is found but does not contain an executable program, the job is terminated.

Programs created by users and programs that are part of SCOPE are both referenced by the program execution card, punched in the following format:

lfn,list.

On this card, lfn is the name of the file containing the program to be executed. Only this entry is required on the card. As with any logical file name, lfn must be 1 to 7 alphanumeric characters, beginning with a letter and followed by any proper terminator if no parameter list follows.

The optional entry, list, is composed of parameters that depend on the program to be executed and are referenced by that program. A separator follows lfn and each succeeding item in the list. The last parameter must be followed by a proper terminator.

Programs created by the user are loaded and executed with loader control cards in the above format, discussed in Section 6. SCOPE programs are loaded and executed with SCOPE control cards, punched in the same format. Among the most frequently used SCOPE control cards are those used to execute the COMPASS assembler or the source-language compilers contained on the SCOPE library.

The following table shows the names that should be used for lfn on the program execution card to assemble or compile a user program:

Source Language	**lfn**
FORTRAN Extended	FTN.
FORTRAN (RUN)	RUN.
COBOL	COBOL.
ALGOL	ALGOL.
COMPASS	COMPASS.
SIMSCRIPT	SIMS.
SIMULA	SIMULA.
SORT/MERGE	SORTMRG.
PERT/TIME	PERT66.
APT	APT.
BASIC	BASIC.

The following program call card would be used to assemble a program in COMPASS:

```
COMPASS.
```

On cards requesting assembly or compilation, the list parameters are used for such functions as:

Naming the file onto which the program is to be translated in object code

Naming the file on which the program to be assembled or compiled is originally stored

Producing source-language or object code listings of the program

Punching the file on binary cards

The following card requests compilation of a FORTRAN Extended program from a file called STANLEY onto a file named OLIVER.

```
FTN(I=STANLEY,B=OLIVER)
```

The list parameters associated with FORTRAN Extended, and with all other source languages are described in detail in the reference manuals covering these languages.

After compilation, loading and execution are normally requested. These functions can be performed individually or jointly with the loader control cards described in Section 6.

In the following job, a COBOL program is compiled, loaded, and executed:

Card	Function
`JOB7,MT1.`	Names job.
`REQUEST(FILE,MT)`	Requests input file FILE on tape for job.
`COBOL.`	Compiles program (onto file LGO).
`LGO.`	Loads and executes program.
`7/8/9`	Signals EOR.
`(COBOL program)`	
`6/7/8/9`	Signals EOF.

PROGRAM OPTIONS

SETTING PROGRAM SWITCHES (SWITCH CARD)

In program branching, where two alternate processing routes are provided, the software sense switch is frequently used to determine which path is taken. This switch is a bit in central memory that a user's program can reference.

The program might contain a request to take one path if the bit is on, (set to 1) and another if it is off (reset to zero).

Up to six switches can be set or reset in a source program. At the start of every job, all switches are reset to zero; each can be set or reset by the presence of SWITCH cards in the control card record:

SWITCH,n.

The parameter n must be given. It specifies the number (1-6) of the switch to be manipulated. The following control card could be used initially to turn on switch 4.

```
SWITCH,4.
```

A switch is turned off when the job terminates; it can be reset to zero prior to job termination by including a second switch card referencing the same switch. Thus, a second SWITCH,4. card would reset switch 4 to zero. A third SWITCH,4. card would again set it to one.

As an example of how a switch is used, consider the following job that contains a program written in FORTRAN Extended. This program is executed once a week with different data each time. Within the program, a routine called TALLY is executed only every fourth week. The program includes an instruction to branch to TALLY only if switch 1 is on. When it is the week for TALLY, a SWITCH,1. card is included in the control card record as shown. When the program reaches the instruction GO TO (100,200)J, the activated switch will cause a branch to call for the routine TALLY. When executed, the final instruction within TALLY will be a return jump to statement 100 in the main program.

```
                      Card                     Function

                    / JOB8,MT1.                Names job.
                    |
                    | REQUEST,TAPE,MT.         Requests an input tape
                    |                          file named TAPE for job.
Control Card Record < REWIND,TAPE.             Rewinds file TAPE.
                    | SWITCH,1.                Sets switch 1 on.
                    | FTN.                     Compiles program.
                    | LGO.                     Loads and executes job.
                    \ 7/8/9                    Signals EOR.
                    /    PROGRAM ALPHA (INPUT,OUTPUT,TAPE)
                    |      .
                    |      .
                    |      .
                    |    CALL SSWTCH(1,J)      Relates switch 1 to J. If
                    |                          switch 1 is on, J is set
                    |                          to 1; if off, J is set to
                    |                          2.
                    |
                    |    GO TO (200,100)J      Transfer to 100 if
 FORTRAN Extended   <                          switch 1 is off;
 Program            |                          transfer to 200 if
                    |                          switch 1 is on.
                    | 200 CALL TALLY           Calls and executes
                    |                          routine TALLY.
                    | 100                      Continue execution of
                    |                          program.
                    |     .
                    |     .
                    |     .
                    \ 7/8/9
               Data { (Data)
                    { 6/7/8/9                  Signals EOF.
```

If TALLY is not to be executed, the SWITCH,1. card is omitted from the control card record. When the FORTRAN program reaches the branching statement, it will determine that switch 1 is off and transfer to statement 100.

Switch reference instructions within a user program vary, depending on the program source language. They are described in the reference manuals covering each language.

ESTABLISHING HALT CONDITIONS (MODE CARD)

Among the various types of errors that can cause a job to terminate prematurely, or to branch to an exit path specified by the user, three can be negated so that program processing will continue in spite of the error. These errors are:

A reference to an operand (any number used in a calculation) that has an infinite value

A reference to an address outside the field length of the job in central memory or ECS

A reference to an operand to be used in floating-point arithmetic for which the decimal point was not defined

Normally, these errors will terminate processing; any or all can be suspended as halt conditions, so that processing continues until another type of error is encountered that terminates the job, or until all control cards are executed. The MODE card, punched in the following format, is used for this purpose.

MODE,n.

The n parameter is a number specifying the halt conditions to remain in effect for a job:

To Halt:	**Enter for n:**
In none of the three negatable cases	0
Only if address is out of range	1
Only if operand is infinite	2
If address is out of range or operand is infinite	3
Only if operand is floating point number with undefined point	4
If address is out of range or operand is floating-point number with undefined point	5
If operand is infinite or operand is floating-point number with undefined point	6
If operand is infinite or operand is floating-point number with undefined point or address is out of range	7

The following card will permit processing to continue if a referenced address is out of range of the field allotted to the job in central memory; processing will halt, however, if an infinite operand or a floating-point operand with an undefined point is referenced:

MODE,6.

Any MODE request that permits processing to continue regardless of a reference to an out-of-range address should be used only with great caution. Resulting output probably will have no value. Under such conditions, an attempt to write outside FL appears to complete normally; however, no writing is done. When an attempt is made to read outside FL, zero is returned to the X register; no information is returned.

At most installations if no MODE card appears in a program, mode 7, which allows a job to halt if any error occurs, is always in effect; this is the normal operating mode.

A mode request remains in effect until a new mode request is encountered or until the current job is processed. When processing is complete, SCOPE returns to the normal operating mode for the installation.

ESTABLISHING EXIT PATHS (EXIT CARD)

Normally, when a fatal error occurs in a job and it is not a negatable error suspended by a MODE request, processing is terminated, a diagnostic message is issued, and output created prior to the error is output. However, special exit routines can be established within the control card record to which the job can branch in the event of certain kinds of errors. The exit routines will be executed before the job is terminated. Such an exit routine might direct the computer to dump the central memory contents of the job, or it might direct execution of an entirely different program.

Certain conditions cause abrupt termination of a job regardless of an exit routine:

A request from SCOPE or the computer operator to terminate the job and inhibit all output

A request from the operator to transfer the job from central memory back into the input queue

An error on the job card

A checksum error encountered during the job input

When other types of errors occur, SCOPE searches the control card record for an exit routine which is executed if found. The following or terminating conditions will result in this search:

The job uses all execution time allotted.

An arithmetic error occurs (the type negatable by a MODE card).

A peripheral processor encounters an improper input/output request.

A central processor program requests job termination.

The operator requests that the job be dropped.

A control card error (other than on the job card) occurs.

An ECS parity error occurs.

If a control card record includes an exit routine but no error occurs, the job will terminate as it would if an EOR card had been encountered in the control card record.

Exit routines are established with the EXIT control card:

EXIT.

An error causes the job to be advanced until either an EOR (7/8/9 card) or an EXIT card is encountered in the control card record. When an EXIT card is encountered, the cards that immediately follow it are executed, as illustrated below:

Card	**Function**
`MYJOB,P1,T400,CM50000,MT1.`	Names job.
`REQUEST,MYFILE,MT.`	Requests input tape file MYFILE.
`RUN.`	Compiles and executes FORTRAN program.
`EXIT.`	Signals beginning of exit routine.
`DMP,1000.`	Dumps first 1000(octal) words of storage.
`7/8/9`	
`(FORTRAN (RUN) program)`	
`7/8/9`	
`(Data)`	
`6/7/8/9`	

In this job, the two dumps requested will be produced only if an error occurs, causing the job to branch to the exit routine.

Errors in control card format, or an attempt to load an object program resulting from erroneous assembly or compilation, result in instant termination of the job even if an EXIT card and routine are present. This action prevents indiscriminate dumping of large loading and compilation routines in cases where an EXIT card is followed by a dump request. To override the instant termination procedure and enter an exit routine regardless of these types of errors, the suffix(S) should be added to the EXIT card:

EXIT(S)

INTERRELATING DEPENDENT JOBS (TRANSF CARD)

The user can submit a string of interdependent jobs to the computer, specifying the order in which they are to be executed. In such a string, jobs can be input in any order and through one or more card readers. A job will not be executed until all prerequisite jobs in the string have been executed. Whenever possible, SCOPE schedules interdependent jobs for execution in parallel (multiprogramming).

When each job is input, the dependency identifier and dependency count on the job card are read into the SCOPE file name table along with other information about the job. Each time a prerequisite job executes a transfer to a dependent job, the dependency count is decremented by one. When the count becomes zero, the dependent job can be executed.

On the job card, the Dym parameter establishes job interdependency. y is the dependency identifier that names the string to which the job belongs. m is the dependency count (number) of prerequisite jobs on which the job depends.

Transfer from a prerequisite job to a dependent job is accomplished by the TRANSF card, placed in the control card record after the card that executes the prerequisite job. The TRANSF card is punched in this format:

TRANSF(p1,p2,...pn)

The p parameter names the jobs to which transfer is directed; only the first five characters of each job name are examined by SCOPE. As many job names as will fit on a card can be noted. Several TRANSF cards can be included in each job, but none should appear in the last job in a string.

An example of an interdependent job string follows. Jobs B and C cannot run until Job A has been executed. Job D must wait until Jobs B and C have been run. Job E must wait only on Job C. Before Job F can be run, Jobs B, D, and E all must have been executed.

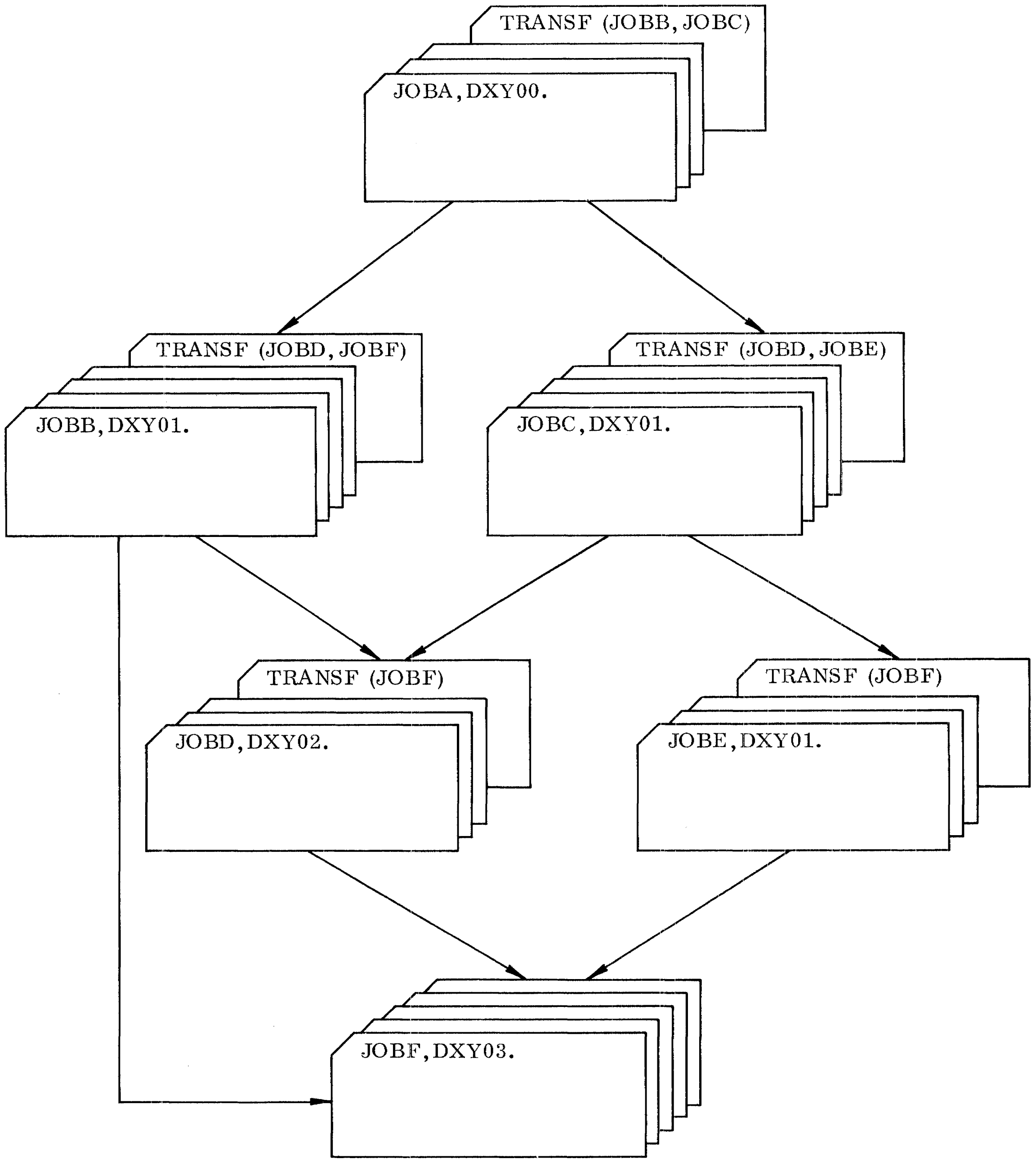


Figure 2-3. Interdependent Jobs

If a job containing a TRANSF card is terminated before that card is processed, control will not pass to the next job in the string. Instead, all succeeding jobs that depend on this job will remain in the input queue. They can be executed only if the prerequisite job is re-run successfully. Otherwise, the entire string should be terminated by the operator. Since no error message appears in this event, the user should inform the operator what jobs belong to an interdependent string.

INSERTING COMMENTS IN THE JOB (COMMENT CARD)

Informal comments, remarks or messages to the computer operator can be inserted after the terminator on any control card. These comments appear in the printed job dayfile, and also in the dayfile display for the operator on the console screens. In the following control card record, examples of such comments are presented.

```
Control Statement               Comment

JOBSAM,T500,CM50000,MT1.        WORK ORDER NO. 2126A.
REQUEST,TAPE1,MT.               THIS IS TAPE NO. 112.
REWIND,TAPE1.
FTN.                            THIS IS NEW COMPILER
LGO.
7/8/9
(FORTRAN program)
6/7/8/9
```

Comments can also be inserted independently of SCOPE requests by punching them on COMMENT cards, in this format:

COMMENT.n....nn

Comments or remarks (n...n) are inserted after the period following the word COMMENT. There may be up to 72 characters which can occupy any column, 9 through 80. Any character can be used, including blanks, commas, periods, and other punctuation. Like the comments on other SCOPE control cards, those on the COMMENT card are printed in the job dayfile and displayed to the operator on the console screen. Only the characters in columns 9 through 80 appear, however.

Since the comment card requires no action by the computer operator, and job processing is not suspended for pending action, the computer operator may not notice all messages because of the speed at which this card is processed. It is better to place such messages on any relevant control card, if possible.

If a comment is too long for one COMMENT card, it can be continued on as many COMMENT cards as necessary, as shown in this example:

```
job card.
COMMENT. THIS JOB CALCULATES THE SPECIFIC IMPULSE DERIVED FROM THE
COMMENT. E-G INJECTOR, UNBAFFLED VERSION, USED WITH GAS
COMMENT. GENERATOR 1117A, ON THRUST CHAMBER YLR2776A
COMMENT. FOR ROCKET ENGINE J-6.
FTN.
LGO.
7/8/9
(FORTRAN Program)
7/8/9
(Data)
6/7/8/9
```

EXAMPLES OF JOB DECK ARRANGEMENTS

The order in which SCOPE control cards are arranged within the input stream (control card record) depends upon the purpose of the job and the programs it contains. The following examples illustrate typical arrangements:

JOBA requests a tape file named SALLY, and loads and executes an object program from that file:

```
JOBA,MT1.
REQUEST,SALLY,MT.
SALLY.
6/7/8/9
```

JOBB, containing a FORTRAN Extended program on Hollerith cards, compiles, loads and executes that program.

```
JOBB.
FTN.
LGO.
7/8/9
(FORTRAN Extended Program)
6/7/8/9
```

JOBC, containing a program on binary cards, loads and executes that program:

```
JOBC,CM43000,T500.
INPUT.
7/8/9
(Program on Binary Cards)
6/7/8/9
```

JOBD compiles, loads and executes an ALGOL program named SMITH, punched on Hollerith cards; loads and executes a program named JONES, punched on binary cards; and loads and executes a program named BROWN on a common file stored on disk.

```
JOBD.
ALGOL.
LGO.
INPUT.
COMMON,BROWN.
BROWN.
7/8/9
(Program SMITH)
7/8/9
(Program JONES)
6/7/8/9
```

JOBE compiles and executes a FORTRAN Extended program and executes this program with one set of data, and then with another:

```
JOBE.
FTN.
LGO.
REWIND,LGO.
LGO.
7/8/9
(FORTRAN Extended Program)
7/8/9
(First Data Deck)
7/8/9
(Second Data Deck)
6/7/8/9
```

JOB TERMINATION

NORMAL TERMINATION

When a job is processed without error, normal termination activity begins when an end-of-record (7/8/9) card or an EXIT card is encountered in the control card record. First, the execution time of the job is written onto the job dayfile, DFILEx and on the system dayfile, DAYFILE. Then, DFILEx is rewound and copied onto the file OUTPUT. Next, OUTPUT and any other files on mass storage designated for output, such as PUNCH or PUNCHB, are rewound and placed in the output queue on disk. OUTPUT is designated for the printer, and PUNCH (Hollerith) and PUNCHB (binary) for the card punch by disposition codes. These file names are then changed to the job name, the type to output, and the assignment to control point 0.

Files on magnetic tape are assigned to control point 0, rewound, unloaded (if SAVE status has been requested), and released from the system. Common files are assigned to control point 0, where they can be assigned to other jobs. Permanent files are released from the job. All remaining local files in central memory and mass storage, including INPUT, LGO, and DFILEx, are cleared and released. The job is then released from the control point area.

All hardware devices assigned to the job are assigned to control point 0, so that they can be reassigned to other jobs.

At this point, files in the output queue relating to the job are all that remains of the job. When an output device of the type requested by the file's disposition code is free, the file will be output through that device. If no device has been requested, the file is assigned to the printer. Where applicable, the job output is arranged in the following order:

Source language listing

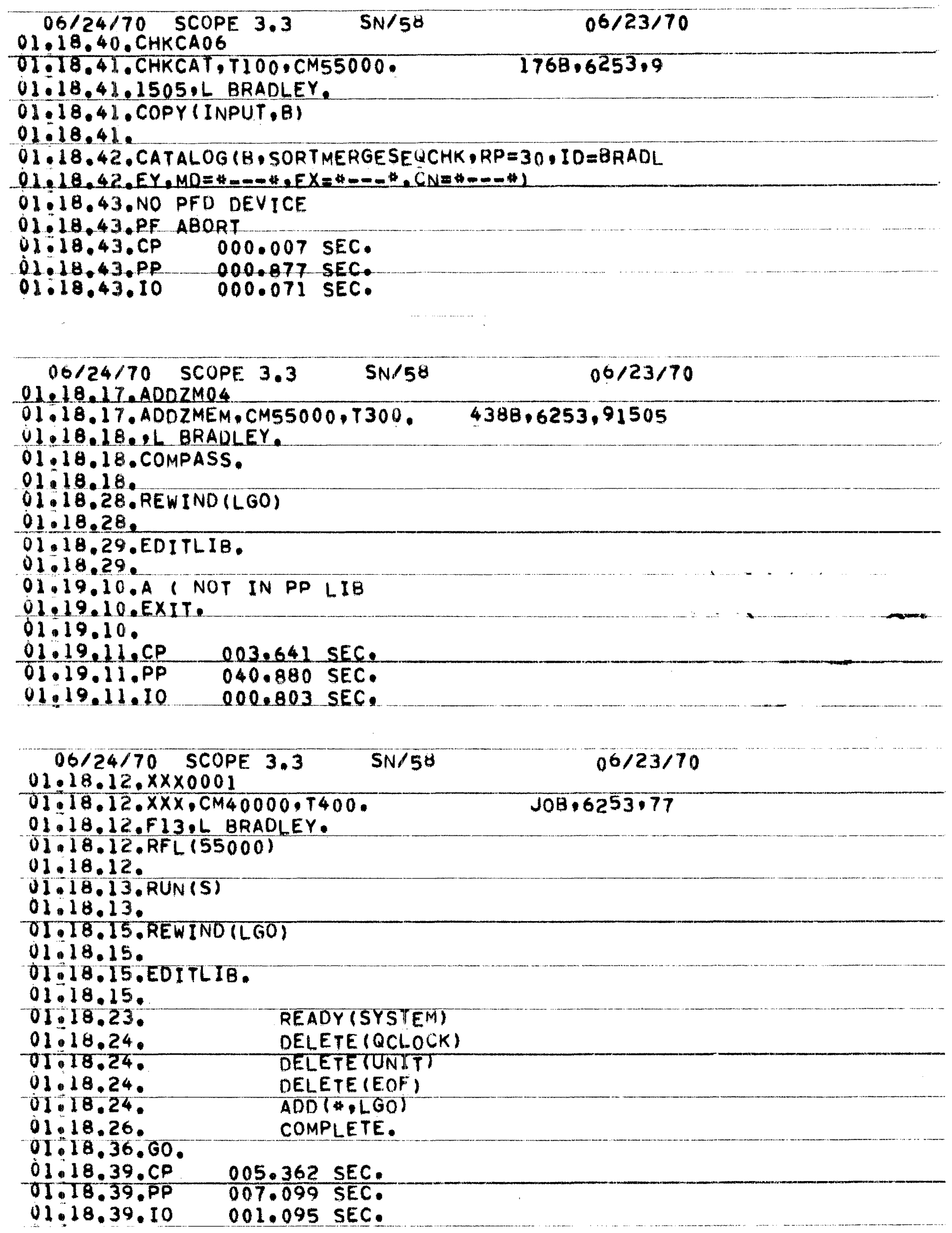
Object listing

Load map

Executed program output (results)

Job dayfile

The dayfile shows all control cards executed, equipment assigned to the job, the total central processor and peripheral processor time, system action reports, and the date and time of day each processing event took place. An example of a dayfile is shown in figure 2-4.



```
  06/24/70  SCOPE 3.3      SN/58                06/23/70
01.18.40.CHKCA06
01.18.41.CHKCAT,T100,CM55000.             1768,6253,9
01.18.41.1505,L BRADLEY.
01.18.41.COPY(INPUT,B)
01.18.41.
01.18.42.CATALOG(B,SORTMERGESEQCHK,RP=30,ID=BRADL
01.18.42.EY,MD=*---*,FX=*---*,CN=*---*)
01.18.43.NO PFD DEVICE
01.18.43.PF ABORT
01.18.43.CP      000.007 SEC.
01.18.43.PP      000.877 SEC.
01.18.43.IO      000.071 SEC.

  06/24/70  SCOPE 3.3      SN/58                06/23/70
01.18.17.ADDZM04
01.18.17.ADDZMEM,CM55000,T300.       4388,6253,91505
01.18.18.,L BRADLEY.
01.18.18.COMPASS.
01.18.18.
01.18.28.REWIND(LGO)
01.18.28.
01.18.29.EDITLIB.
01.18.29.
01.19.10.A ( NOT IN PP LIB
01.19.10.EXIT.
01.19.10.
01.19.11.CP      003.641 SEC.
01.19.11.PP      040.880 SEC.
01.19.11.IO      000.803 SEC.

  06/24/70  SCOPE 3.3      SN/58                06/23/70
01.18.12.XXX0001
01.18.12.XXX,CM40000,T400.                JOB,6253,77
01.18.12.F13,L BRADLEY.
01.18.12.RFL(55000)
01.18.12.
01.18.13.RUN(S)
01.18.13.
01.18.15.REWIND(LGO)
01.18.15.
01.18.15.EDITLIB.
01.18.15.
01.18.23.              READY(SYSTEM)
01.18.24.              DELETE(QCLOCK)
01.18.24.              DELETE(UNIT)
01.18.24.              DELETE(EOF)
01.18.24.              ADD(*,LGO)
01.18.26.              COMPLETE.
01.18.36.GO.
01.18.39.CP      005.362 SEC.
01.18.39.PP      007.099 SEC.
01.18.39.IO      001.095 SEC.
```

Figure 2-4. Job Dayfile

ABNORMAL TERMINATION

When an error occurs, SCOPE sets a flag indicating the error. If the error causes abrupt termination with no exit path, SCOPE begins termination processing. If the error permits branching to an exit path, SCOPE searches the control card record for an EXIT card. If such a card is found, SCOPE clears the error flag and executes the control cards following the EXIT card. If no EXIT card is found, SCOPE ignores the remainder of the control cards and begins termination processing.

As the first step in termination processing, diagnostic messages are written in DFILEx and DAYFILE. Also, the current contents of the SCOPE exchange package, the first 100(octal) words in the job field length, and 100(octal) words preceding and following the program address are copied onto DFILEx; this is the standard error dump. Next, the procedures described under Normal Termination are begun. In addition to the items output under normal termination, the error dump will also be printed.

REGAINING CONTROL BEFORE JOB TERMINATION

The RECOVR utility function, initialized by a system macro, allows a user to regain control of the central processor before a program terminates. Control can be regained if the program executes successfully or if a variety of error conditions caused an error flag to be set. RECOVR makes the exchange jump package and RA+1 contents available to the program, if user recovery code is executed, and gives the user the option of normal or abnormal job termination output. At least five seconds of central processor time always will be available for user code execution.

Unlike the OWNCODE capability where the user gains control only after file action request errors, RECOVR covers a wide range of error conditions, as shown below, with the octal value that will select them, in the call to RECOVR:

Arithmetic mode error	001
PP call or auto- recall error	002
Time or field length exceeded	004
Operator drop or rerun	010
System abort	020
CP abort	040
Normal termination	100

Conditions can be combined as desired; octal values up to 177 are allowed in the flag field of the call to RECOVR.

Initializing RECOVR at the beginning of a program results in an entry in a stack of requests for PP program RPV. Although RPV can be called directly by a monitor request in RA+1,the RECOVR utility is preferable for all except stand alone system utilities because SCOPE routines also use this capability. Only one set of recovery condition flags can exist within RPV, but RECOVR allows up to five user and system sets of flags and code for each program. The last RECOVR code initialization will receive control first if conditions flagged for recovery occur.

If a program calling RECOVR contains overlays or segments, both the call to RECOVR and the user recovery code should be a part of the level 0.0 code.

The exchange jump package will be returned in its normal form, with the system error code that caused recovery code execution in bits 0-17 of the first word. If the P register field in the exchange package shows zero because of a mode error, bits 31-47 of RA+0 will contain the P register value.

System error codes are:

Normal termination	0
Requested time limit exceeded	1
Arithmetic mode error	2
Illegal parameter passed to PP	3
CP program requested abort	4
PP program not in library	5
Operator dropped control point	6
Operator initiated rerun of job	8
ECS parity error	10
Required auto-recall status missing	13
Job hung in auto-recall	14
Requested mass storage limit exceeded	15

A checksum of the user recovery code can be requested during initialization. If flagged conditions subsequently occur, RECOVR will again checksum the code before returning control to it, giving some assurance of user code integrity before execution.

COMPASS Call to RECOVR

RECOVR is called from a COMPASS program with:

RECOVR name,flags,checksum

name	Address of code to be executed if flagged conditions occur. A return jump will be made to this location.
flags	Octal value for conditions under which recover code is to be executed, as outlined above; default is 77.
checksum	Last word address of recover code to be checksummed;0 if no checksum.

If flagged condition occurs, the address of the exchange package will be in register B1 and the RA address in B3. Register A1 will contain the address of the list of parameters passed in B1-B3. Register B2 will contain a 0; if the recovery code sets it to non-zero, or if the code contains an ENDRUN macro or an RA+1 request for END, normal job execution resumes.

FORTRAN and FORTRAN Extended calls to RECOVR

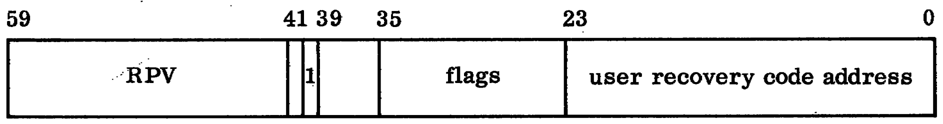
During program initialization, the RECOVR routine must be called with three arguments to establish recovery conditions:

CALL RECOVR (subroutine, flags, checksum)

subroutine	Name for the system to call after a flagged condition occurs. This subroutine must have three arguments: Name of 17 word array to receive contents of the exchange jump package plus word RA+1. Endrun indicator. If user code sets this flag to non-zero, abnormal job termination does not occur. Name of variable to receive contents of word RA+0.
flags	Value indicating conditions for executing recovery subroutine.
checksum	0 for no checksum, or last word address plus 1 of subroutine to be checksummed. Address can be provided by a dummy subroutine containing no code and the LOCF library function which returns the address of the named variable. For example, this argument can be LOCF(DUMMY) where a subroutine containing only the statements SUBROUTINE DUMMY and END immediately follows the subroutine named in the first argument of RECOVR.

Calling RPV

The PP program can be called by establishing RA+1 as follows:



The code at the recovery address should allow for a 21-word array (octal) to be returned. Control will be returned to word 22 if recovery code is executed.

An optional checksum of the recovery area can be requested in the user call. If the word at the recovery address contains all zeros, no checksum will be taken. If the upper 30 bits contain the last word address of the recovery area, a checksum of the recovery area+18 through last word address will be made and stored at the recovery address+1.

FILES 3

In the SCOPE system, except for some central memory tables, all information defined to the system is considered to be either a file or a part of a file. This section describes the concepts and terminology underlying the file organization and the structuring of data which SCOPE can process. SCOPE activities related to the creation, processing and disposition of files are also included.

FORMATS

Local to a given job, a file is identified by a logical file name. This name consists of one to seven alphanumeric characters, the first of which must be a letter. All control card references to a file identify it by the logical file name. The internal central memory representation of a logical file name consists of its literal value in display code, left justified, and zero-filled in bits 59 to 18 of the central memory data word. Reference to a file, using one of the SCOPE macros defined to COMPASS, always symbolically addresses a parameter table, the first word of which contains an internal representation of the logical file name.

All information in the SCOPE system is stored on files which may be allocatable or non-allocatable. Files stored on disks or drums, devices which can hold many files and be allocated to many control points at the same time, are allocatable files. All other files are non-allocatable including magnetic tape files, card files, and private disk packs.

LOGICAL RECORDS

All files within the SCOPE system, regardless of type, are organized into logical records: for input files, through the use of (7/8/9) cards; for output files, through the language translator or other program producing the output.

Since the logical record concept is defined for all devices, files may be transferred between devices without losing their structure. The physical format of a logical record is determined by the device on which the file resides. The physical record unit size (PRU) is the smallest amount of information that may be transferred during a single physical read or write operation for each device. Logical records are written as one or more PRU's, the last of which is short or zero-length. A zero-length PRU is written if the logical record is an even multiple of the PRU size or if a write operation was requested with no data in the buffer. A zero-length PRU contains no data, but it has control information (see Level Numbers).

Coded files on 1/2-inch magnetic tape receive special treatment. Within the SCOPE system, all coded information is carried in display code; therefore, a conversion to external BCD for 7-track tapes (ANSI or EBCDIC for 9-track tape) must be made before writing on the tape. Translation is character-for-character.

For SCOPE tapes, the display code end-of-line mark (12-bit zero byte) is converted to the external BCD code 1632(octal). The display code end-of-line mark is recognized only when it appears in the lower 12 bits of a central memory word.

LEVEL NUMBERS

Related logical records within a file may be grouped by the user into an organized hierarchy. The level number (0-17(octal)) of a logical record is contained in the short or zero-length PRU which terminates the record. This PRU is the level mark. The level number is declared in the write request. If no number is specified, a level of 0 is assigned. If, when no data is in the buffer, a level number is specified in a write request, a zero-length PRU containing the level number is written.

number is written. A write end-of-file request causes a zero-length PRU of level 17 (logical end-of-file mark) to be written. The level mark appended to each logical record is not placed in the circular buffer when the file is read; but it is returned as part of the status information. Level number 16 should not be used for a job which includes a request for a checkpoint dump as this level number is used in a unique way by the checkpoint dump program.

The lowest level within a file is associated with a single logical record. A higher level defines a set of records consisting of the logical record at that level plus all preceding records at a lower level.

For instance, a file might be regarded as a multi-volume book; level 0 would be equivalent to a page, level 1 to a chapter, and level 2 to a volume. In the following example, the lowest level 0 is associated with a single logical record called a page; level 1 marks delimit a group of pages called chapters; chapters are grouped by level 2 marks into volumes. A reference to a logical record of level 1 includes all information between the referenced level 1 mark and the succeeding one. Included, therefore, will be several logical records as shown in the diagram.

Logical Record	Level Mark	Page	Chapter	Volume
1	0	1		
2	1	2	1	
3	0	3		I
4	0	4	2	
5	2	5		
6	0	6		
7	0	7	3	
8	1	8		
9	0	9		
10	0	10		
11	0	11	4	II
12	1	12		
13	0	13		
14	2	14	5	
15	0	15		
16	1	16	6	
17	0	17		III
18	0	18	7	
19	2	19		
End of Information				

The format of the level mark varies depending on the device type on which the file resides.

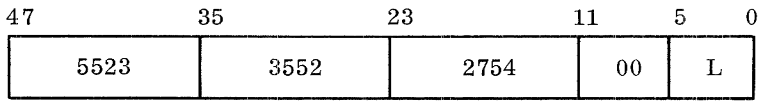
CARD FILES

Each logical record is terminated by a card with 7,8,9 punches in column 1. Columns 2 and 3 may have an octal integer, 00-17, to denote level number. Level zero is assumed in the absence of punches in columns 2 and 3.

The end of information is signaled by a card with 6,7,8,9 punches in column 1. A card with 7,8,9 punches in column 1 and 1,7 punches in columns 2 and 3, is treated as an end-of-information card. For card files, EOI and EOF are synonymous.

BINARY MODE 1/2-INCH MAGNETIC TAPE FILES (SCOPE STANDARD)

Each logical record is terminated by 8 characters (48 bits) as follows:

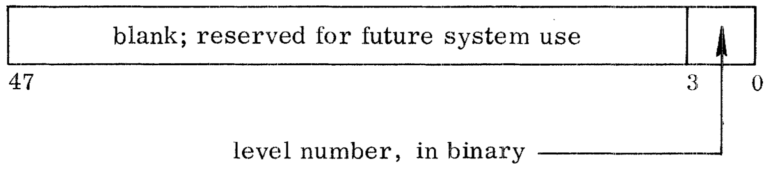


where the 4-bit level number is right justified in the L field.

If the last information in the logical record does not fit exactly into a physical record unit, the 8-character marker is appended to the last written PRU; otherwise, the marker is written as a single PRU of zero length.

CODED MODE 1/2-INCH MAGNETIC TAPE FILES (SCOPE STANDARD)

Each logical record is terminated by 8 characters as follows:



The level number is the low-order 4 bits of the last character. The upper 2 bits of this character are always zero, except for level zero which is represented by 010000 (binary). For example, level five would be represented by 2020202020202005 in external BCD. Level zero would be represented by 2020202020202020 in external BCD. If the last information in the logical record does not fit exactly into a physical record unit, the 8-character marker is appended to the last written PRU; otherwise, the marker is written as a single zero-length PRU.

Level numbers are not supported for 9-track tapes.

ACTIVE FILES

SCOPE is a file-oriented system: all information contained within the system is considered to be either a file or part of a file. Active files–those immediately available to the system at any moment–are defined to be any of the following:

Each job file waiting to be run. This set of files is called the job stack or input queue.

Output files from jobs which have been run and are waiting to be disposed of by printing, punching, etc.

Job files presently in some state of execution.

Files currently being used by the jobs in execution.

Common files which maintain active status by specific request.

Permanent files attached to a job.

SCOPE maintains a file name table (FNT) in central memory resident. This table contains one 3-word entry for each active file in the system. The first word identifies the file and contains other information about it. The second and third words, which describe its status, are sometimes called a file status table or FST. When the user references a file with a REQUEST control card or macro call, or when he issues an I/O request referencing a file that does not exist, the SCOPE system creates an FNT entry for the file and assigns the file to a device. Thereafter, each time a user makes an I/O request, file status information is transmitted between the user's FET (file environment table) and the file name/status table.

The four types of active files are: input, local, output, and common. When a permanent file is attached to a job, it becomes a special kind of local file. Private disk pack files and random files are described in this section under File Processing.

As a job progresses, the job file goes through several type changes. When a job file is read from the card reader, it is copied onto mass storage and becomes an input file; it is not assigned to any control point. The file name is that given on the job control card. The file name/status table contains a priority (from the control card) for the file which becomes the priority for the job.

When the job is assigned to a control point, the input file becomes a local file; and its name is changed to INPUT. The original name of the input file is saved in a word of the control point as the name of the job. New local files named OUTPUT, PUNCH, and PUNCHB will be established, if referenced, and given disposition codes of print, punch coded and punch binary, respectively.

INPUT, OUTPUT, PUNCH and PUNCHB are all local files on mass storage. They are the immediate source of card input and the immediate destination of printer output and coded and binary card output. Because several jobs may run concurrently at different control points, several local files called INPUT, OUTPUT, PUNCH, and PUNCHB may be in the file name/status table simultaneously. When a local file is sought in the table, both the name and the control point number are used to identify it.

When a job terminates, the local file called INPUT for the assigned control point is released. Entries in the file name/status table for the local files called OUTPUT, PUNCH, and PUNCHB for that control point are altered so that their names are changed to the name of the job itself, which is found in the control point area. The control point is then released.

Other local files can be created by the job. For instance, the first time a job references a file called RASP, the system consults the file name/status table entries for a local file of that name assigned to the job's control point. If one does not exist, a file is immediately created, initially consisting only of an end-of-information mark. This file is named RASP and entered into the file name/status table as a local file assigned to that control point. When the job terminates, all local files created in this manner are eliminated completely from the system.

The common file is a local file for which active status is maintained by a control card request, so that the file does not disappear when the job originating it is terminated. Each common file must have a name unique among all common files regardless of control point.

Example:

A job contains the control statement:

```
COMMON,RASP.
```

If the job generates a local file called RASP, that file does not disappear when the job terminates. The entry in the file name/status table for the local file RASP is altered so that it no longer belongs to any control point, and its type will be common. If RASP is assigned to a private disk pack, however, it will be preserved on the disk pack when the job terminates and the COMMON card will have no effect.

An attempt to declare a permanent file COMMON is illegal but not fatal to job execution.

It is assumed that the file name/status table did not already contain an entry for a common file called RASP. However, if it did contain such an entry, when a job is processed that contains the control statement COMMON RASP., file RASP would be assigned to the control point of that job. RASP would then be available to that job just as if it were a local file.

If a third job contained the control statement COMMON RASP. and if, when this card was processed, it was found that the common file RASP had been assigned to the control point of a running job, the earlier job would have to terminate and file RASP be detached from its control point before RASP would be available to the latest job.

To eliminate a common file like RASP from the system, a job must contain the control statement COMMON RASP. and a later control statement:

```
RELEASE RASP.
```

When the latter control statement is processed, RASP is converted from a common file to a local file, but not otherwise altered. When the job is terminated, the local file RASP is destroyed.

FILE NAME TABLE

The File Name Table (FNT) is a system table containing a three- word entry for every active file. It provides a link between the user's FET and the system input/output routines. The FNT is protected from user access; it resides in low core and is outside the field length of user jobs.

Status information is stored in the last two words of each FNT entry; these words are often referred to as the file status table (FST) entries. The first word of the FST is the second word of the FNT; there is no separate FST. The file name table is sometimes referred to as the FNT/FST.

The following information is contained in each FNT entry.

FILE NAME

If the file is created by REQUEST or by Circular Buffer I/O (CIO) calls, the file name must be 1-7 alphanumeric characters beginning with a letter; it cannot include embedded blanks. Otherwise, the name may be any 42-bit quantity in which the high-order 12 bits are not all zeros.

FILE TYPE

A number which identifies the file type: input, output, local, or common.

CONTROL POINT NUMBER

The control point to which the file is assigned. If a file is associated with a job running at a control point, it is assigned to that control point. Otherwise it is assigned to control point zero.

Input files are always assigned to control point zero; and each input file must have a unique name. Each file assigned to any control point other than zero must have a name which is unique among files at that control point.

Local files at control point zero need not have unique names. Each common file must have a name unique among all common files regardless of control point.

EQUIPMENT TYPE

A number which specifies the type of device or equipment on which the file resides. This could be a mass storage device such as ECS, drum, disk, or disk pack; or it could be a sequentially accessible equipment such as magnetic tape, line printer, card reader, or card punch. Most mass storage devices are called allocatable, since portions of the device can be allocated to different jobs. Sequential access devices are non-allocatable. A private disk pack is a non-allocatable mass storage device.

The system enters the equipment type in the FNT when the file is created. If the user creates the file with a REQUEST control card or macro, he can specify the type of device. If the file is created when the user issues an input/output function for a non-existent file, the device with the least activity is selected. Each time a user requests an input/output function, the device type and allocation style is set in the FET device type field.

LAST CODE AND STATUS

To perform an I/O operation, the user must set a code in the code and status field of his File Environment Table (FET). If he uses a system macro, this will be done automatically. When the system starts to process the I/O request, it stores the code and status from FET in the File Name Table (FNT). When the I/O operation is complete, the system stores status information in the FNT code and status field, and also sets the completion bit (bit 0 of field). The FNT code and status is then copied to the FET. If an end-of-record or end-of-file status was returned on the last READ, the user must clear these bits before the next READ is issued; otherwise central program control will not honor the next READ action requested.

E/N, INDEX BIT

If the file resides on magnetic tape, the E/N bit is used to determine the tape label action to be performed. If the bit is 1, a label will be written when the file is first referenced; and the tape has new status. Otherwise, the tape is said to have existing status and a label will be read and checked when the file is first referenced. The REQUEST card/function is used to declare existing or new status. The E/N bit is not used if the tape file is unlabeled.

If the file resides on a mass storage device, the E/N bit is called the index bit. It is used to determine whether or not the contents of a random file were altered and therefore whether it is necessary to rewrite the index when the file is closed.

WRITE BIT

The FNT write bit is used to indicate that the file is currently positioned after a newly written record. Operations such as READ are not allowed if the bit is set. Trailer label procedures on tape files must be performed before any backward motion is initiated if the bit is set.

PERMISSIONS

Data transfer operations on a file require that the appropriate permission be granted. All permissions are granted on any non-permanent file. For permanent files, passwords are required to obtain permission.

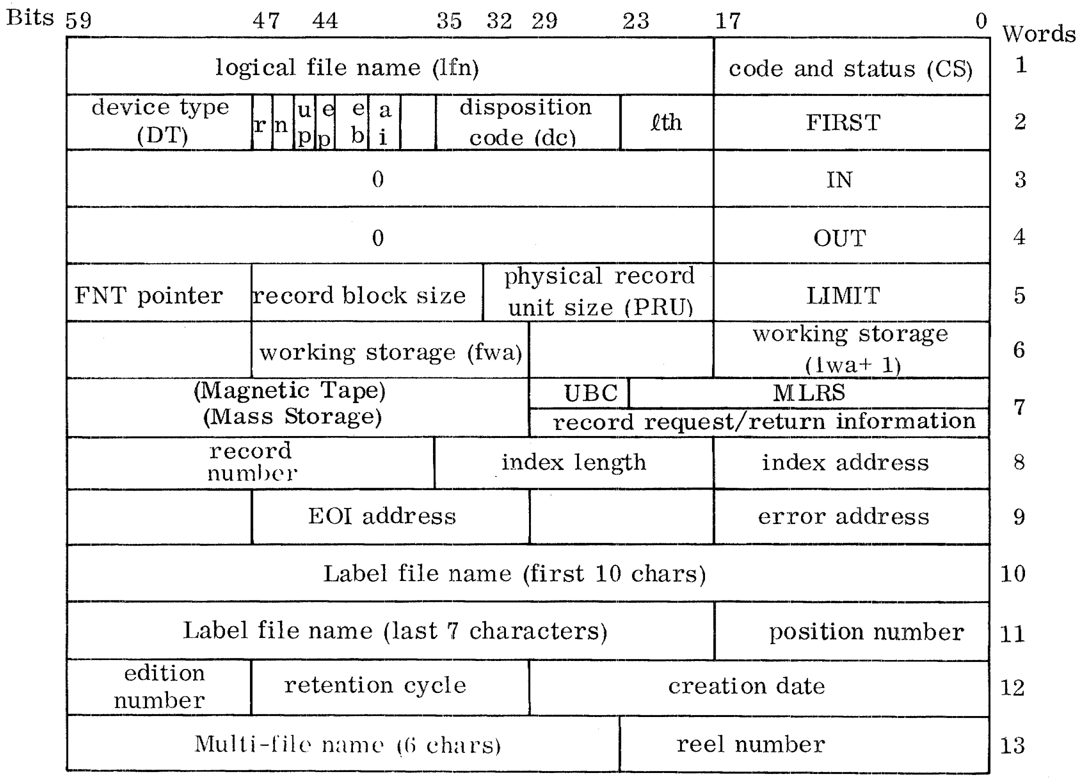
DISPOSITION CODE

This code indicates the action to be taken when it is time to dispose of a file. Mass storage files with non-zero disposition codes are placed in the output queue. Output files created by batch jobs are processed by JANUS. Files submitted by remote terminals are processed by the remote job entry programs. A mass storage file with a special name (OUTPUT, PUNCHB) automatically is assigned a disposition code when it is created. The DISPOSE card/function provides the only other means of assigning a disposition code to a mass storage file.

Magnetic tape files are positioned according to the value of the disposition code field. Positioning declaration is made on the REQUEST card/function.

FILE ENVIRONMENT TABLE

The file environment table (FET) is a communication area initiated by the user; it is interrogated and updated by the system and the user during file processing. An FET must be declared for each file. The system section of the FET is used by the peripheral processor input/output routines and central program control (CPC) as well as by the user program. A user section may be appended to the system FET to centralize other information pertinent to the file. All FET's reside within the field length of the program. The format of the system FET is shown below.



To facilitate rapid changes of IN and OUT values, bits 18-59 of words 3 and 4 are never used; all other fields not specified are reserved for future system use.

Words 1 through 5 form the minimum length FET; the basic FET for S and L tapes is words 1 through 7. Word 6 is used for blocking/deblocking; words 7 and 8 are required for indexed files. Word 9 is used for user's OWNCODE routines. Words 10 through 13 are present when the LABEL control card or macro is used.

BASIC FILE ENVIRONMENT TABLE

LOGICAL FILE NAME (lfn) (42 bits)

The lfn field contains one to seven alphanumeric display-coded characters starting with a letter, left justified; if less than seven are declared, unused characters are zero-filled. This field is used as a common reference point by the central processor program and the peripheral processor input/output routines.

The lfn parameter declared in an FET creation macro is also used as the location symbol associated with the first word of the FET. A reference to lfn in the file action requests is a reference to the base address of the FET.

CODE AND STATUS (CS) (18 bits)

The CS field is used for communication of requested functions and resulting status between the central processor program and the peripheral processor input/output routines. This field is set to the request code by CPC when a request is encountered for this file. The request codes are defined in the file action request descriptions. The code and status bits have the following significance:

Bits 14-17	Record level number. On skip and write record requests, this subfield is set by CPC as part of the function code. On read requests, it is set by CIO as part of the status when an end-of-record is read. Initially the level subfield is set to zero when the FET is generated.
Bits 9-13	Status information upon request completion. Zero indicates normal completion. Non-zero indicates an abnormal condition, not necessarily an error; an OWNCODE routine, if present, will be executed. Status codes are described under OWNCODE routines. Initially, this subfield is set to zero when the FET is generated.
Bits 0-8	Used primarily to pass function codes to a peripheral processor. Function codes are even numbers (bit 0 has a zero value). When the request has been processed, bit 0 is set to one. When the FET is generated, bit 0 must be set to one to indicate that the file is not busy. Bit 1 specifies the mode of the file (0 = coded, 1 = binary). Bit 1 is not altered by CPC when a request is issued. Bits 2-8 are used to pass function codes to a peripheral processor (file action requests). Bits 3 and 4 may be altered by the peripheral processor routine when the request is completed if an end-of-record (ten binary) or end-of-file was read (eleven binary). The initial value of bits 2-17 should be zero.

SCOPE CIO CODES IN OCTAL (CIRCULAR BUFFER I/O)

All codes indicated by - are illegal; all reserved codes are illegal. All codes are shown for coded mode operations; add 2 for binary mode. Example: 010 is coded READ, 012 is binary READ. Upon completion of operation, code/status in FET is changed to an odd number, usually by adding 1 to the code. In some cases, code is further modified to indicate manner in which operation concluded. Example: a READ function (010), at completion, becomes 011 (buffer full), 021 (end of logical record), or 031 (end of file).

000	RPHR	054	-	130	CLOSE,NR
004	WPHR	060	UNLOAD	134	-
010	READ	064	-	140	OPEN
014	WRITE	070	-	144	OPEN,WRITE
020	READSKP	074	-	150	CLOSE
024	WRITER (*)	100	OPEN,NR	154	-
030	-	104	OPEN,WRITE,NR	160	OPEN
034	WRITEF	110	POSMF	164	-
040	BKSP	114	EVICT	170	CLOSE,UNLOAD
044	BKSPRU	120	OPEN,NR	174	CLOSE,RETURN
050	REWIND	124	-		

(*) When a WRITER function is issued with level 17 specified, SCOPE changes the function to a WRITEF. Thus, a function issued as a 24 will return as a 34.

200 Series for special reads or writes (reverse, skip, non-stop, rewrite, etc)

200	-	230	-	254	-
204	-	234	REWRITEF	260	READN
210	-	240	SKIPF	264	WRITEN
214	REWRITE	244	-	270	-
220	-	250	READNS	274	-
224	REWRITER				

300 Series used for tape OPEN and CLOSE

300	OPEN,NR	324	-	360	-
304	-	330	CLOSER	364	-
310	-	334	-	370	CLOSER,UNLOAD
314	-	340	OPEN	374	-
320	-	350	CLOSER		
		354	-		

400 Series reserved for CDC

500 Series to be reserved for installations

600 Series

600	-	630	-	654	-
604	-	634	-	660	-
610	-	640	SKIPB	664	-
614	-	644	-	670	-
620	-	650	-	674	-
624	-				

700 Series reserved for CDC

DEVICE TYPE (DT) (12 bits)

The device type value will be returned to the FET device type field when a file action request is issued, if FET length is greater than the minimum.

GROUP I

Mnemonic	Device Type	Device
AA	01	6603-I disk**
AB	02	6638 disk
*--	03	data cell
AC	04	6603-II disk**
AL	05	821 data file
AM	06	841 multiple disk drive
AP	07	3234/854 disk pack drive
AF	10	814 disk file
*AE	11	3637/863 drum
AD	12	3637/865 drum
--	13-17	CDC reserved
AX	20	ECS
--	21-27	CDC reserved
--	30-37	Reserved for installations, mass storage only

*Codes are defined but supporting software is not provided by SCOPE.

**6603-I disk is a basic 6603 with or without field option 10098 (disk speedup) installed; 6603-II is a 6603 with both field options 10098 and 10124 (speedup augment) installed.

GROUP II

Mne-monic	Device Type	Device	Recording Technique
MT	40	7-track magnetic tape	(Right 6 bits in binary) xxxx00 HI density 556 bpi xxxx01 LO density 200 bpi xxxx10 HY density 800 bpi xxxx11 CDC reserved xx00xx Unlabeled xx01xx SCOPE standard label (USASI) xx10xx Alternate label xx11xx CDC reserved 00xxxx SCOPE standard data format 01xxxx CDC reserved 10xxxx S data format 11xxxx L data format
NT	41	9-track magnetic tape	10xx10 HD density 800 cpi 10xx11 PE density 1600 cpi 10xx00 CDC reserved 10xx01 CDC reserved 1000xx Unlabeled 1001xx SCOPE standard label (USASI) 1010xx Alternate label 1011xx CDC reserved 10xxxx S data format 00xxxx CDC reserved 01xxxx CDC reserved 11xxxx CDC reserved
--	42	member file* 7-track tape	Same as in MT
--	43	member file* 9-track tape	Same as in NT
**--	62	7-track multi-file set tape	Same as in MT
**--	63	9-track multi-file set tape	Same as in NT

*File in a multi-file set.

**CODE is generated when a tape is declared to have MF characteristics; the multi-file set code is used only in system tables and is not returned to the users FET.

GROUP III

Mne-monic	Device Type	Device
*TR	44	paper tape reader
*TP	45	paper tape punch
--	46	reserved for installations
--	47	reserved for installations
LP	50	501, 512, 505 line printer
L1	51	501, 505 line printer
L2	52	512 line printer
--	53-55	CDC reserved
--	56-57	reserved for installations
CR	60	405 card reader
KB	61	remote terminal keyboard
CR	64	200 user terminal card reader or Teletype paper tape reader
LP	65	200 user terminal line printer or Teletype paper tape punch
--	66-67	reserved for installations
CP	70	415 card punch
DS	71	6612 keyboard/display console
*GC	72	252-2 graphic console
*HC	73	253-2 hard copy recorder
*FM	74	254-2 microfilm recorder
*PL	75	plotter
--	76-77	reserved for installations

*Codes are defined but supporting software is not provided by SCOPE.

RANDOM ACCESS (r) (1 bit)

A one in the r field indicates that the file is a random access file r may be set to 1 by using the RFILEB or RFILEC macro.

When a file is opened or closed, the r bit setting determines action performed with regard to the file SCOPE index as shown below.

OPEN	**FET r=0**	**FET r=1**
File has no SCOPE index	No index action	FET r bit is set to zero and a nonfatal diagnostic is written to the dayfile.
File has a SCOPE index	No index action	Index is read into buffer; if index buffer is not specified, FET r bit is set to zero and a nonfatal diagnostic is sent to dayfile.

If a non-existent file is opened, the value of the r bit is not altered. Only files on allocatable devices may have an index. The FET r bit is set to zero if the file is on a non-allocatable device.

CLOSE	FET r = 0	FET r = 1
File had SCOPE index when last opened	File is flagged as not having index	If index buffer exists, the index is written and file is flagged as having SCOPE index. If buffer is not specified, nonfatal diagnostic occurs.
File had no SCOPE index when last opened	No index action	If index buffer is specified, index is written and file is flagged as having a SCOPE index. If index buffer is not specified, a nonfatal diagnostic occurs.

The above actions will be performed only if the file contents have been altered since the file was last opened.

When any other file action request is issued, the r bit setting determines the access method to be used. If r = 0, the file will be read or written beginning at the current location. If r = 1, the file will be read or rewritten according to the logical disk address in FET word 7, or written at the end-of-information, and the logical disk address returned to FET word 7.

RELEASE (n) (1 bit)

The release bit is set to one when record blocks are to be released after a forward skip or read operation. This bit setting has no meaning for any other operation.

UP BIT (1 bit)

The UP bit may be used to control end-of-reel processing. If UP is zero, reels are swapped automatically without notification to the user. If UP is one, control returns to the user with (02) in bits 9–13 of the FET code and status field.

If EOR is detected during a write operation, the request is examined. For WPHR, WRITE, WRITER or WRITEF (with data remaining in the circular buffer) an exit is made according to the UP conventions above. For WRITER or WRITEF with no data in the buffer, the function continues to its logical completion (including any recovery necessary) on this reel before exiting.

When EOR is detected during a read operation, this status is saved in the FNT. Reading continues, and the user is not notified until a tape mark is read, initiating normal EOR procedures.

If CPC is in use, control will be returned to the EOI OWNCODE routine. For a read, a CLOSER should be issued to cause a switch to the next reel to continue processing. For a write, end of volume information should be written by issuing a CLOSER or by writing tape marks and label information (for S and L tape).

ERROR PROCESSING (EP) (1 bit)

The EP bit is set when the calling program is to be notified of error conditions. If EP = 0, the operator may terminate the job (DROP) or continue it (GO).

ERROR BYPASS (EB) (1 bit)

Reserved for future use

DISPOSITION CODE (dc) (12 bits)

The value shown below will be returned to the FET disposition code field when a file action request is issued if FET length is greater than the minimum. A file with the specified default name will automatically be assigned the corresponding disposition code value at job completion.

For FR, FL, HR, HL, and PT, SCOPE recognizes the code and its value, but does not provide drivers.

Code	Value (octal)	Disposition	Default File Name
CK	xx01	Checkpoint	-
IU	xx02	Inhibit unload	-
SV	xx04	Save	-
PU	xx10	Punch Hollerith	PUNCH
PB	xx12	Punch Binary	PUNCHB
P8	xx14	Punch 80 Columns	P80C
FR	xx20	Film Print	FILMPR
FL	xx22	Film Plot	FILMPL
HR	xx24	Hard Copy Print	HARDPR
HL	xx26	Hard Copy Plot	HARDPL
PT	xx30	Plot	PLOT
PR	xx40	Print (501,505,512)	OUTPUT
P1	xx41	Print (501,505 only)	-
P2	xx42	Print (512 only)	-
-	xx7x	Reserved to Installation	-
-	x1xx	Change common file	-
-	1xxx	INTERCOM file	-
-	2xxx	INTERCOM batch job file	-
-	4xxx	EXPORT/IMPORT file	-

All other codes are reserved to the system.

LENGTH OF FET (lth) (6 bits)

The system FET length is determined as follows: FET first word address + 5 + lth = last word address + 1. The minimum FET length is five words (lth = 0). If the minimum FET is used, only the logical file name, code and status field, FIRST, IN, OUT, and LIMIT are relevant. No other field will be set or checked by SCOPE. A length of six words (lth = 1) is used if a working storage area is needed for blocking/deblocking. A length of eight words (lth = 3) is used if the r bit is set, indicating an indexed file. Length is nine words (lth = 4), if OWNCODE routines are declared. The maximum system FET length is 13 words (lth = 8). The maximum size is used if a labeled tape file is declared.

FNT POINTER (12 bits)

The FNT pointer is set by SCOPE, upon return from a file action request, to the location of the file in the FNT/FST. The pointer is placed in the FET to minimize table search time and does not affect the program. The pointer will not be set if a minimum FET is used.

PHYSICAL RECORD UNIT SIZE (PRU) (15 bits)

The physical record unit size of the device to which the file is assigned is returned in this field when a file is opened. It is given as the number of central memory words. The PRU size is used by CPC to determine when to issue a physical read or write. PRU size will not be returned if a minimum FET is used.

RECORD BLOCK SIZE (15 bits)

If the file resides on an allocatable device, the size of the device record block is returned in this field when the file is opened. It is given as the number of physical record units in a record block. If the number of PRU's is not defined or is variable, the field is set to zero. Record block size is not returned if a minimum FET is used.

FIRST, IN, OUT, LIMIT

Data is transmitted in physical record units, the size of which is determined by the hardware device. For example, the 6603 disk has an inherent PRU size of 64 CM words; binary mode magnetic tape files are assigned a PRU size of 512 words.

For each file, the user must provide one buffer, which can be any length greater than a PRU size. This is called a circular buffer because it is filled and emptied as if it were a cylindrical surface in which the highest addressed location is immediately followed by the lowest. The FET fields FIRST, IN, OUT and LIMIT control movement of data to and from the circular buffer.

FIRST and LIMIT never vary; they permanently indicate buffer limits to the user and to SCOPE. During reading, SCOPE varies IN as it fills the buffer, and the user varies OUT as he removes data from the buffer. During writing, the user varies IN as he fills the buffer with data, and the system varies OUT as it removes data from the buffer and writes it out—the program that puts data into the buffer varies IN, and the program that takes it out varies OUT. The user cannot vary IN or OUT automatically except when using READIN and WRITOUT functions; he must do this within the program by inserting a new value into lfn + 2 (IN) or lfn + 3 (OUT). For convenience, the words containing IN and OUT contain no other items, eliminating the need for a masking operation.

The system dynamically checks the values of IN and OUT during data transfers, making continuous read or write possible.

If IN = OUT, the buffer is empty; this is the initial condition. If IN > OUT, the area from OUT to IN - 1 contains available data. If OUT > IN, the area from OUT to LIMIT - 1 contains the first part of the available data, and the area from FIRST to IN - 1 contains the balance.

To begin buffering, a READ function may be issued. SCOPE will put one or more PRU's of data into the buffer beginning at IN, resetting IN to one more than the address of the last word filled after each PRU is read. Data may be processed from the buffer beginning with the word at OUT, and going as far as necessary, but not beyond IN - 1. The user must then set OUT to one more than the address of the last word taken from the buffer. He sets OUT = IN to indicate that the buffer is empty.

When a READ request is issued, if the buffer is inactive (no physical read occuring), CPC determines how much free space the buffer contains. If OUT > IN, OUT - IN words are free. If IN > OUT, (LIMIT - IN) + (OUT - FIRST) words are free. The system subtracts 1 from the number of free words, because it must never fill the last word; this would result in IN = OUT, and falsely indicate an empty buffer. If the number of free words, minus 1, is less than the PRU size, CPC does not issue a physical read request; control is returned normally.

The example below illustrates the use of IN and OUT pointers. Speed of operation is not considered; simultaneous processing and physical I/O are not attempted.

The initial buffer pointer position is:

FIRST = BCBUF

IN = BCBUF

OUT = BCBUF

LIMIT = BCBUF+500

The user issues a READ with recall request.

Ignoring the possibilities of an end-of-record or end-of-file, the system reads as many PRU's as possible (if PRU size is 64 words, 7 x 64 = 448 words) and leaves the pointers:

FIRST = BCBUF

IN = BCBUF+448

OUT = BCBUF

LIMIT = BCBUF+500

The user is processing items of 110 words. He takes four items from the buffer, leaving the pointers:

FIRST = BCBUF

IN = BCBUF+448

OUT = BCBUF+440

LIMIT = BCBUF+500

The user issues another READ request, since he knows the buffer does not contain a complete item. The system is aware that IN > OUT, so that the vacant space amounts to LIMIT - IN + OUT - FIRST = 492 words; since it must not fill the last word, it must read fewer than 492 words.

The nearest lower multiple of 64 is 7 x 64 = 448, so it reads 52 words into IN through LIMIT - 1, and then 396 more words into FIRST through FIRST + 395. It then resets IN so that the pointers look like:

FIRST = BCBUF

IN = BCBUF+396

OUT = BCBUF+440

LIMIT = BCBUF+500

The system has just used the circular feature of the buffer; now the user must do so. The next time he wants an item, he takes the first 60 words from OUT through LIMIT - 1, and the remaining 50 from FIRST through FIRST + 49. Then he resets OUT, making the pointers:

FIRST = BCBUF

IN = BCBUF+396

OUT = BCBUF+50

LIMIT = BCBUF+500

On input, this can continue indefinitely, with OUT following IN, around the buffer. The system stops on encountering an end-of-record or end-of-file, and sets the code and status bits accordingly. The system may, or may not, have read data before the end-of-record; so it is up to the user to examine the pointers and/or process the data before taking end-of-record or end-of-file action.

In writing, the process is similar, but the roles are reversed. The user puts information into the buffer and resets IN; and when he calls the system, it removes information from the buffer and resets OUT. For writing, the system removes data in physical record units and empties the buffer if possible. The user must be careful not to overfill the buffer; IN must not become equal to OUT. During the process of emptying the buffer, SCOPE resets OUT after each PRU has been written and checked for errors.

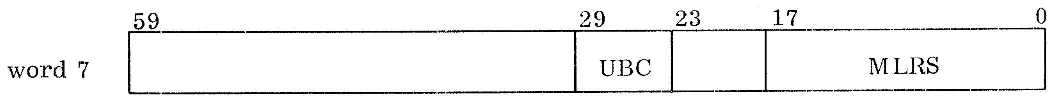
WORKING STORAGE AREA

The two fields in word 6 of the FET specify the first word address (fwa) and last word address + 1 (lwa + 1) of a working storage area within the program field length. Logical records may be deblocked into or blocked from this area into the circular buffer. (See READIN and WRITOUT.)

FILE INDEXING FIELDS

Words 7 and 8 are used for communication between the peripheral processor input/output routines and the running program depending on the device and file type.

For magnetic tapes with S or L data format, the structure of word 7 of the FET is:



UBC (Unused Bit Count) Bits 24-29

The UBC field is used for a file declared to have either S or L format. For a READ or READSKP function, SCOPE will store into this field the number of low-order unused bits in the last data word of the record. The UBC field is not used during a READN request. For a WRITE, WRITER or WRITEF function, SCOPE will read the contents of UBC and adjust the length of the record accordingly.

For example, to write a single record of 164 decimal characters, the data length is 17, to the nearest CM word. The number of low-order unused bits in the last word would be 36. The user would set UBC = 36, set IN and OUT pointers to reflect 17 words of data, and then issue a WRITE or a WRITER.

SCOPE does not use the UBC field during a WRITEN request. UBC may range from 0 to 59, but will always be a multiple of 12 when set as a result of a read operation. If it is not a multiple of 12 for a write request, SCOPE will truncate the value to the nearest multiple of 12; if UBC is 18, SCOPE will execute as though it were 12, and if UBC is 6, SCOPE will execute as though it were 0. The field in the FET remains unchanged.

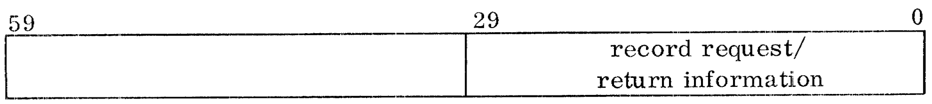
MLRS (Maximum Logical Record Size) Bits 0-17

The MLRS field contains the size of the largest logical record to be encountered (considered as valid when either reading or writing) when the S or L tape format is used. The size is given in number of CM words.

The MLRS field is required for all S and L tape operations; therefore, a 7-word FET is mandatory.

For S tape format, if MLRS = 0, the value of the maximum PRU is assumed to be 512 words. For L tape format, if MLRS = 0, the assumed maximum PRU is LIMIT - FIRST - 1 for standard reads and LIMIT - FIRST - 2 for READN.

For mass storage random files, the format of word 7 of the FET is:



The file indexing fields (record request/return information, record number index length and index address) are used for communication between the peripheral processor input/output routines and the CP programs. Index address and index length fields are declared when the FET is generated; the index buffer must be within the program field length. The record request/return information field is set to zero when the FET is generated. Both the indexing functions and the peripheral processor input/output routines set the field during random file processing.

For other than the SCOPE indexing method, the following information is pertinent. At the start of writing a new logical record, if the random access bit and the record request/return information field are non-zero, the latter field is assumed to contain the address of a location within an index. The PP routine inserts into that location (in bits 0-23) the PRU ordinal (starting from 1) of the logical record. To read the record again, the random access bit should be set to non-zero and the PRU ordinal should be entered in the FET in the record request/return information field.

OWNCODE ROUTINES

Addresses of user supplied routines may be given in word 9 of the FET. These routines are executed by CPC as indicated below. A zero value indicates that no routine is supplied.

An OWNCODE routine should be set up like a closed subroutine with execution beginning in the second word of the routine. CPC calls an OWNCODE routine by copying the exit word of CPC into the first word of the OWNCODE routine, putting the contents of the first word of the FET into X1, and branching to the second word of the OWNCODE routine.

Termination of an OWNCODE routine by a branch to its first word causes a branch to the point in the program to which CPC would have returned if the OWNCODE routine had not been called. CPC saves and restores all registers except X1, A1, X6 and A6.

EOI Address Field

CPC enters the end-of-information (EOI) routine under the following circumstances:

Bits 9-13 of Code and Status:

01 End-of-information encountered after forward operation JX.

02 End-of-reel reflective spot has been reached on magnetic tape. When a read is performed, an and-of-volume consists of both this spot and end-of-file: bits 3 and 4 of Code and Status are ones; for labeled files the end-of-reel code will not appear in the FET until an EOV label is encountered.

Just before entering an end-of-information OWNCODE routine, CPC zeros bits 9 and 10 of the first word of the FET. However, as the routine is entered, X1 still contains the first FET word as it appeared before those bits were zeroed.

ERROR Address Field

This field specifies an address to receive control if an error condition occurs after a file action request. The FET code and status field will reflect the error condition. If processing can continue, the error routine should exit through its entry point; otherwise, an ABORT request may be issued.

If the error address field is zero, the run continues normally. The FET code and status bits reflect the error condition upon normal return to the program.

Bits 9-13 of Code and Status (values are octal):

End-of- information or end-of-reel may occur simultaneously with code 4 and code 10; code and status will contain their logical sum.

04 Irrecoverable parity error on last operation, or lost data on write.

10 During a magnetic tape reading, the physical record size exceeded circular buffer or maximum allowable PRU size (MLRS for S and L tapes). During a mass storage write, all mass storage space meeting the file requirements was in use or otherwise unavailable.

20 Reserved for system.

21 End of multifile set. An attempt was made to position to a file whose position number is greater than that of the last member in the set. Any subsequent attempt to reference the logical file name assigned to the nonexistent member will result in a fatal error.

22	Fatal error.
23	Index full.
24	Reserved for future use.
25	An attempt was made to read or write record number n of a random file. but the index of the file is full.
26	An attempt was made to read a named record from a random file. but the name does not appear in the index.
27	An attempt was made to write a named record on a random file. but the name does not appear in the index. and there is no room to add a new name.
30	Function legal but not defined on device.
31	Permanent file permission not granted.
32	Function legal except for permanent files.
33 37	Reserved for future use.

If both EOI and error routine execution are needed. the error routine is executed. Just before entering an error OWNCODE routine. CPC zeros bits 11-13 of the first word of the FET. However. as the routine is entered, X1 contains the first word of the FET as it appeared before those bits were zeroed.

FET CREATION MACROS

System macros in the COMPASS language facilitate generation of the system FET, as follows. The subfields (WSA, UPR, IND, OWN. LBL, EPR, UBC. MLR) are order-independent; within the subfield, order is fixed. Upper case characters designate subfield content. lower case characters indicate parameters to be supplied by the user. All parameters except lfn. fwa, and f are optional.

These macros generate a 13 word FET which is truncated, if necessary, to the minimum length required. All 13 words remain in the block when a USE statement immediately follows the macro; truncated label common block may occur if this is the last code in the block.

CODED FILE – SEQUENTIAL

lfn FILEC fwa, f, (WSA = addrw,lw), (OWN = eoi, err), LBL, UPR, EPR UBC = ubc, MLR = mlrs

FILEC also is applicable to conversion-mode files on 9-track tapes.

BINARY FILE – SEQUENTIAL

lfn FILEB fwa, f, (WSA = addrw,lw), (OWN = eoi, err), LBL, UPR, EPR, UBC = ubc, MLR = mlrs

FILEB also is applicable to packed-mode files on 9-track tapes.

CODED FILE – RANDOM

lfn RFILEC fwa, f, (WSA = addrw,lw), (IND = addri,li), (OWN = eoi, err), LBL, UPR, EPR

BINARY FILE – RANDOM

lfn RFILEB fwa, f, (WSA = addrw,1w), (IND = addri,1i) (OWN = eoi, err), LBL, UPR, EPR

lfn	File name
fwa	Substituted in FIRST, IN, and OUT
f	Length of circular buffer (fwa + f is substituted in LIMIT so that it reflects the lwa + 1 address of the buffer area)
WSA	Working storage area parameters
addrw	First word address of working storage area
lw	Length of working storage
IND	Index buffer parameters
addri	First word address of index buffer
li	Length of index buffer
OWN	OWNCODE routines
eoi	End-of-information address
error	Error address
UPR	User specifies processing at end-of-reel
LBL	Label information will follow. The LABEL macro providing label information, must immediately follow the FILE macro to which it pertains.
EPR	User specifies handling of error conditions.
UBC	Unused bit count: 6-bit code in FET field descriptions (S and L tapes only). Generates a 7-word FET mandatory for S and L tapes.
MLR	Maximum logical record size: 18-bit code in FET field descriptions (S and L tapes only). Generates a 7-word FET mandatory for S and L tapes

Examples:

To create a minimum FET for the standard INPUT file:

```
LBUFFER EQU 65
INPUT FILEC BUFFER,LBUFFER
```

To create an FET for a binary random file:

```
LBUFFER EQU 65
LINDEX EQU 25
FILEABC RFILEB BUFFER,LBUFFER,(IND=INDEX,LINDEX)
```

To create an FET for a labeled tape file with user processing at end-of-reel condition. OWNCODE routine is supplied:

```
TAPE1 FILEB BUFA,LBUFA,LBL,UPR,(OWN=PROCEOR)
TAPE1 LABEL SORTINPUTTAPE,32,90
```

To create an FET for a list file. OWNCODE routines are supplied and the working storage area is used:

```
PRINT FILEC BUFB,LBUFB,(WSA=LINE,14),(OWN=ENDING,ERRORS)
```

LABEL MACRO

lfn LABEL fln,ed,ret,create,reel,mfn,pos

The LABEL macro may be used to generate FET label field information. It must immediately follow the FILEx macro to which it pertains. The LABEL macro is a data generation statement, which does not directly cause any action on the file. If any parameter is absent, the field is set to binary zero. Alphanumeric values are left justified , numeric values right justified in the field. If the parameter is smaller than the field size, the fill character is binary zero. Parameter values are shown below.

If FET label field information is generated by other means, display code zero may be used as the fill character in numeric fields; and display code blank in alphanumeric fields.

File Label Name (fln)	File identification, 1-17 alphanumeric characters. Default is 17 blank characters.
Edition Number (ed)	File version, 1-2 digits. Default is 01.
Retention Cycle (ret)	Number of days a tape is to be protected, 1-3 digits. Default is installation parameter.
Creation Date (create)	Creation date in Julian format (YYDDD), 5 digits. Default is today's date.
Reel Number (reel)	Reel of file, 1-4 digits. Default is 0001.
Multifile Name (mfn)	Logical set name of which current file is member, 1-6 alphanumeric characters. Default is 6 binary zeros in FET.
Position Number (pos)	Relative position of current file in multifile set, 1-3 digits. Default is 000.

When a label is created, default values are assigned for any field containing binary zero. Alphanumeric fields are blank filled; numeric fields are display code zero filled. The resulting field values are returned to the FET label fields and used to format the 80-character file header label (Appendix C).

When a label is checked, non-default alphanumeric fields will be blank-filled and non-default numeric fields will be display code zero-filled. Resulting fields will be compared with header label fields. Default FET fields (containing binary zero) are not compared. If all declared FET label fields match their counterparts in the header label, all header label fields will be returned to the FET label fields. If the operator chooses to accept a non-matching label, the FET label fields will be set to the values contained in the header label.

FILE PROCESSING

The FNT entry for a file in the input or output queue differs slightly from other FNT entries. The FNT will contain a priority field for a file in either queue. The priority for input queue files depends on the priority specified on the job card, the length of time the file has been in the queue, and possibly the other job card parameters. Input queue priorities determine the order in which jobs are brought to control points.

For a file in the ouput queue, priority is determined according to the time the file entered in the queue; it is incremented with time. JANUS and other output routines normally process files in order of priority; but the priority of files in either queue can be changed by an operator type-in.

The input queue FNT entry contains job card parameters such as central memory and ECS field length requirements and the time limit. Output queue FNT entries contain an additional field which specifies how much data is on the file, given as a multiple of blocks of 100(octal) central memory words.

INPUT QUEUE FILES

Files are placed into the input queue by JANUS, as it reads in jobs from the card reader, by other system jobs such as LOAD and RESQ, or by EXPORT/IMPORT. An input queue file contains the entire job which consists of a record of control cards followed by any number of data records. Files must be type input, assigned to control point zero, with non-zero priority. The file name will be the job name. Input queue files are always put on allocatable devices.

OUTPUT QUEUE FILES

Files can be put in the output queue by the user when CLOSE,UNLOAD, CLOSE,RETURN, or DISPOSE is performed on a file, or by the system at job termination. Files in the output queue must be type local or output, assigned to control point zero, with non-zero disposition and non-zero priority. The file name is the name of the job which created the file. Files are always rewound before they are put into the output queue.

A file will not be put in the output queue unless it is on an allocatable device, as JANUS and other output routines cannot handle non-allocatable files. A random file can be put in the output queue; however, JANUS handles random files sequentially. The categories of local files which may be attached to the control point of a running job are described below.

INPUT FILE

When a job is brought to a control point, the input queue FNT entry is changed so that the file is assigned to the control point, has the name INPUT, and is type local. It is still the same file as the input queue file; it contains the control card and data records; and it resides on an allocatable device. Since the input file contains the control cards for the job, it is illegal to CLOSE, UNLOAD the input file; any attempt to do so will be ignored. SCOPE maintains special pointers to the control card record on the input file. Normal operations such as READ or REWIND on the input file do not affect these pointers. The user can, however, call the system macro, CONTRLC, which can change the pointers to the control card record and affect the order in which control cards will be processed.

The user should never write on the file INPUT or detach it from his control point, although both are possible. In the first case, the user could destroy the control card record; in the second, the entire file could be destroyed or lost.

OUTPUT FILE

The output file is a local file at the user's control point with the name OUTPUT. When an error condition occurs, the system will dump to the output file the exchange package plus 100(octal) central memory locations preceding through 100 following the address to which the P register pointed at the time of error. At job termination, the job dayfile is copied to the end of the output file. If the operator types in KILL to terminate the job, no output is produced.

If no output file exists, the system will create one. The output file is always on an allocatable device, as OUTPUT cannot be requested with either the control card or macro. An attempt to do so will terminate the job.

When it is time to dispose of the OUTPUT file, it is treated as a special name file. If the user does not specify a disposition code, the file will be given print disposition and placed in the output queue. If an output file is disposed of by a CLOSE,UNLOAD it is put into the output queue with type local. At job termination, the output file is put into the output queue with type output; therefore only one file for each job can have type output. However, a file may be output to the printer through the DISPOSE command.

SPECIAL NAME FILES

Several other file names are treated as special cases. The user can specify disposition for these files; however, if a local file with one of the special names has a zero disposition code, the system will automatically set the disposition code to an appropriate value before disposing of the file.

Names treated as special cases are the following:

OUTPUT	FILMPL
PUNCH	HARDPR
PUNCHB	HARDPL
FILMPR	PLOT

NON-ZERO DISPOSITION FILES

All local files which have disposition set by the user and all special name files which have disposition set by the system will be rewound and placed in the output queue at disposal time if they reside on allocatable devices.

NON-ALLOCATABLE FILES

If a local file resides on a non-allocatable device, it will be dropped (its FNT zeroed) at disposal time, regardless of its name or disposition code. It will not be put into the output queue, because JANUS and other output routines cannot process non-allocatable files.

COMMON FILES

A user can change the type of any allocatable local file to common if it is attached to his control point and it is not a tape, a permanent file or a private disk pack file.

He can also attach an existing common file to his control point. If a common file is not being used, it will be attached to control point zero. A common file may have a special name and it may have non-zero disposition.

When it is time to dispose of a common file, the file is assigned to control point zero and made type common regardless of its name or disposition code. The disposition code is saved; however, and when the file is released and made local, it is treated as a special name file or a non-zero disposition code file, if appropriate.

PERMANENT FILES

A permanent file can survive across deadstarts. The file and sufficient information to access the file is maintained on a mass storage device. An FNT entry is made for a permanent file only when the file is attached to a control point; it will be type local and marked as a permanent file. A permanent file can never be assigned to control point zero.

At job termination, a permanent file is disposed of by deleting its FNT entry and releasing its RBT chain. The file will still exist on the mass storage device. A CLOSE,UNLOAD or CLOSE,RETURN on a permanent file also saves the file but deletes the FNT. A special name or a non-zero disposition for a permanent file is meaningless and is ignored.

A user can remove completely a permanent file from the system with the PURGE command. The mass storage space occupied by the file will be released.

A permanent file cannot be used as a common file; any attempt to do so will be ignored, and a non-fatal error message will be issued.

PRIVATE DISK PACK FILES

A disk pack unit may be designated at system deadstart as public or private. A public unit is treated the same as any other mass storage device; it may contain one or more files of various types assigned to different control points.

A private disk pack unit is treated somewhat like a magnetic tape. The unit can be assigned to a control point with the RPACK control card, and it may contain one or more files associated with the control point. A private disk pack unit cannot be attached to control point zero. The REQUEST control card must be used to create a file on a unit attached to a user's control point. The name on the REQUEST card cannot be one of the special names; if it is, the job is terminated with a control card error.

When a private disk pack unit is attached to a control point, an FNT entry is made for each file on the unit as well as one FNT entry for the unit itself. The latter is marked so a file on the pack can have the same name as the pack. The files will be type local and will be assigned to the control point.

A private disk pack file cannot be made type common or put into the output queue. A disposition code specified for a private disk pack file will be ignored. At job termination, information about each private disk pack file (such as name and physical disk locations used by the file) is written to the disk pack; and the file itself remains on the pack; but the FNT entry for each file is deleted from the FNT. The FNT entry for the pack is also deleted.

Other entries in system tables referring to the private pack files are deleted, and the equipment is detached from the control point.

To eliminate a file from the private disk pack and from the system, the REMOVE control card is used.

RANDOM FILES

If a random file is to be saved, the file index must be written as the last logical record on the file. A user may write the index or he may call the system macro CLOSE or CLOSE,UNLOAD to write the index. CLOSE automatically writes out an index for a random file if the file contents were changed by a write with the FET random bit set. A permanent file must also have EXTEND permission.

If the user neglects to write out the index on a random file or an error terminates the job before he has a chance to do so, the system will write out the index if the file satisfies all the following conditions:

Random file has been changed since the last close.

The FET indicated by the pointer in the FNT must appear valid: The random bit must be set, the FNT pointer must be correct, the index must be less than the field length. This is to prevent the system from writing out the index if the index and the FET which points to it have been destroyed.

The file must be permanent with EXTEND permission, a common file, or a private disk pack file.

FILE DISPOSITION

When jobs terminate, the method of file disposal varies according to the circumstances that cause termination.

NORMAL TERMINATION

1. All local files with non-zero disposition code (including output and other special name files) which reside on allocatable devices are rewound and put into the output queue by changing the file name to the job name and assigning the file to control point zero.

2. All common files are assigned to control point zero with type common. They are not rewound.

3. All FNT entries for permanent files at the control point are deleted.

4. All FNT entries for private disk pack files and the FNT entry for the disk pack unit are deleted from the FNT. Information identifying each file is written on the disk pack and the equipment is detached from the control point.

5. All other files are dropped including the input file, local files with zero disposition, and local non-zero disposition files on non-allocatable devices. All tapes are rewound.
   The entry in the FNT for the file is deleted, any reserved mass storage space on an allocatable device is released, and non-allocatable equipment is dropped. A file on a non-allocatable device remains at its current physical position, or it is rewound if legal.

KILL

When the operator types in n.KILL at the control point, the job terminates and all local files associated with the job, including the output file, are dropped regardless of name or disposition. Permanent files, private disk pack files, and common files are treated the same as for normal termination. No files are put into the output queue. No dayfile is printed for the job.

RERUN

When the operator types in n.RERUN at the control point, the job is terminated and returned to the input queue, so that it can be run later. Files associated with the job are handled as follows:

The input file is returned to the input queue. Its file name will be the job name, and it will be type input at control point zero.

The output file is dropped, and a new output file is created. The job dayfile is copied to the new output file which will not be rewound, but made local at control point zero; the file name will be the job name. It is called a pre-output file, and becomes an output file when the job is run again. The output file for the rerun job will contain the dayfile from the first partial run of the job and the output and dayfile from the second complete run of the job.

A job that uses common files or attaches permanent files with any permission except READ cannot be rerun.

All other files, regardless of name or disposition, are dropped.

NO RERUN

In some cases, a job might perform a function which would make it impossible to restore conditions to their initial state (before the job was run). For example, if a job writes on an existing common file, that information cannot be erased. When such a job is rerun, results are unpredictable. To avoid this, the no-rerun flag can be set in the control point area. If set, the operator type-in n.RERUN will be rejected. The no-rerun flag will be set in the following cases:

1. The user attaches an existing common file to his control point (because it cannot be determined if information has been written on the file).

2. The user detaches a common file from his control point with CLOSE,UNLOAD or COMMON macros, or the RETURN control card.

3. The job has attached a permanent file to its control point with EXTEND, MODIFY, or CONTROL permission, or has cataloged a file.

DISPOSAL PRIOR TO TERMINATION

Prior to job termination, a user can dispose of any file attached to his control point by calling the system macros CLOSE,UNLOAD or CLOSE,RETURN, by using the RETURN control card, or by using the DISPOSE macro or control card. A common file can be disposed of by calling the system macro COMMON. After the user disposes of a file, he may create and reference a new file having the same name.

CLOSE,UNLOAD

The system macro CLOSE can include the parameter UNLOAD to specify disposal of a file at the user's control point. Files disposed of in this way are handled as for normal job termination with the following exceptions:

1. CLOSE cannot be used for the INPUT file.

2. A sequential file is rewound.

3. A file on magnetic tape is physically unloaded.

4. A file on a private disk pack is locked.

5. CLOSE,UNLOAD (or any other CLOSE) will write out the index to any random file.

RETURN

The RETURN control card can be used to dispose of a file at the user's control point. RETURN will perform a CLOSE,RETURN on the file with the exception that no index is written on a random file.

DISPOSE

The DISPOSE control card or macro can be used to dispose of a file at the user's control point. A disposition code and special handling of the file may be specified.

COMMON

The system macro COMMON can be used to dispose of a common file at the user's control point. Disposal is the same as in normal termination. If the file referenced in the call is not type common, no action is taken; but an error indication is returned to the user.

BUFFER EMPTYING

The system will empty the buffers of certain files before disposing of them. A write end-of-record will be issued to write any information still in a user's buffer onto the associated file. Buffers are emptied when a job terminates and the error flag is not KILL or RERUN; or when an error (other than KILL or RERUN) occurs which does not cause termination because the job contains an EXIT card followed by more control cards.

In the former case, the system will empty the buffers as described above. In the latter, the system will empty the buffers, process the error condition, and continue processing cards after the EXIT card. A CLOSE,UNLOAD will not empty the buffers.

Files with non-zero disposition (including output and other special names) are emptied if they meet all the following conditions:

The write bit in the FNT is set.

The FET pointer in the FNT points to an FET with the same names as the file name.

The buffer is not empty and the write bit is set.

The FET pointer in the FNT points to an FET with the same name as the file name.

The buffer is not empty.

For common files, buffers are emptied regardless of disposition.

The FET pointer in the FNT points to the address of the FET used in the last I/O operation. If a user has two FET's for a file, it may not be possible to determine which buffer is emptied.

EXPORT/IMPORT

When files attached to an EXPORT/IMPORT job are disposed of by the user or the system, they are handled the same as for a normal job, except that 4000(octal) is added to the disposition codes of all files placed in the output queue. Files in the input queue with such disposition codes are ignored by JANUS, as they are picked up by the EXPORT/IMPORT package. In some cases, the EXPORT/IMPORT package may reset the disposition so that JANUS will process the output.

TAPE FILE STRUCTURE

SCOPE defines a structure on all files. The basic unit of this structure is called a physical record unit (PRU). All data within a file is recorded in PRU's. The representation of a PRU is device dependent.

A file may be viewed as an ordered collection of PRU's, each of which is a collection of bits and characters depending on the file mode: binary or coded. The collective content of the PRU's makes up the data of the file.

When a file is attached to a job, a position is defined to the file. This file will always be positioned at one of the delimiters between 2 PRU's. On tape this is the inter-record gap. The first position is called beginning of information. The last position is called end of information (EOI). End of information is defined as an EOF trailer label. All SCOPE tapes and labeled S and L tapes contain EOF trailer labels. No EOI is defined for unlabeled S and L tapes because they have no trailer labels. End of file is returned if a tape mark is read before the end of tape reflective marker, end of reel (EOR) if a tape mark is read after it. A rewind function will position to beginning of information except on multireel tape files.

A read function transfers data from one or more PRU's to the user's buffer, and the position advances the delimiter following the last PRU read. The position may not advance beyond EOI. If an attempt is made to read past EOI (when defined), EOI status will be returned.

A write function on a file transfers data from the user's buffer to the file. Write functions on mass storage devices always occur from EOI as successive PRU's are written. Rewrite-in-place is not defined for tapes. Any write occurs at the current position of the tape.

In general, a PRU does not have a fixed length. Depending on the device and, in some cases, on the mode (binary or coded), a PRU does have a maximum length. The term "length" applies to the amount of data contained in a PRU and not to the amount of space occupied by a PRU. A short PRU has a length that is less than maximum for the device. A zero-length PRU does not contain any user data. Short and zero-length PRU's are used by SCOPE to delimit the logical structure specified by the file creator.

A tape record of three bytes or less is considered to be noise and is ignored by SCOPE. Coded tapes are written in external BCD, even parity. Binary tapes are written in binary, odd parity.

SCOPE STANDARD BINARY FORMAT

A physical record unit consists of two parts: a data block of 5n 12-bit bytes, $0 \leq n \leq 512$; a suffix block of k 12-bit bytes, $k = 0$ or $k = 4$. If the data block is a full 512 words, the suffix block is of length 0; otherwise, it is 4 bytes long.

The PP generates the suffix on writing the data and strips it off on reading the data. Only the data block is transferred between a PP and a user buffer.

SCOPE STANDARD CODED TAPES

A physical record unit consists of two parts: A data block of 10n 6-bit characters, $0 \leq n \leq 128$; a suffix block of k characters, $k = 0$ or $k = 8$. If the data block is a full 128 CM words, the suffix block has length zero; otherwise, it is 8 characters long.

The PP generates the suffix on writing and strips it on reading. Only the data block is transferred between a PPU and a user buffer.

The PP performs a conversion from display code to internal BCD on output and from internal BCD to display code on input. (This conversion is done by hardware if tape unit is connected to 6684.) The tape controller converts the BCD codes so that the tape is recorded in external BCD. The conversion table is given in Appendix A.

Peculiarities for coded tape for the 63-character set:

	OUTPUT			INPUT	
	Display Code	Internal BCD	External BCD	Internal BCD	Display Code
	0	16	16	16	0
	33	00	12	00	33
	63	12	12	00	33
Line Terminator	0000	1672	1632	1672	0000

Display code 00 (colon) is not a valid character; display code 63 (percent) is lost. Line terminators (byte of all zeros in lowest byte of a central memory word) will not result in the loss of zeros.

Peculiarities for coded tape for the 64-character set:

	OUTPUT			INPUT	
	Display Code	Internal BCD	External BCD	Internal BCD	External BCD
	0	12	12	12	33
	33	0	12	12	33
	63	16	16	16	63
Line Terminator	0000	1672	1632	1672	0000

Display code 00 (colon) is lost; display code 63 (percent) is now a valid character. All zero characters (up to 9) preceding a line terminator are changed in the PP buffer to 63B. On tape, they result in external BCD 16's. When tape is read, all 63's preceding line terminators are converted to display code zeros. This substitution ensures preservation of all zeros preceding a line terminator, regardless of the graphic character set used.

S AND L TAPES

S and L tapes are special cases. A logical record is equivalent to a physical record. Physical records contain multiples of two characters with a maximum of 5120 for S tapes; L tapes are limited only to the size of the user buffer. On S and L tapes, the minimum size is determined by the installation parameter governing the noise record size, and they have no level numbers.

MAGNETIC TAPE LABEL PROCESSING

The maximum benefit from the SCOPE tape scheduling and automatic tape assignment features can only be derived if most magnetic tape files are labeled. Two standard formats of tape labels are allowed: ANSI standard labels (U labels) and Control Data 3000 series labels (Y labels). Label formats and tape structures are given in Appendix C.

The presence/absence of a label and label format is determined by the user. Labeled tapes provide the following advantages:

When a write ring is inadvertently left in an input tape reel, software checking will ensure that a tape with an unexpired label is not over-written without the express permission of the operator.

The visual reel number of any U labeled tape reel read or written will be recorded in the dayfile.

Labeled tapes can be located and assigned by SCOPE without operator intervention, reducing the amount of time spent by the job at a control point.

The number of tape blocks (physical records) written on a file will be recorded both in the trailer label and in the dayfile. On subsequent file reading, the count serves as additional verification that data was read properly.

System checking will ensure that reels belonging to a multireel labeled tape file are read in the proper sequence.

More than one distinct U labeled tape file may be stored on one reel of tape. Label utility programs are supplied listing the contents of multifile tapes and for positioning to a particular file. Label utility programs are callable by control card. The label utility program is described in section 8.

RANDOM ACCESS FILES

RANDOM ACCESS FILE STRUCTURES

A file on an allocatable device is potentially a random access file. The allocatable devices are drums, disks and disk packs. Generally, these random access devices are shared by many files. A disk pack however, may be privately assigned to a single file attached to a single job, in which case it is not considered allocatable.

A file may be of arbitrary length and may be segmented over more than one device. The data will be recorded in a logical sequence of record blocks which may be arbitrarily scattered about the disk or drum surface. SCOPE maintains a central memory table for each file, called the record block table (RBT), in which the sequence of allocated record blocks is defined. The end-of-information position is also defined in the RBT.

No physical distinction exists between binary and coded files recorded on allocatable devices. Furthermore, no information is inherent to the file or the descriptive tables which indicates mode. The user is responsible for proper interpretation. Data is recorded in an integral number of CM words from a user buffer and is returned to the buffer without transformation. A file may be written in binary and read as coded, or conversely. This practice is not recommended, as it will not work on other devices.

For a description of disk labels, refer to Appendix C.

RANDOM FILE TYPES

SCOPE may be directed to read or rewrite data, beginning at any relative physical record unit (PRU) position within a file; when data is added to a file, SCOPE may be directed to return the relative PRU position at which the data begins. The following methods are available for accumulating such record addresses for future file processing.

SCOPE INDEXED SEQUENTIAL

The SCOPE Indexed Sequential (SIS) system is a set of related central processor routines that provides data and index management for large random files. SIS files may be processed through COBOL, FORTRAN, and COMPASS language programs and by control card call of SIS utility programs. SIS files may be declared as permanent files to be maintained permanently on an allocatable device, or they may be reloaded as needed through SIS utility programs.

Through the SIS language, the SIS system can retrieve and update fixed or variable length data records by keys as well as sequentially. Data record keys may be integer, floating point, or up to 4096 alphanumeric characters. One or more levels of index, maintained by SIS, allow direct access to any data record within a SIS file. SIS files may be read sequentially also without reference to indexes. Because SIS files are formatted uniquely to permit fast access in either indexed or sequential mode, they may be processed successfully only through the SIS system.

SIS utility programs allow creating, dumping, and reloading of a SIS file by control card call. Statistics about a SIS file may also be requested by control card.

The SCOPE Indexed Sequential language and control card formats are described in a separate publication.

No specific provision is made for sequential processing of SCOPE name/number indexed files (not SIS files), but sequential mode operations are not forbidden. Meaningful results can be expected only if the file has not been altered since its creation, or if the only modifications were made by rewriting in place, where the new record is the same length as the replaced record. In either case, the last record in the file is the SCOPE index.

Updating of a SIS file in sequential mode is limited to replacing records with records of the same length.

USER DEFINED

SCOPE permits much flexibility in the handling of random access files. Single-level SCOPE name/number indexed files may be created and maintained using IORANDM and system macros OPEN and CLOSE; data record management at any level lower than a SCOPE logical record is left to the user.

IORANDM may be used to create and maintain index contents during program execution without using OPEN/CLOSE to manage the index records. The user is responsible for managing his index records. Index records could be kept on a separate file, for example.

Multilevel SCOPE name/number indexed files may be created and maintained using IORANDM and system macros OPEN and CLOSE plus a user generated sub-index management routine. A master index record would contain addresses of sub-index records interspersed throughout the file. The master index record would be processed by OPEN/CLOSE as is a single-level index record. The user routine would need to ensure that IORANDM was referencing the correct index or sub-index block.

Other index formats may be defined by supplying a user routine to format and retrieve record names and mass storage addresses.

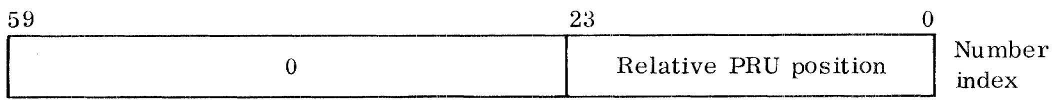
Mass storage addresses may be computed on files containing fixed length records since they are in the form of a relative PRU count and the PRU size is fixed.

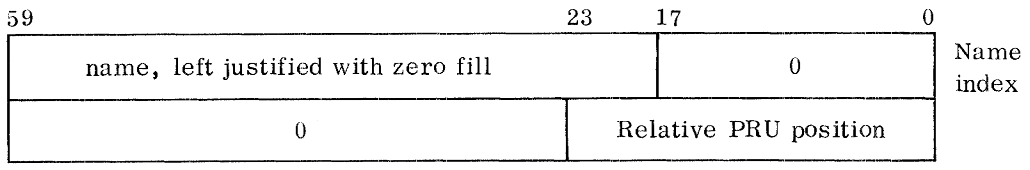
SCOPE NAME/NUMBER INDEX

SCOPE indexed files may be created and accessed by using the central processor routine IORANDM. SCOPE indexed files may be processed through COBOL, FORTRAN, and COMPASS language programs. SCOPE indexed files may be declared as permanent files to be maintained permanently on an allocatable device.

SCOPE indexed files may be created, read, written, and rewritten using the macros OPEN, CLOSE, READIN, WRITOUT, WRITIN, and WRITER. Management of a single index level is provided through system macros OPEN and CLOSE.

The first word in the SCOPE index determines how the records are referenced. A positive nonzero value indicates that records are referenced by number; a negative value indicates that records can be referenced by name or number. The number of a record is defined as equal to the relative position of the index entry for that record; the first entry in the index points to record 1, the second entry points to record 2, etc. If a name index is used, records may be assigned a name of up to 7 alphanumeric characters. The value of index word 1 is determined when the first record is written. The formats of a name index entry and a number index entry are shown below.





The smallest unit of information that can be entered in either index is a SCOPE logical record, and each SCOPE logical record must begin in a new PRU. The most economical use of SCOPE indexes occurs when SCOPE logical record length is equal to an integral number of PRU's minus one word.

INPUT/OUTPUT 4

The control cards direct the SCOPE system to initiate and terminate user programs. SCOPE also supervises user program runs, controlling all input/output operations and providing program-system interface.

These run-time functions are performed by system subroutines called by the user program. A comprehensive set of system macros is available for calling subroutines and generating tables for passing the parameters between system components. System subroutines and communication tables reside in the user's field length and should be considered when field length is specified.

During a user program run, the system performs two types of operations:

Input/output operations, initiated by file action requests in the user program.

System operations, initiated by system action requests.

USER/SYSTEM COMMUNICATION

The following describes user/system communication on four levels, beginning with the most basic.

BASIC COMMUNICATION

All requests from the user to the system are made through relative location 1 of the user program (absolute location RA + 1). RA + 1 is initialized to zero by the loader; a zero value indicates that no request is being made. When a service of some sort is required by the user, RA + 1 is set to some nonzero quantity. This quantity always assumes the general form:

Bit 59-42	3 display code characeters
Bit 41	zero = file action request; one = system action request
Bit 40	auto-recall bit
Bit 39-36	zero
Bit 35-0	parameters as required by the function

The system monitor, MTR, frequently examines RA + 1 of the program currently in control of the central processor (CP). When a nonzero quantity is detected, MTR begins to process the request. RA + 1 is zeroed when monitor processing is completed.

REQUESTS PROCESSED BY MTR

Four requests are processed directly by MTR: END, ABT, TIM, and RCL. END causes normal termination of a run; ABT causes abnormal termination; TIM returns the clock status.

RCL causes the program to be put into central processor recall. Either periodic or automatic recall may be requested. In either case, the program loses control of the CP when the RCL request is received. Periodic recall is requested with the letters RCL in RA+1 with the remainder of the word containing zeros. The CP program will regain control of the CP the next time around MTR's main loop (approximately 64 milliseconds). Typically, the program then will check the condition that made it enter recall and determine whether or not to re-enter recall. For most conditions, it is easier and less time consuming to enter automatic recall.

Automatic (or auto) recall is requested with the letters RCL in RA+1, the auto-recall bit set to one, and bits 17-0 containing the relative address of a word within the program field length. For auto-recall to function properly, this word should initially contain an even quantity (bit zero = 0);the CP program will be given control again when bit zero of the specified word becomes one. Thus, if auto-recall is used improperly, either the program regains control the next time around the MTR loop (specified word is initially odd), or the program never regains control (word is initially even and no PP program sets it to an odd value). If auto-recall is requested, MTR saves the contents of RA+1 in the control point area; and each time around the loop, it checks the specified word for an odd value. Auto-recall may also be used in conjunction with any other request, provided only that the request has in bits 17-0 an address of a word which conforms to the above rules, and that the auto-recall bit is one.

In the case of END, RCL, TIM, and ABT, zeroing of RA+1 indicates completion of the request. Typically, the sequence of code enters the request into RA+1 and loops until RA+1 contains zero. One implication of this feature is that, in the case of an auto-recall request, the specified word will be odd upon exit from the loop. This is true because control is transferred by the CP program before RA+1 is set to zero and will not regain control until the specified word is odd.

If the contents of RA+1 specify neither END, ABT, TIM, or RCL, the request is assumed to be one of the two types given below. MTR checks the first character of the request to ensure that it is alphabetic. Bits 41 and 39-36 are set to zero, and the control point number is inserted in bits 38-36; the request then is placed in the input register of a free PP. If the auto-recall bit is set, the program is put into automatic recall. RA+1 then is set to zero. Again, if auto-recall was requested, the specified word will be odd (which usually signifies request completion) when the RA+1 zero loop is satisfied.

INPUT/OUTPUT REQUESTS

Input/output requests include all requests which call CIO into a PP. These requests are made through RA+1 with the appropriate program name, an optional auto-recall bit, and the first word address of the file environment table (FET). A user issuing I/O requests on the basic level has the responsibility of checking and setting the first word of the FET before setting RA+1. Before the contents of FET word 1 are altered, they should be checked to ensure that they are odd, (that the previous request has been completed); when odd, any error conditions present should be processed and the error bits cleared. Then, the request code (always an even value) should be set in bits 17-0 of FET word 1.

If automatic recall is selected when the request is made, the operation may be considered complete when RA+1 becomes zero; otherwise, the program must explicitly test word one of the FET for an odd value which signifies completion of the operation.

MISCELLANEOUS REQUESTS

Miscellaneous requests are: COMMON, REQUEST, MEMORY, CHECKPT, MESSAGE, TIME, CLOCK, DATE, JDATE, LOADER, READIN, and WRITOUT. The latter three do not generate PP requests directly; they merely serve as calling sequences to central processor library routines which are loaded and linked with the user program.

With the exception of MESSAGE, all the other requests are made through RA+1 with the appropriate program name and the first word address of a block of parameters. In several of the requests, the auto-recall bit must be set because of the nature of the request (the PP program must ascertain that the CP program is inactive).

All these requests have in common the fact that the first word of the block is set to an even value when the request is initiated; an odd value signals completion of the request.

The MESSAGE request is a special case. If the auto-recall bit in RA+1 is zero, the address in the low order 18 bits is considered to be the address of the message. If the auto-recall bit is set, the specified address is that of a pointer word in the following format:

Bits	**Contents**
59-48	Zero
47-30	First word address of message
29-0	Initially zero: set to odd value upon completion of request

CENTRAL PROGRAM CONTROL

The next highest level of user/system communication is that which is handled through a central processor library subroutine called CPC (Central Program Control). Calling sequences to CPC may be generated directly or through the use of the COMPASS system macro statements.

Several advantages accrue by using the CPC as opposed to setting RA+1 directly, and its use is highly recommended. By using CPC, COMPASS programmers are relieved of writing code each time it is necessary to check and set RA+1; also, code which uses CPC is considerably more readable and therefore easier to maintain. The somewhat complicated processing of MESSAGE requests is also simplified, as CPC will determine whether to generate an indirect address word based on the presence or absence of the auto-recall option.

Bit 41 of word 2 in the calling sequence of all the requests which generate PP calls is set equal to one. This bit is actually a flag in CPC which has no relevance to either MTR or the processing PP program. The setting of bit 41 causes CPC to recognize that the address given in A1 is not relevant, and that the word following the return jump to CPC contains a properly formatted request. No additional processing is done on these requests, except for MESSAGE. The request is simply placed in RA+1.

A request which utilizes an FET is signalled by a value of zero in bit 41 of word 2 of the calling sequence. CPC will, in this case, do considerable processing for the user. The processing basically consists of three steps: wait until the FET is inactive; process any abnormal conditions; and initiate the new request. The high order 18 bits of word 2 in the calling sequence may contain a numerical value rather than a PP program name. These values are of the form 2X + Y, where X represents the ordinal in a table of PP program names; and Y determines whether or not the FET must be inactive before processing can continue (Y = 0 or 1). A PP program name may appear in these 18 bits.

1. Upon receipt of a file action request, CPC will wait for previous activity on the specified FET to be completed unless the Y bit is one; CPC requests automatic recall until FET record 1 contains an odd value. The Y bit is 1 for READ, WRITE, and OPEN requests. If the request is OPEN, the assumption is made that no previous activity has occurred. READ and WRITE are handled specially.

2. If the Y bit is zero, the results of the previous operation will be tested. A zero in bits 9-13 of the FET code and status field indicates there are no abnormal conditions and processing goes to set 3. However, if there are abnormal conditions but no OWNCODE addresses are given, the contents of FET word 1 are saved for subsequent use as an exit parameter and processing goes to step 3. The error OWNCODE routine will be entered if bits 9-13 have a value of 4 or higher (end- of-information or end-of-reel may also be present); the EOI OWNCODE will be entered if the value is less than four. An OWNCODE routine is entered as though a return jump instruction was issued-execution begins at the start address plus 1. An exit from the routine will, however, return control to the main program, not to CPC, which indicates that the request which triggered this activity has not been issued; and the program must decide whether to re-issue it. An OWNCODE routine is entered with X1 containing word 1 of the FET complete with bits indicating abnormal conditions; FET word 1 itself has been cleared of the abnormal bits.

3. If the new request is for READ or WRITE and the FET is already active with the same request, CPC exits; it would be pointless to stop the I/O merely to reactivate it. If, however, the FET is inactive or active with a different request, steps 1 and 2 above will be executed as a subroutine. If the new request is a READ, an additional check will be made for end-of-record or end-of-file status on the previous request; the new READ will be ignored and an exit taken from CPC if either status is present. If a program is reading without recall, the user is forced to clear the EOR bit at the end of each RECORD to ensure that he is aware of the end-of-record.

CPC now will make preparations to communicate the new request to the system. The new request code from word 2 of the calling sequence is inserted into bits 17-0 of FET word 1; the old mode bit (bit 1) is not disturbed. The RA+1 request is formatted from the following items:

PP program name obtained either from the CPC calling sequence or from a table position specified in the calling sequence.

Setting of the auto-recall bit in the calling sequence.

First word address of the FET.

RA + 1 is set and CPC waits for a zero quantity to re-appear. If the auto-recall bit was set, CPC executes step 2 above as a subroutine. CPC then exits with X1 containing zero if no abnormal conditions were encountered; otherwise X1 will contain the value from FET word 1. CPC saves and restores all registers except A1, A6, X1, and X6.

RECORD BLOCKING/DEBLOCKING READIN/WRITOUT REQUESTS

The next level CP subroutine is designed to alleviate the necessity for the user to deal with the IN and OUT pointers in the FET. The FET for a file to be read or written by READIN or WRITOUT requests must be at least 6 words in length. Word 6 contains beginning and ending addresses of a workspace area. The user may process all data within the workspace area without concerning himself with circular buffer processing.

If the file is in binary mode, the length of the workspace determines the amount of data transferred to or from the circular buffer; a WRITOUT request will cause the workspace to be transferred intact to the circular buffer. If the file is in coded mode, the amount of data transferred depends on the presence or absence of trailing blanks.

A READIN request will cause data to be transferred to the workspace from the circular buffer until a zero byte is encountered; the remainder of the workspace area will be blank-filled. The converse happens in a WRITOUT; trailing blanks are removed and a zero byte is inserted.

The circular buffer will be filled or emptied, as needed, through calls to CPC. If a READIN request cannot be satisfied with data remaining in the buffer, a READ with recall is issued; likewise, if a WRITOUT request cannot be satisfied because of lack of buffer space, a WRITE with recall is issued. These actions ensure that a READIN or WRITOUT will always be complete when the user program regains control.

Immediately prior to returning to the user program, a further check on I/O progress is made. To buffer I/O with computing as much as possible, a READ (or WRITE) without recall will be issued if the buffer is not already busy and if the buffer is more than half empty or is full.

When a file is being read with READIN requests, end-of-record or end- of-file boundaries may be ignored if desired. When such a boundary is encountered, the filling of the wordspace will cease; and the user will regain control immediately with the contents of X1 indicating both the state of the workspace area and the type of boundary encountered. The user may take special action if he chooses, but he need not. The next READIN request issued will begin with the next record.

All data entered into the buffer with a WRITOUT request is considered to be part of a single logical record, and the buffer will be emptied automatically as needed. If the file is to be grouped into several logical records or when the buffer is to be emptied at the end of a run, a WRITER request must be issued. A WRITER request goes directly to CPC and causes the buffer to be emptied and an end-of-record mark written. A WRITEF request may also be issued, if desired.

RANDOM ACCESS READIN/WRITOUT REQUESTS

The highest level CP subroutine, IORANDM, deals with random access disk files with specially formatted indexes. (Other forms of index may be devised and used by writing another CP subroutine similar to IORANDM). IORANDM also is called with READIN and WRITOUT requests but with an additional parameter, either a record name or record number.

A random WRITOUT request is issued when a new record or a replacement record in a random file is to be started. If the FET shows a previous status of write completed rather than write end-of-record completed, IORANDM will first terminate the preceding record by issuing a WRITER request. Then, if a workspace area is defined, data will be located in the index if it is to replace an existing record; or an empty position will be located if the record is a new one.

The index address is inserted into FET word 7; a disk table pointer will be written into the word so addressed by the PP input/output routines. A WRITE then will be issued through CPC. Thereafter, writing in the record may be continued by issuing subsequent calls to (non-random) WRITOUT if a workspace area is being used; or by refilling the circular buffer directly and issuing WRITE requests. In either case, the record is terminated when a WRITER request is issued.

A random READIN request is issued to position to another record in a random file. The current contents of the circular buffer are regarded as irrelevant; the IN and OUT pointers will be set equal, thereby logically destroying any data in the buffer. The index will be searched for the desired record, and the disk table pointer from the index will be entered in FET word 7. A READ request with recall will be issued to CPC. Upon completion of the read, the workspace area, if defined, will be filled; and IORANDM will return to the user. Thereafter, reading of the record may be continued simply by subsequent calls to (non-random) READIN if a workspace area is being used, or by issuing direct READ requests to refill the buffer. End- of-record boundaries are respected in a random file; when an end-of-record mark is reached, the user must re-position with a random READIN request. Any attempt to read without re-positioning will be ignored and end-of-record status returned.

Both random READIN and WRITOUT will enter the index address of the current record into FET word 8, which enables the user program to request the next record to be read or written. The next record is defined as the next index position and does not necessarily have any relation to the next physical record in the file. The next record in the index is requested by specifying a zero value as the record name/number in the random READIN or WRITOUT request. The zero value must explicitly appear as different calling sequences are generated depending on the record number/name parameter.

USING CPC

Both file and system requests call the Central Program Control subroutine (CPC) which provides linkage with the monitor (MTR). Before CPC can honor a file action request, the file environment table (FET) must have been established for the file to be processed. Calling sequences to CPC may be generated either directly or through the use of system macro statements.

The user program communicates with CPC through macro requests and the FET. Communication with SCOPE is handled by CPC through setting and checking RA+1. CPC may also cause the execution of one or more user subroutines for which addresses are specified in the FET. Such a subroutine is entered at the address given in the FET+10(octal).

The exit from CPC is stored at the OWNCODE routine given in the FET; (X1) = the first word of the FET.

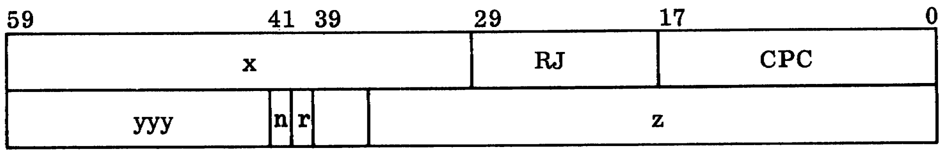
A normal exit is made if the request is honored and no error conditions occur. X1 contains word 1 of the FET upon exit if the status is other than request completed.

The status known as recall is provided to enable efficient use of the central processor and to capitalize on the multiprogramming capability of SCOPE. Often a CP program must wait for an I/O operation to be completed before it can do any more computation. It would be a waste of CPU time to have the CP program go into a loop waiting for I/O completion. To avoid this waste, a CP program can ask SCOPE to put the control point into recall. Essentially this means that the CP program does not need the CPU until later. In the meantime, SCOPE can assign the CPU to execute some other CP program in central memory.

The two types of recall are automatic recall (auto-recall) and periodic recall. Automatic recall should be used when the program makes an I/O or system action request, and it cannot proceed until that request is satisfied. SCOPE will not return control to the program until that request is satisfied. Periodic recall can be used when the program is waiting for any one of several requests to be satisfied. In this case, SCOPE will activate the program periodically so that it can determine whether or not it can proceed.

CALLING SEQUENCE

Format of the calling sequence to the central program control subroutine:



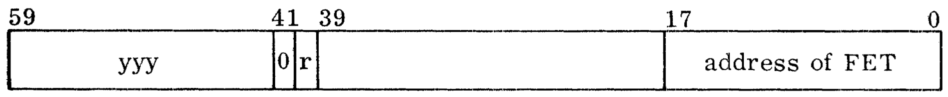
RJ	Return jump instruction
CPC	Entry point to the CPC subroutine
n=0	File action request flag bit

yyy	Display code characters CIO, or
000001	if only file recall is desired. Display code characters RCL are generated.
000002	for most read or write functions. A function in progress will not be reissued by CPC. When the file becomes inactive, CPC issues the next request. Display code characters CIO are generated.

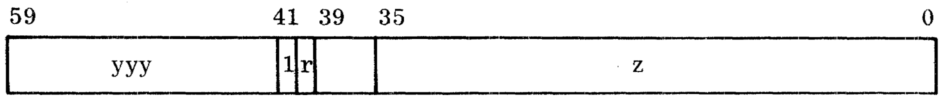
000003	for all other functions. When the file becomes inactive, CPC issues the request. Display code characters CIO are generated.
000004 or 000007	equivalent to 000003; included only for compatibility with previous systems.
x	SA1 base address of FET
z	Request code

n=1	System action request flag bit	
	yyy	Display-coded name of the called PP program
	x	not relevant
	z	parameters as required
r=1	Issue request and enter RECALL	
r=0	Issue request and return control to the program	

A file action request to the SCOPE monitor is formatted by CPC in RA+1 as follows:



A system action request to the SCOPE monitor is formatted in RA+1 as follows:



z appears in the buffer code and status field of the FET.

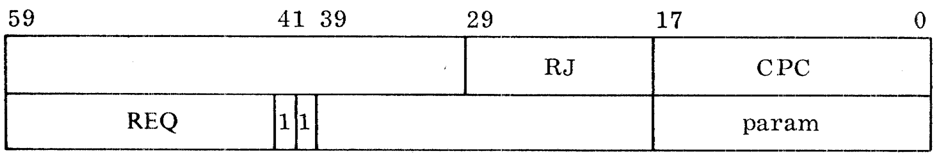
Bits not specified in the calling sequence are reserved for future system use.

FILE ACTION REQUESTS

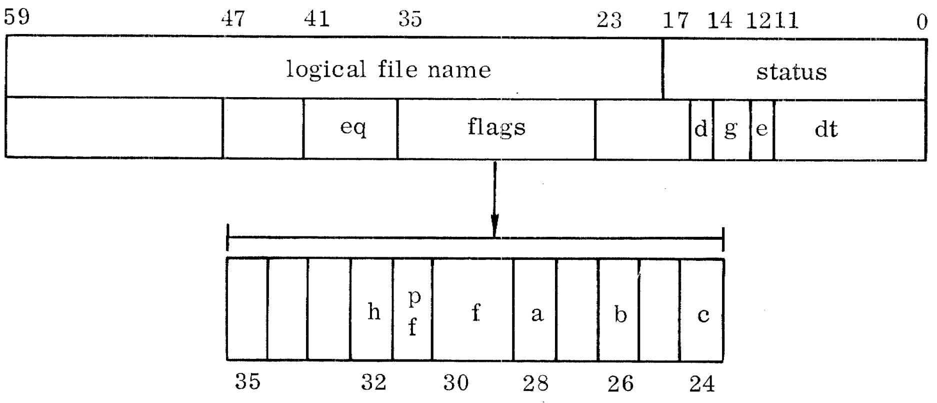
In the following descriptions, the system macro is followed by the macro expansion.

File action requests result in a return jump to the central program control subroutine. Subsequent actions depend on the state of the file. An OWNCODE routine may be executed and/or a request to SCOPE may be posted. File action requests will be posted but not honored if the EOI or error bits are set, and OWNCODE addresses are present where a call is issued to CPC. In either case, control returns to the calling program after SCOPE accepts the request if the recall bit, r, is equal to zero; or after SCOPE completes the request if r is equal to one. If the optional recall parameter is non-blank, r is set to one.

REQUEST param



Assignment of equipment may be requested from a running central processor program by the REQUEST function, which has the same effect as a REQUEST card. Param is the first word address of a two-word parameter list, which is of the form:



Parameters written after lfn may appear in any order. Upper case characters designate subfield content; lower case characters indicate parameters to be supplied by the user. All parameters except lfn are optional.

lfn	Logical file name (or multi-file set name if a multi-file tape is to be assigned).
dt	First character in parameter represents octal device type; lower 6 bits second character represents recording technique.
eq	As described under equipment ordinal on REQUEST card.
pf	Set to 1 to request a mass storage device for a file to be cataloged subsequently as a permanent file. If set, the upper 6 bits of the dt field will be ignored.

If magnetic tape device type is specified, characteristics of the device may be declared with the following settings:

Bits	**Characteristics**
a	1 for automatic assignment
b	1 if two devices are requested
c	1 for Existing status, 0 for New status
d	1 for system default density
e	1 if a checkpoint dump file is to be requested
f	01 for EBCDIC character set for 9-track tapes 10 for ASCII character set for 9-track tapes
g	01 for inhibit unload disposition 10 for save disposition
h	1 indicates special device overflow action. See device type OV parameter on REQUEST control card.

The status field is initialized to zero. Bit zero is set to one when the function is completed. In addition, the following octal values may be returned in bits 9-13 of status:

22	REQUEST function was issued without the recall bit
23	Device type of declared equipment ordinal does not match dt or dt is a non-allocatable device.
24	FNT is full
26	Either requested equipment does not exist in the configuration, or all equipment of this type is already assigned to this control point.
30	File is already assigned to a device; the actual device type will be returned in the dt field.

If any of the above conditions occur, the REQUEST function is ignored and control is returned normally.

Two dayfile messages result from a successful REQUEST function. The first is directed only to the operator and contains parameters corresponding to those used in the internal parameter list. After assignment, a second REQUEST message is written to the job and system dayfiles reflecting the assignment. For example, if a REQUEST function is made with dt set to zero, the operator display will show no device type. If the operator assigns a 7-track tape, however, the mnemonic MT will appear in the job dayfile message.

OPEN lfn,x,recall

59	47	41		39	29	17 0
SA1	lfn				RJ	CPC
000004		0	r			z

Values are given in octal for the following parameters.

If x is absent, z = 160 (open with rewind)

The following will be accepted on a source or binary level, but they will be handled as though they were 160.

x = READ, z = 140
x = WRITE, z = 144
x = REEL, z = 340
x = ALTER, z = 160

If x = NR, z = 120 (open with no rewind)

The following will be accepted on a source or binary level, but they will be handled as though they were 120.

x = READNR, z = 100
x = WRITENR, z = 104
x = REELNR, z = 300
x = ALTERNR, z = 120

If x is WRITE, z = 144 (open/write with rewind)

If x is WRITENR, z = 104 (open/write with no rewind)

The OPEN-WRITE and OPEN-WRITENR functions may be used to ensure that the file circular buffer will be written out if the job terminates before a write function is issued. Read and forward position functions are illegal on files opened with WRITE and WRITENR until a backward motion has occurred to clear the write bit in the FNT.

Issuing an OPEN function will cause action in one or more of the following four areas. An OPEN function is required if any of the following actions are to be performed on a file; otherwise OPEN is not necessary.

File Positioning

If the NR parameter is absent, sequential files will be positioned at begining-of-information. For magnetic tape files, beginning-of-information is defined to be before the first data record on the current reel. If NR is specified, the file position is not changed.

Positioning is not defined on files for which a SCOPE index is read.

SCOPE Index Processing

The file index is read into the index buffer if one is specified, the FET r bit is set, and the file has an index. (see RANDOM ACCESS). If the index record is shorter than the buffer area, the unused part of the buffer is set to zeros.

FET information

Values are returned to the device type, disposition code, and FNT pointer fields in the FET by the system when any file action request is issued. For physical record unit size, record block size, and file label fields, values will be returned in the FET by an OPEN function. If the file is not of random format, the FET r bit will be set to zero during OPEN processing.

Circular Buffers and Buffer Pointers

The circular buffer and buffer pointers are affected by OPEN processing only if a tape file header label is read. The label is delivered to the buffer beginning at the current value of the IN pointer, and the IN pointer is moved to reflect the presence of the label; unless there is insufficient room between the initial value of IN and LIMIT, in which case this action will not occur.

POSMF lfn,recall

59	47	41	39	29	17 0
SA1	lfn			RJ	CPC
000003		0	r		000110

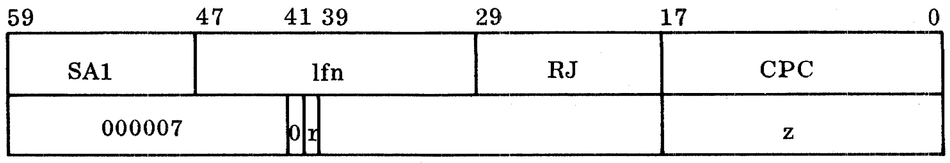
The POSMF function directs SCOPE to position to a particular member file within a multifile set, and to perform an open rewind on that file. The multifile set name is given in the lfn field of the FET. The position number, obtained from word 11 of the FET, determines the member file to be opened. This 1 to 3 digit number may be set or changed with the LABEL utility (section 8).

If the position number in the FET is blank, the set will be positioned to the next file (current file + 1) and open procedures for an existing file will be performed. If the position number = 999, the set will be positioned after the last member file, and open procedures for a new file will be performed. If the set was declared to have new status, the first POSMF may be issued only if the position number in the FET = 999. Otherwise, the set will be positioned to the specified member, and open procedures for an existing file will be performed.

If the explicit or implied position number of the requested member file is greater than the last member of the set, end-of-set status will be returned to the FET.

If the requested member file is not found on the current reel, the operator will be requested to mount a new reel.

CLOSE lfn,x,recall



Values are octal:

If x is absent, z = 150, close with rewind
If x = NR, z = 130, close with no rewind
If x = UNLOAD, z = 170, close with rewind and unload
If x = RETURN, z = 174, close with rewind and unload

The action caused by CLOSE functions is defined below. Issuance of a CLOSE without unload does not prevent further action on the file; closed status is not possible.

File Positioning

If no rewind is specified, the file position will not be altered. Otherwise, the file will be rewound (and unloaded if CLOSE- UNLOAD or CLOSE-RETURN).

Positioning is not defined on files for which an index is read.

SCOPE Index Positioning

The index will be written as the last logical record of the file, if the FET r bit is set, an index buffer is specified, and the file contents have been altered (see RANDOM ACCESS). A user should close all indexed random files at the end of each run to ensure that the index reflects the latest condition of the file.

Label Processing Magnetic Tape Only

If no rewind is specified and the file is positioned after a newly written record, a file mark and an EOF trailer label will be written and the file will be positioned immediately before the file mark. If CLOSE with REWIND, UNLOAD, or RETURN is specified and the file is positioned after a newly written record, an EOF trailer label will be written before the rewind is initiated.

All variations of CLOSE will cause one or more of the above actions. In addition, CLOSE-UNLOAD and CLOSE-RETURN will cause end-of-job procedures for the specified file to be performed as follows:

Non-allocatable equipment assigned to a local file is released. The unload option returns magnetic tapes to the control point pool; the return option returns them to the system pool.

Local allocatable file space is released.

Common files are returned to control point zero.

Local allocatable files with a non-zero disposition code are entered in the output queue.

Permanent files are detached from the control point.

Private pack files are locked and made unavailable to the job.

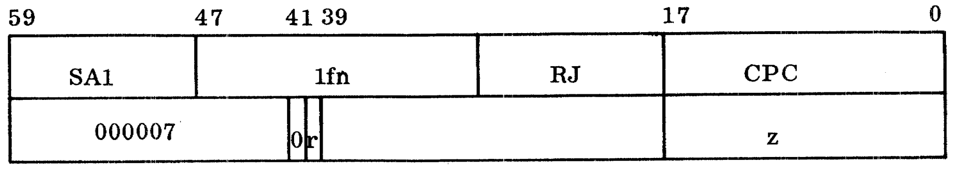
The job is flagged as no rerun possible if:

1. The file was attached as an existing common file and is being detached as a common file or as a local file.

2. The file was created by the job as a local file and is being detached as a common file.

If the file is a member of a multi-file set, no special action is taken for a CLOSE-UNLOAD or CLOSE-RETURN. Equipment is not released.

CLOSER lfn,x,recall



x is effective only if the UP bit is on when the CLOSER function is issued. Values are octal.

If x is absent, z = 350, current reel is rewound

If x = NR z = 330, accepted on a source or binary level but handled as though it were 350

If x = UNLOAD z = 370, current reel is rewound and unloaded.

Under SCOPE, reel boundaries can be crossed automatically when data is skipped in a forward direction, read, or written. With the CLOSER (close reel) function the user can request notification when a reel boundary is about to be crossed or if reels are to be processed in other than ascending order.

The CLOSER function affords a degree of user control over processing at end-of-reel on magnetic tape, depending on the setting of the UP bit when CLOSER is issued. Processing of a tape reel also can be terminated prematurely.

If the value of the UP bit is zero when a CLOSER function is issued, the system will initiate reel swapping. The file will be positioned on the next reel and file operations can continue normally. An OPEN REWIND function may be issued if the user program requires delivery of the header label; otherwise OPEN is not necessary. A reel swap is performed by the following steps:

1. If the tape is positioned after a newly written record, a volume trailer label will be written.

2. The tape is unloaded and the operator notified that processing on that reel is completed.

3. If two units were assigned to the file, unit numbers will be switched.

4. The reel number of a labeled file will be incremented by one in the system label table and, if declared, in the user's FET label fields.

5. The FET completion bit is set. End-of-reel status is not returned.

By setting the UP bit to one when the CLOSER function is issued, the user may specify the next reel to be processed. SCOPE then performs the following:

1. If the tape is positioned after a newly written record, a volume trailer is written.

2. The tape is rewound or rewound/unloaded according to the CLOSER parameter.

3. The operator is notified that processing on that reel is completed.

4. If two units were assigned to the file, unit numbers will be switched.

5. The end-of-reel status and completion bits are set.

It is then a user responsibility to enter the reel number into the FET label field before the next function is issued to the file. An OPEN function should be issued if the user program requires delivery of the new header label; otherwise OPEN is not necessary.

EVICT lfn,recall

59	47	41	39	29	17 0
SA1	lfn			RJ	CPC
000003		0	r		000114

The EVICT function declares that contents of file lfn are to be discarded.

When a file on an allocatable device is evicted, all space occupied by that file is released to the system. The space immediately becomes available for use by the releasing program or other programs. An EVICT function directed to a permanent file is ignored; a dayfile message is issued; and the job continues normally.

When a file on a magnetic tape is evicted, the tape is rewound and set to new status; thus declaring that the data and label are no longer valid and cannot be read by the job. If the file was declared to be labeled, a new header label will be written on any subsequent file reference, as defined under LABEL.

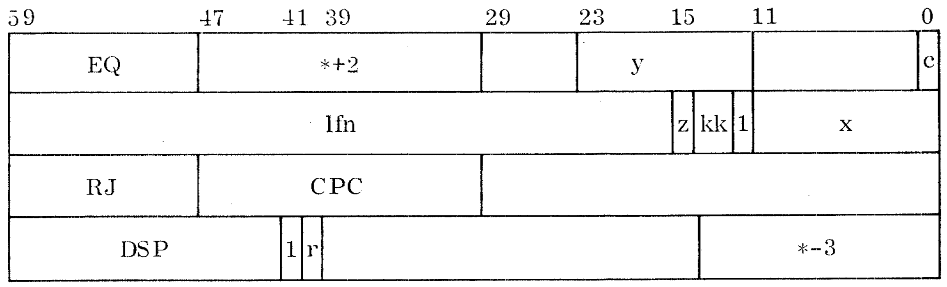
If an EVICT function is directed to a member of a multi-file set, the set must have been already positioned at that file. Eviction of a member file also implies eviction of all files occupying higher numbered positions.

The logical file name in any case, is retained.

EVICT is undefined and, therefore, illegal on unit record equipment. A fatal error results if it is tried.

FILE DISPOSITION

DISPOSE lfn,x=ky,recall or *DISPOSE lfn,*x=ky,recall*



With the DISPOSE function, a CP program may declare a file disposition code and initiate job termination for the file lfn. The parameter lfn must be the actual character string representation of the logical file name for the file being disposed.

c = completion bit, set to 1 by CPC when the file has been detached from the job.

z is set to 1 when *x is used; indicates file is to be disposed at end of job.

x is one of the following two-character disposition codes:

PR	Print
P1	Print on 501 or 505
P2	Print on 512
PB	Punch binary
PU	Punch coded
P8	Punch 80 columns
FR	Print on microfilm recorder
FL	Plot on microfilm recorder
PT	Plot
HR	Print on hardcopy device
HL	Plot on hardcopy device

Driver is not provided for the last five above dispositions.

kk=00 if no site designator specified
01 if C specified (output at central site)
10 if I specified (output at INTERCOM site)
11 if E specified (output at EXPORT/IMPORT site)

C designates output at the central site, E designates an EXPORT/IMPORT terminal, and I designates an INTERCOM terminal.

If kk=C, y is an alphanumeric, 2-character, installation-defined identifier.

If kk=I, y is a 2-character, alphanumeric user identifier.

If kk=E, y is a 1- or 2-character, alphanumeric terminal identifier.

CONTRLC addr

59	41		39	29	17 — 0
				RJ	CPC
ACE	1	1			addr

With the CONTRLC function, a CP program can read or backspace the control card record. addr is the address of a word in the following format:

59	17 — 0
Reserved for system use	code

code is both the function and status reply as follows; values are octal:

READ code = 000010

The next control card is placed in RA + 70 to RA + 77 with zeros in the space remaining between the end of the card and RA + 100.

When the function is completed, bit zero of the code is set to one. If on the read there are no more control cards, the area RA + 70 to RA + 77 is set to zero and bit 4 is set to one for end of record.

BKSP code = 000040

The control statement pointer is reset to the previous statement.

If the record is set to the start of the control card record when a BKSP function is issued, the statement pointer is unchanged and the end of record reply is issued.

With this request the user can position the control card record for SCOPE job processing. If he does not position it properly, for example in front of an EXIT card, results are unpredictable.

DATA FUNCTIONS

The following data functions apply equally to magnetic tape and mass storage files unless otherwise indicated.

READ lfn,recall

59	47	41		39	29	17 — 0
SA1	lfn				RJ	CPC
000002		0	r			000010

This function reads information into the circular buffer if the specified file is open. If there is room in the circular buffer for at least one physical record unit, reading is initiated and continues until:

Buffer does not contain enough room for the next physical record.

End-of-record or end-of-file is encountered.

End-of-information is encountered.

An error is encountered (see FET, section 3).

For S or L tapes, one physical record is read.

Mode is determined by bit 1 in the first word of the FET. If end- of-record (bit 4) is set upon entry to CPC, no operation is performed. For S or L tapes, the unused bit count is returned to the UBC field in FET word 7.

READN lfn,recall

59	47	41		39	29	17 0
SA1	lfn				RJ	CPC
000002		0	r			000260

S or L magnetic tape only: The READN function reads data from tape to the circular buffer until one of the following conditions occurs:

Buffer does not contain enough room for next record (MLRS+1 words), end-of-file is encountered, end-of-information is encountered, error is encountered.

The mode of the file is determined by bit 1 in the first word of the FET. Word 7 of the FET must contain the size of the largest possible logical record when this function is used.

READN enables non-stop reading for maximum tape throughput, only for S or L tapes. As long as the user provides sufficient room in his buffer (room for two records and their header words), tape reading continues without releasing and reloading the PP between logical records; and maximum utilization of interrecord gap time is realized. The concept of circular buffering is retained; however, a header word is placed in the buffer along with data from each logical record. It precedes the data and contains the number of CM words in the logical record and the number of unused bits in the last data word. IN is moved by the I/O system after a complete logical record together with its header word have been placed in the buffer. The format of the header word is:

59	29	23	17 0
	No. Bits unused		#CM words

READSKP lfn,lv,recall

59	47	41		39	29	17	13 0
SA1	lfn				RJ	CPC	
000003		0	r			lv	00020

READSKP functions like READ, except when the buffer is filled prior to the end-of-record, then the rest of the information is discarded and the file is positioned to read the next logical record. If the MLRS field in the FET is zero, it is set to 512 words for S tapes and for L tapes to LIMIT- FIRST-1.

If a level parameter lv is specified, information is skipped until the occurrence of an end-of-record with a level number greater than or equal to the one specified. Only a level 17 (EOF) is recognized by S and L tapes; any other level parameters in the request will be ignored.

Executing a READSKP sets the end-of-record (bit 4) to 1, since an end- of-record is encountered. If the next operation on the file is READ, the EOR bit must first be set to zero by the calling program.

The SKIPF portion of a READSKP function is considered to be complete if an end-of-reel condition arises on a magnetic tape file for which the UP bit is set.

RPHR lfn,recall

59	47	41		39	29	17 0
SA1	lfn				RJ	CPC
000003		0	r			000000

Magnetic Tape (SCOPE standard tape)

The RPHR function causes any information already in the buffer to be discarded by setting the OUT pointer equal to the IN pointer; then the next physical record is read into the buffer. The mode is determined by bit 1 of the first word of the FET. For coded files, only conversion from external to internal BCD is performed. If the data read does not fill an integral number of CM words, the last word is filled with zeros.

READNS lfn,recall

59	47	41		39	29	17 0
SA1	lfn				RJ	CPC
000002		0	r			000250

MASS STORAGE FILES ONLY

READNS operates in the same way as READ, except that it is not ignored by CPC if the last code/status in the FET is 02X or 03X, and reading does not necessarily stop at the end of a logical record. Instead, a READNS operation terminates under any of the following conditions:

1. Next PRU will not fit into the circular buffer.

2. A zero length logical record (any level) has been read.

3. A level 16 or 17 logical record (any length) has been read.

4. End-of-information has been encountered.

5. An irrecoverable error is detected; the error code ee is usually 04 for parity error.

The status stored in the FET for each case is as follows:

case 1	000011 coded, 000013 binary
case 2 or 3	740031 coded, 740033 binary
case 4	741031 coded, 741033 binary
case 5	0ee011 coded, 0ee013 binary

In cases 2 and 3 the level of the terminating logical record is lost.

READIN lfn,x

READIN may be used for indexed or sequential mass storage files or tape files. The format depends on file mode, file index, working storage area, and the x parameter as shown in the following pages. In the descriptions, n represents the number of words in the working storage area.

READIN stores the next n words from the circular buffer of file lfn into the working storage area; a READ request is issued if the buffer is empty. If the file is binary mode, READIN attempts to fill the working storage area until end-of-record or end-of-information is encountered. For a coded file, information is moved to the working storage area until a zero byte (end-of-line) is encountered or until the working storage area is full. When a zero byte is encountered, two blanks are substituted and the remainder of the working storage area is filled with blanks. If a zero byte is not encountered before the working storage area is full, the remainder of the line is skipped and a subsequent READIN request reads the next line.

The status of the request is returned in X1 as follows:

+0	Requested number of words was read and the function completed normally.
positive nonzero	Fewer than n words remained in the logical record when the request was issued. When control is returned to the user program, X1 contains the last address + 1 of the data transferred to working storage or first word address if no data was trans- ferred. For coded files, this is always the first word address.
negative nonzero	If end-of-information or end-of-file is encountered, X1 contains a negative number. No information is transferred into the working storage area.

If a working storage area is not specified, a READIN request has no effect; and no error indication is given unless it addresses a file with a name or number index. In that case, the effect of the request will be to terminate any previous action on the file and locate the specified logical record; the next READIN request to transfer data from that file will begin with the first word of the specified record.

READIN lfn (x is absent)

59	29	17 0
	RJ	IOREAD
		lfn

This form of the READIN request may be used for tape or mass storage files; it transfers data from buffer to working storage area; if the buffer is empty a READ is issued. If lfn is a mass storage file, it may be sequential or random.

READIN lfn,/name/ (x is of the form /name/)

59	29	17 0
	RJ	IORR
		lfn
name		

With this form of the READIN request, logical record /name/ on the mass storage file named lfn is read into the circular buffer. n words are transferred to the working storage as described above. The file must have a name index.

READIN lfn,m (x is m, a logical record number)

59	29	17 0
	RJ	IORR
		lfn
		m

The logical record number m of the file, lfn, is read into the circular buffer. n words are transferred to the working storage area as described above. The file must be indexed by name or number. If m is zero, the next record is read. The next record is the first logical record of the file if this is the first request, or the last logical record read + 1, for any subsequent READIN.

WRITE lfn,recall

59	47	41		39	29	17 0
SA1	lfn				RJ	CPC
000002		0	r			000014

When this function is called, information is written from the circular buffer. For mass storage files and SCOPE standard tapes, only full PRU's are written. Writing continues until the buffer is empty or there is not enough data in the buffer to fill a PRU. For S and L tapes, only one record is written.

For S or L tapes, if the requested record length, indicated by the OUT and IN pointers, is greater than MLRS, the job is terminated with the message DATA EXCEEDS MLRS.

WRITER lfn,lv,recall

59	47	41		39	29	17	13 0
SA1	lfn				RJ	CPC	
000003		0	r			lv	00024

This function is processed the same as WRITE, with the following exceptions:

> For mass storage files and SCOPE standard tapes, data in the circular buffer is written out and terminated by a short or zero-length PRU to indicate end-of-record. If no information is in the buffer, a zero-length PRU is written. If the specified level number is 17, the function will be changed to WRITEF.

If the level parameter (lv) is present, the short or zero-length PRU will reflect the level number. If the level parameter is absent, the field is set to 0 and level zero is assumed. This field is ignored for S and L tapes.

WRITEF lfn,recall

59	47	41	39	29	17 0
SA1	lfn			RJ	CPC
000003		0	r		000034

For SCOPE standard tapes and mass storage, the WRITEF function produces a logical end-of-file mark; it is written as a zero-length physical record of level 17. Any data present in the buffer is written and terminated with a level zero end-of-record. If the buffer is empty and the last operation was WRITE, a zero length PRU is written.

For an S or L tape, if data in the buffer is less than or equal to MLRS, it is written to tape followed by a physical tape mark. If data in the buffer exceeds MLRS, the job is terminated with the message DATA EXCEEDS MLRS.

WRITEN lfn,recall

59	47	41	39	29	17 0
SA1	lfn			RJ	CPC
000002		0	r		000264

MAGNETIC S AND L TAPES ONLY

WRITEN improves throughput by efficient utilization of interrecord gap time. As long as the user provides data ahead of the I/O system, tape writing continues without releasing and reloading the PP between logical records, making full use of the interrecord gap on tapes (as long as the circular buffer contains at least two records). The concept of circular buffering is retained. However, a header word must precede each logical record in the buffer. This header word gives the number of CM words in the logical record and the number of unused bits in the last data word. OUT is moved by the I/O system after a complete logical record has been written to tape. Writing continues until the buffer is empty or until an end-of-reel or error condition is detected. No writing will take place if the buffer is empty. The user should not move IN beyond the header word until the header and the complete record are in place. An error will result if SCOPE detects this condition. The format of the header word is:

59	29	23	17 0
	No. Bits unused		#CM words

WPHR lfn,recall

59	47	41	39	29	17 0
SA1	lfn			RJ	CPC
000003		0	r		000004

SCOPE STANDARD MAGNETIC TAPE ONLY

This function causes information in the circular buffer to be written as a single physical record on a magnetic tape. Mode is determined by bit 1 in the first word of the FET.

If the buffer contains fewer than 512(decimal) words, IN and OUT pointers in the FET are set to the same value at completion of writing to indicate an empty buffer. Only internal to external BCD conversion is performed. If the buffer contains more than 512 words when the request is issued, the first 512 words are written; and IN and OUT pointers are set to show that words remain in the buffer. Status 10 is returned (device capacity exceeded).

A WPHR function issued for any device other than 1/2-inch magnetic tape in SCOPE standard format is ignored, and status 22 (illegal function) is returned.

WRITOUT lfn,x

WRITOUT transfers n words from the working storage area to the circular buffer, adding them to the logical record currently being constructed. If there is no current record, they become the first words of a new logical record. If the file is indexed, however, the request is rejected, because the system cannot determine which record of the file is addressed. A WRITE request is issued automatically when the buffer is full.

With this function, writing depends on file mode and the presence or absence of a file index, a working storage area, and the x parameter. In the following paragraphs, n represents the number of words in the working storage area.

For a binary mode file, the entire working storage area is transferred to the circular buffer. In coded mode, trailing blanks are removed; and a zero byte (end-of-line) is inserted as data is transferred to the buffer. The WRITER function may be requested to terminate record writing; but if the file is indexed, and no record is being written, the request is rejected.

If a working storage area is not specified, a WRITOUT request has no effect and gives no error indication, unless it addresses a file with a name or number index. In that case, this request terminates any previous action on the file, locates the specified logical record, and sets up the pointers so that the next request to continue writing the current record on that file will begin with the first word of the specified record.

WRITOUT lfn (x is absent)

59	29	17 0
	RJ	IOWRITE
		lfn

Data from the working storage area is added to the logical record currently being constructed; if there is no current logical record (no WRITOUT has been executed), this request is rejected as the system cannot determine which record is addressed.

WRITOUT lfn,/name/   (x is of the form /name/)

59	29	17 0
	RJ	IORW
		lfn
name		

Data is transferred from the working storage area and is added to the logical record currently being constructed; if the circular buffer is empty and the file is indexed, this request will be rejected since the system cannot determine which record of the file is to be addressed.

WRITOUT lfn,m
(x is of the form m, which is a logical record number)

59	29	17 0
	RJ	IORW
		lfn
		m

This form of the WRITOUT request begins constructing logical record number m on the file named lfn, using the words in the working storage area as the dst words of the record. The file must be indexed, either by name or by number. If m = 0, the request will address the record with a number one higher than that of the record most recently addressed, or record number 1 if the file has not been addressed. The first record of an indexed file is number 1; there is no record number 0.

Requests 2 and 3 above perform the following functions:

1. Perform a WRITER on the file if its previous status was write, or the buffer contains data as a result of a previous WRITOUT.

2. Set the buffer to empty, and the FET status to write completed.

3. Set up the random file index and FET to point to the correct record.

4. Transfer the working storage area to the buffer.

5. Call WRITE if the buffer contains at least one PRU of data.

The WRITOUT lfn, m or /name/ statements are used only to begin an indexed record. Records can be continued with WRITOUT lfn statements, and should be terminated by:

WRITER lfn

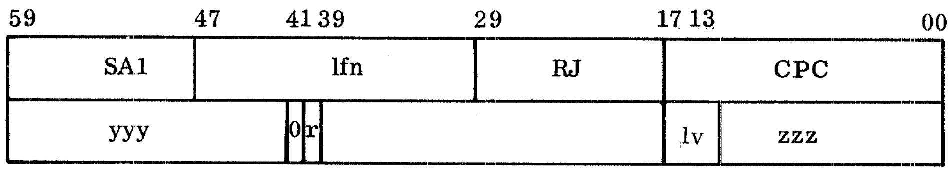
If this step is neglected, however, the next WRITOUT lfn,m or /name/ statement for the same file will be treated as in step 1 above.

If the working storage area size is zero, nothing is transferred to the buffer; but steps 1, 2, 3 and 5 above are carried out.

REWRITE lfn,recall

REWRITER lfn,lv,recall } Mass Storage Only

REWRITEF lfn,recall



yyy	002 for REWRITE 003 for REWRITER and REWRITEF
zzz	214 for REWRITE 224 for REWRITER 234 for REWRITEF

The REWRITE functions make use of a previously allocated file on a mass storage device to update records in an existing file without changing its index or mass storage allocation. They should be used to replace records in sequential or random indexed file by records of the same length. A knowledge of the structure and record lengths of the file to be rewritten is essential. If the length of the rewritten record differs from the original record, results are unpredictable. The system cannot determine the length of the original record; no protection is afforded from over or underwriting, nor is a diagnostic issued. The system guarantees only that a rewritten record does not extend beyond the end-of-information. The write takes place up to end-of-information, and end-of-information indicators are not moved; a diagnostic is issued; but the user may destroy an index previously written on the file.

Files are rewritten with information from the CIO buffer, and writing always begins at the current position of the file. (REWRITE transfers a minimum of one PRU of data.) For REWRITER, an end-of-record, level number lv is written at the end of the data transferred. For REWRITEF, a logical end-of-file (level 17) is written at the end of the data transferred. After REWRITEF is used, other data in the file remains allocated to the file; therefore an end-of-file may occur in the middle of the file. Rewrite operations do not change the storage allocation of the file; no additional record blocks are assigned, nor is the index of a random file modified. It is, in effect, a write-in-place operation. If rewrite is used for a common file when the job is rerun, the data of that common file will not be the same as when it was previously used by that job.

Application of Rewriting for Sequential Files

Rewriting begins at the current file position. After each request is executed, the current file position is updated to point to the PRU that follows the last PRU rewritten by the request. A series of REWRITE operations, followed perhaps by a REWRITER, can be used to replace a single logical record.

When a logical record is to be replaced with a record of the same length in a single rewrite operation, REWRITER should be used.

When a record is replaced with a shorter record, the new record might be followed by another record which is the last part of the original record. For example, if the original record is 120 words (one full length and one short PRU) and it is replaced by REWRITER with a 60-word record, the end result is one 60-word record followed by another 56-word record (a short PRU) which is the second PRU of the original record.

If a logical record is replaced with a longer record, the new record will overwrite more than one record, and part of the surplus might appear as a logical record that immediately follows the new record. End-of-information, however, is not destroyed.

Application of Rewriting for Random Indexed Files

Rewriting begins at the current file position. A logical record may be replaced by a single REWRITER or by a series of REWRITE's followed by a REWRITER. The programmer must reposition the file for each logical record to be rewritten. Otherwise, records will be rewritten in their original order—not in index order.

To position an indexed file for rewrite of a particular record, the user may use one of two methods:

> Set up the file's FET the same as for a random READ; insert into the Record Request/Return Information field in FET+6 the record address found by searching the file's index.
>
> Use the WRITIN function, which causes the system to search the user's index and set the necessary FET fields. (WRITIN will also issue a REWRITE request in some cases. Refer to the WRITIN description for detail).

Once the file is positioned for a record, a REWRITE/REWRITER sequence or a WRITIN/REWRITER sequence can be executed without further repositioning. The FET+6 field will be cleared by the system after the first REWRITE (or after WRITIN) and will remain cleared until repositioning for another record is required.

The methods of rewriting, and the results of underwriting or overwriting a logical record are the same as for sequential files. Index integrity, however, can be destroyed by the user if he writes shorter or longer records than those in the original file. A longer record destroys records originally written just after the one being replaced (not necessarily records which appear next in the index). A shorter record may create an extra record which is not represented in the index. The index is never modified by the system for REWRITE. The PRU's and RB's allocated to the file remain assigned, though they might contain some useless inaccessible data as a result of rewriting with shorter records. Such unused areas can be utilized later when a longer record (but shorter than or equal to the original) is rewritten. Use of such techniques requires an understanding of the file's logical and internal structure.

WRITIN lfn,x

WRITIN provides automatic index and working storage area management for REWRITE. WRITIN is a write-in-place function, unlike WRITE which writes at end-of-information. It may be used for indexed or sequential mass storage files. The results of a WRITIN request depend on file mode and the presence or absence of a file index, a working storage area, and the x parameter. In the following paragraphs, n represents the number of words in the working storage area.

WRITIN transfers n words from the working storage area to the circular buffer, putting them into the area where the file is currently positioned. A REWRITE request is issued automatically when the buffer is full.

If the file is in binary mode, the entire working storage area is transferred to the circular buffer. If the file is in coded mode, trailing blanks are removed and a zero byte (end-of-line) is inserted as the data is transferred to the buffer. The REWRITER function may be requested to terminate writing of a record.

If a working storage area is not specified, a WRITIN request has no effect and gives no error indication, unless it addresses a file with a name or number index. In that case, the request will terminate any previous action on the file, locate the specified logical record, and set up the pointers; so that the next request for REWRITE, REWRITER or another WRITIN (without x parameter) to continue writing the current record on that file will begin with the first word of the specified record. If the next request is WRITOUT or WRITIN, the current logical record will be terminated by execution of a REWRITER of level 0 before the function is executed.

WRITIN lfn (x is absent)

59	29	17 0
	RJ	IOREWRT
		lfn

This form of the WRITIN request transfers data from the working storage area to the buffer.

WRITIN lfn,/name/ (x is of the form /name/)

59	29	17 0
	RJ	IORRW
		lfn
name		

This form of the WRITIN request begins rewriting the /name/ record on the file named lfn, using the words in the working storage area as the first n words of the record. The named record must have been written previously, so that the file index has the name to which a storage address has been already assigned.

WRITIN lfn,m
(x is of the form m, which is logical record number)

59	29	17 00
	RJ	IORRW
		lfn
		m

This form of the WRITIN request begins rewriting logical record number m on the file named lfn using the words in the working storage area as the first n words of the record. The file must be indexed, either by name or by number, and the logical record referenced must have been written previously.

POSITION FUNCTIONS

The following position functions apply equally to files on magentic tape and mass storage, except where otherwise indicated.

SKIPF lfn,n,lv,recall

59	47	41		39	29	17	13 00
SA1	lfn				RJ	CPC	
000003		0	r	n		lv	000240

SKIPF causes one or more logical records to be bypassed in a forward direction. The request may be initiated at any point in a logical record. The number of logical records or record groups to be skipped is specified by the n parameter; the value 1 is assumed if n is absent. The maximum octal value of n is 777776(octal); if n = 777777(octal), the file is not positioned.

If the level parameter (lv) appears, logical records are skipped until an end-of-record with a level number greater than or equal to the requested level is reached; the file is positioned immediately following the end-of-record mark. For example, using the level number table in FILES (section 3), a SKIPF lfn, 2, 1 issued while positioned at page 6 would cause repositioning to the beginning of chapter 5 (Level Numbers, 1-3). If lv = 17(octal), skipping is performed until record level 17 or an end-of-file mark (tape mark) is encountered.

For S and L tapes, if lv ≠ 17, the level is assumed to be zero.

If the level parameter is absent, this field is set to zero and the file is positioned forward n logical records (or partial logical records if SKIPF is issued in the middle of a logical record).

If the end-of-information is encountered before an end-of-record with the specified level is found, the end-of-information status bit will be set. Parity errors encountered during a SKIPF operation are ignored.

On external tapes, level numbers are not appended to records. However, level numbers may be specified for SKIPF requests. If level number 17 is specified, a skip to end-of-file is performed. For other level numbers, one record is skipped.

SKIPF is considered to be complete if an end-of-reel condition occurs on a magnetic tape file for which the UP bit is set.

REVERSE FUNCTIONS

Reverse functions will not go beyond the beginning of the current reel of magnetic tape. If beginning of reel is encountered before the requested number of backspaces is reached, the beginning of information status bit is set.

Parity errors encountered during backspace operations are ignored.

If a magnetic tape file is positioned immediately after a newly written record when a reverse motion function is issued, trailer label procedures will be executed before the function is performed. Four tape marks will be written if a trailer label format is not defined.

BKSP lfn,recall

59	47	41	39	29	17 0
SA1	lfn			RJ	CPC
000003		0	r		000040

One logical record is bypassed in a reverse direction. This request may be issued at any point in a logical record. This function is a subset of SKIPB; it is included for compatibility with previous systems.

BKSPRU lfn,n,recall

59	47	41	39	35	29	17	0
SA1	lfn				RJ	CPC	
000003		0	r		n	000044	

One or more PRU's are bypassed in a reverse direction. The request may be issued at any point in a logical record. The number of PRU's to be bypassed is indicated by n: if n does not appear, one PRU is bypassed. Parity errors encountered during a BKSPRU are ignored.

SKIPB lfn,n,lv,recall

59	47	41	39	35	29	17	13	0
SA1	lfn				RJ	CPC		
000003		0	r		n	lv	000640	

One or more logical records are bypassed in a reverse direction. The request may be initiated at any point in a logical record. The number of logical records or logical record groups to be skipped is specified by the n parameter; the value 1 is assumed if n is absent. When n is the maximum value of 777777(octal) the file is rewound.

If the level parameter is used, logical records are read backwards until a short PRU of the specified level has been read. A forward read is issued, leaving the file positioned after this short PRU. If the file is positioned initially between logical records, the level number immediately preceding the current position is ignored in searching for a logical record of the specified level. This positioning process is performed n times.

Consecutive logical records within a file may be organized into a group by using level numbers. The file will be composed of one or more groups of logical records. This may be done by choosing a minimum level number lv $\neq 0$ and assigning a greater than or equal level number to the last logical record of each group, and a smaller level number to all other logical records.

Then SKIPB lfn,,lv will skip the file backward to the beginning of the logical record group which immediately follows a logical record of level lv. In the example of level numbers shown in section 3, the minimum level number was 1; a SKIPB lfn,2,1 issued while positioned at page 14 would cause repositioning to the beginning of chapter 4.

If the level paramter is absent, this field is set to zero and the file is positioned backward n logical records (or partial logical records if the SKIPB is issued in the middle of a logical record).

If the beginning-of-information is encountered before the requested level number is found, the beginning-of-information status bit is set. Parity errors encountered during a SKIPB operation are ignored.

For S and L tapes, only levels 0 and 17 are recognized. If lv $\neq$ 17, zero level is assumed.

REWIND lfn,recall

59	47	41	39	29	17	0
SA1	lfn			RJ	CPC	
000003		0	r		000050	

REWIND positions an allocatable device file at the beginning of information. A tape file is positioned at the beginning of the current reel; labels will be skipped when the next forward motion function is issued. A REWIND function on a file already rewound has no effect. A REWIND request for a device not capable of being repositioned causes a status 22 (illegal function) to be returned.

When REWIND function is issued on a member file of a multi-file set, the member file is repositioned just before the header label of the member file; or to the beginning of the current reel, if the member spans two or more reels and the second or subsequent reel is currently mounted. If the multifile set name is used instead of a logical file name, the job will be terminated.

UNLOAD lfn,recall

59	47	41		39	29	17	0
SA1	lfn				RJ	CPC	
000003		0	r			000060	

UNLOAD operates in a manner similar to REWIND. If the file resides on magnetic tape, the tape is rewound, and then unloaded. The unit is returned to the control point pool.

COMMON addr,recall

59	41		39	29	17	0
				RJ	CPC	
CTS	1	r			addr	

This function may be used to manipulate common files from a central processor program. A new common file is one created during the job and not yet detached from the control point. An old common file is one that was common at the beginning of job, but has since been released by a RELEASE card or the COMMON request (n = 2). An old common file is treated as a local file except for the RERUN feature. Addr is the address of a word in the following format:

59	17	0
lfn	n	

lfn is left justified with binary zero fill. n specifies both the function and the resultant status reply word.

Attach or create common file lfn: n = 0

n = 1	Request was completed normally; either a local file was changed to a new common file or an unassigned common file was attached to this control point. In the latter case the job will be marked as unable to be rerun.
n = 3	A common file named lfn is attached to another control point, but no local file named lfn is at the calling control point.
n = 5	No common file lfn is in the system at present, and no local file lfn is at the calling control point.
n = 7	Duplicate names; the calling program has a local file named lfn and a common file in the system is named lfn.

n = 11	A common file named lfn is already at this control point.
n = 13	The common file lfn resides on equipment which cannot be assigned to this control point.
n = 15	The file is a permanent file.

Release common file lfn: n = 2

n = 1	Request was completed normally; the common or the new common file lfn assigned to this control point will be changed to type local at job termination. If the file is an old common file, the job will be marked as unable to be rerun.
n = 3	Common file named lfn is not at this control point.

Detach common file lfn: n = 4

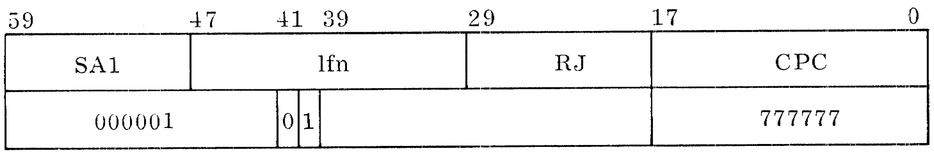
n = 1	Request was completed normally; the common file or the new common file lfn was detached from this control point; lfn is no longer available to this control point. The job will be marked as unable to be rerun. Operation is the same as a CLOSE, UNLOAD, except that no index is written.
n = 3	Common file named lfn is not at this control point.

SYSTEM ACTION REQUESTS

RECALL

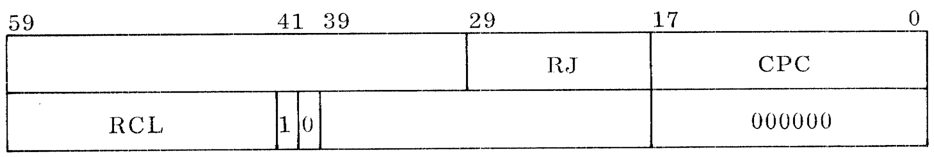
The RECALL request generates one or two calling sequences depending on the presence or absence of the lfn parameter. Execution of either request causes the job to relinquish the central processor.

RECALL lfn



lfn is the base address of a file environment table. Control returns to the program when bit 0 of the code and status field becomes a one, indicating completion of an input/output request for that file. Error checking is performed and an OWNCODE routine executed, if necessary, before control is returned. Since recall may be entered when the operation is initiated if the recall parameter is used, RECALL is needed only if some useful processing can be done between initiating and completing an input/output operation.

RECALL



When RECALL does not specify lfn, the central processor is relinquished only until the next time around the monitor loop. The user must determine whether the condition that required a recall is still present.

MEMORY type,status,recall

59	41	39	29	18	17 0
			RJ		CPC
MEM	1	r		t	status

The field length assigned to a job may be obtained or changed by the MEMORY request. Control will not be returned until the request is complete.

The type parameter should be CM or ECS indicating central memory or extended core storage field length referenced. In the macro expansion t is set to 0 for CM and 1 for ECS.

If the location addressed by status initially contains zero, no field length is altered; the current field length is returned in the upper half of the location, and bit 0 is set to one.

If the upper half of the location initially contains a number, the field length is altered to equal the value of the number and bit 0 is set to one.

Bits 0-29 of the location addressed by status should initially contain zero in either case.

MESSAGE addr,x,recall

59	41	39	35	29	23	17 0
				RJ		CPC
MSG	1	r		x		addx

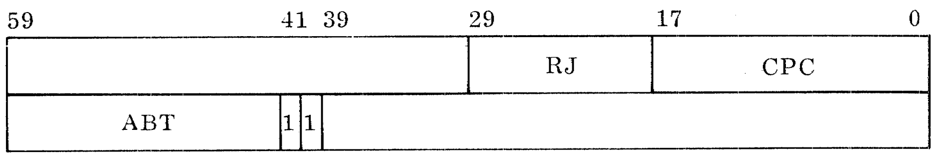
The MESSAGE request enters a message into the job dayfile. The message must be stored in display code and must not contain any characters with display codes greater than 57 octal (appendix A). The maximum message length is 80 characters because of truncation by the dayfile routine. SCOPE considers the message to end either at the first word with all zeros in the rightmost 12 bits or at the 80th character, whichever comes first. Any characters beyond the 40th appear on a second line. If the x parameter is non-blank, the message is displayed but not entered into the dayfile. If the recall parameter is omitted, addx is addr which is the address of the start of the message; if recall bit is set, addx is a pointer to addr. (See Miscellaneous Requests, section 4.)

ENDRUN

59	41	39	29	17 0
			RJ	CPC
END	1	1		

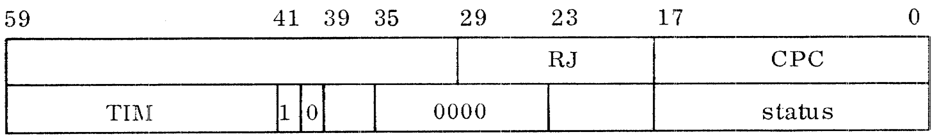
Normally a run ends with execution of the ENDRUN request. SCOPE examines the control card record of the job deck, and begins execution with the next unprocessed control card. If there are no more control cards or if the next card is an EXIT card, the job is terminated.

ABORT

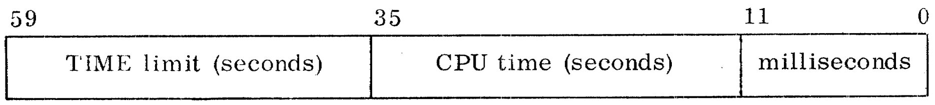


Execution of this request causes the monitor to terminate the job, just as if an error, such as out-of-bounds memory reference, had occurred. If the control card section of the job deck contains an EXIT card, the system continues processing the job with the next control card after the EXIT card.

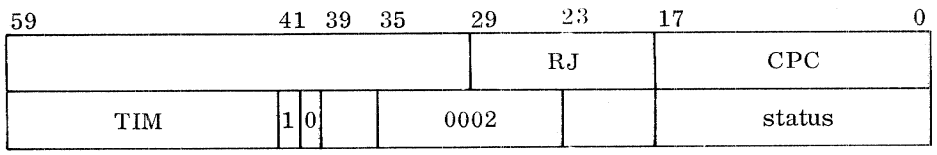
TIME status



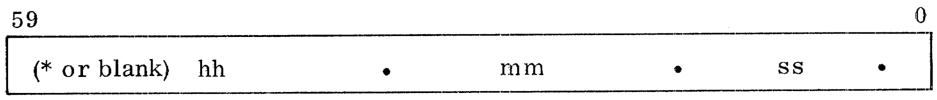
Before clearing RA + 1. monitor returns, in location status, the job time limit and the central processor time already used by the job in the following binary format:



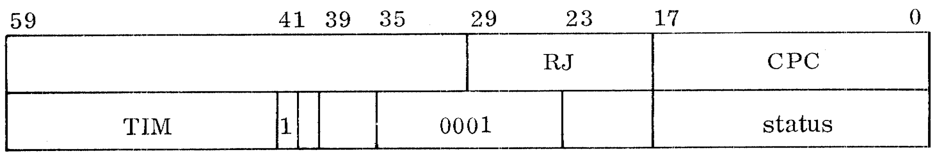
CLOCK status



Before clearing RA + 1, monitor returns the current reading of the system clock, in location status. An asterisk precedes the time if elapsed time since deadstart is returned.

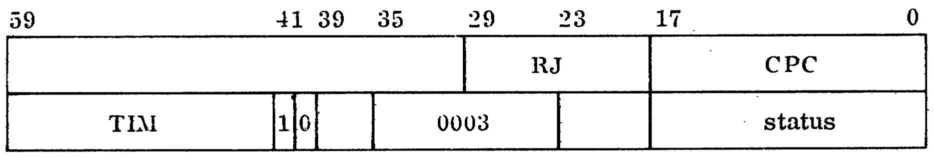


DATE status

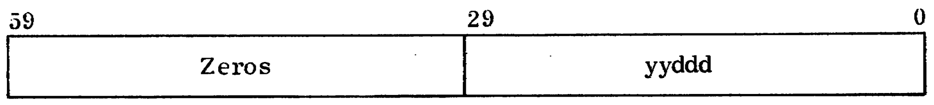


Before clearing RA + 1, monitor returns, in location status, the current date, as typed by the operator, with one leading blank and one trailing blank.

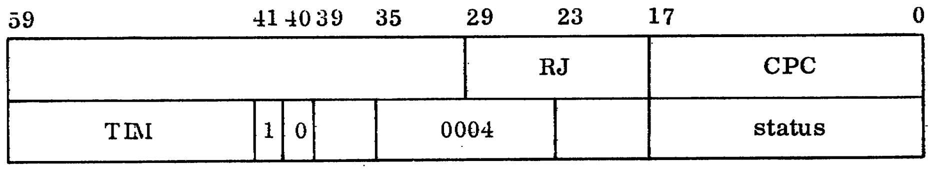
JDATE status



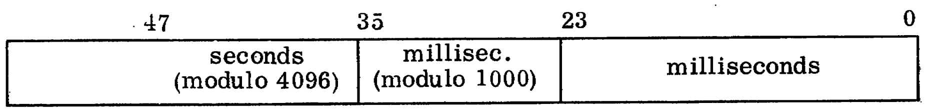
Before clearing RA + 1, the Julian date is returned, in status, in the following format: (yyddd is in display code)



RTIME status



Before clearing RA + 1, the real time clock maintained by monitor is returned in location status in the following format:



PERMANENT FILES 5

A permanent file is a mass storage file cataloged by the system, so that its location and identification are always known to the system. Frequently used programs, subprograms and data bases can be immediately available to requesting jobs. Permanent files cannot be destroyed accidentally during normal system operation, including deadstart; they are protected by the system from unauthorized access according to the privacy controls specified when they are created.

Any local file, regardless of mode or content, which is not already permanent or common can be made permanent on a valid rotating mass storage device specified by the installation. Unless the user explicitly requests the system to catalog a file, it will not be made permanent. Common files cannot be made permanent.

The following permanent file functions are available under SCOPE through control cards:

CATALOG	Make existing local mass storage file permanent.
ATTACH	Attach a previously cataloged file to a control point.
RENAME	Rename a file in the permanent file directory.
EXTEND	Make permanent an extension to a currently attached file.
PURGE	Remove a file from the permanent file directory.

The preceding five functions plus the following two functions are available for use as system macros.

PERM	Determine permissions granted to an attached file.
FDB	Generate a file description block required for interface with the permanent file system.

In addition, the SCOPE system macro

CLOSE lfn,UNLOAD

can be used on an attached permanent file to effect its logical detachment prior to the end of job. The following SCOPE control card may be used for permanent files:

RETURN	Logically detach an attached permanent file before termination of a job.

TERMS AND CONCEPTS

Terms and concepts used throughout this chapter are defined as follows:

ATTACH

This function causes a permanent file to be assigned to a job control point. No user may access a permanent file until his right to access the file is established and the file is attached to his job. Usually, a permanent file is detached from the job at job termination. Before job termination, a file may be detached through the RETURN request.

CATALOG

This function enters a file's location and other pertinent information in the permanent file directory maintained by the system, thereby making the file permanent.

CYCLE

Up to five separate files may be cataloged under one permanent file name. Each file is called a cycle, and each shares the same user ID and set of passwords. No restrictions are imposed on the content or size of any cycle, as each is a unique file. A cycle can exist in either of two states: complete or incomplete. A cycle is incomplete until the information in the corresponding permanent file tables is updated.

Each cycle is identified by the combination of permanent file name, cycle number and owner ID. Cycle numbers from 1 to 63(decimal) are assigned by the creator of the file; when cataloging is in process, the default value is 1. If a cycle number is not given in the ATTACH request, the largest complete cycle number cataloged is assumed (which is not necessarily the latest cycle cataloged).

EXTEND

This function makes permanent an appendage written to an existing permanent file. For local extensions to be cataloged as part of the permanent file, the EXTEND permission must be granted when the file is attached and the EXTEND function must be used.

ID AUTOMATIC MODE

When the permanent file system is working in automatic mode, a system program supplied by the installation causes an installation or job-associated' ID' name to be placed in the control point area assigned to the job. When a file is cataloged, the automatically supplied name is cataloged in the permanent file directory. The ID parameter may be needed on the CATALOG control card if a new cycle is being cataloged under an existing permanent file name and the ID in the control point area does not match the ID in the permanent file directory. When a file is attached or purged under automatic mode, the ID in the permanent file directory is checked against the automatically supplied name in the job control point area. If it matches, the password list on the control card will be processed. If it does not match, the ID parameter submitted by the user is checked against the ID in the PFD. If they match, or the ID parameter is not present, the password list is processed; otherwise, the job is terminated. In renaming a file, the name supplied on the control card is checked against the installation-supplied name in the control point area. If they match, the new ID will be recorded in the PFD in place of the old ID. If they do not match, the new ID will be ignored and the name will not be changed. An installation may inhibit this function of checking ID at attach time.

ID NORMAL MODE

The function of the ID parameter in normal mode is to supply a file owner/user name to the permanent file manager. When a file is cataloged in normal mode, the ID name specified is cataloged in the permanent file directory. When a file is attached or purged, the ID supplied must match the name cataloged in the permanent file directory; otherwise the requesting job will be terminated. When renaming a file in normal mode, the requestor must have successfully attached the file and gained all four permissions: READ, EXTEND, MODIFY and CONTROL. The ID name specified will be written in the directory to replace the previous owner ID used to attach the file. An installation may inhibit the ID checking feature.

MULTIPLE ACCESS

Any number of users simultaneously may attach a permanent file for read-only access. If a file is attached for write/rewrite purposes, multiple-read access is not available; however, multiple access for rewrite can be granted at the installation's discretion. Multiple ATTACH requests are handled in the order received.

PERMANENT FILE

A mass storage file of any standard mode, content and length, locatable by a system search of the permanent file directory (PFD). It is protected from unauthorized access and accidental destruction in any normal running environment by any version of SCOPE that supports permanent files.

PERMANENT FILE PRIVACY, SECURITY AND PROTECTION

Privacy in permanent files is designed to minimize software interference (threats made to the permanent files by non-authorized central processor programs). In general, no attempt is made to prevent possible threats by operational personnel or by hardware interference, as these depend on the environment. The Installation is responsible for ensuring the integrity of the operating system at deadstart time and preventing the dynamic changing of the operating system (via EDITLIB).

The permanent file system offers a standard set of privacy controls. An installation may replace the standard privacy procedure if it requires a different kind of protection.

SCOPE automatically ensures that no normal operation will cause permanent mass storage files to be overwritten or otherwise destroyed, and that the permanent disk tables will not be destroyed.

In addition to normal system protection of permanent files, the individual file owner can prevent unauthorized access to his permanent file. He can stipulate, when he catalogs a file, the degree to which the file is to be protected from read, write, rewrite access. Once a file is cataloged, it cannot be used by any job unless the necessary passwords are given when a request is made to attach the file.

PUBLIC FILES

An installation may define combinations of one or more permissions that may be granted automatically by activating the Universal Permission option (IP.UP). A nine-character password is defined by the installation for this permission combination. When the password is given on an ATTACH request, the associated permissions are granted.

A file may be cataloged under the ID of PUBLIC if the universal permission password is correctly specified. Attaching a PUBLIC file does not preclude the necessity of gaining the correct permission code. As all cycles of a file share the same ID, if the first cycle is cataloged as PUBLIC, all subsequent cycles will become PUBLIC.

With the RENAME function, PUBLIC files can be given a new owner ID, thus making them non-PUBLIC (or private).

PURGING/RETENTION

Purging causes a permanent file's control information to be removed from the directory and its mass storage space to be released. Anyone who knows the control password and user ID may purge a file; only one cycle of a file may be purged at a time. Purging is done with the PURGE function. When a permanent file is cataloged, a retention period may be specified; but the file will not be purged automatically when its expiration date is reached. An installation can obtain a listing of all expired files not purged from the system.

RENAME

Rename allows certain information cataloged for a permanent file to be changed, such as file name, cycle number, owner's ID, passwords and retention period.

REWRITE OR MODIFY

Any action that modifies the content of an existing permanent file, including the data function macros REWRITE, REWRITER, REWRITEF, or WRITIN.

SUB-DIRECTORY

The permanent file directory is divided into sub-directories which are identified numerically. Each permanent file is associated with the number of the sub-directory in which it has been cataloged. The sub-directory reference may be used to expedite a system action requested for that file.

WRITE

When a permanent file is attached to a job and positioned at end-of-information, any write function, except a rewrite, will add data to the file and extend the end-of-information point. Such an extension is considered to be temporary; unless the EXTEND function is performed successfully on the file, the extension is released when the job is terminated.

ACCESS PERMISSIONS AND PASSWORDS

The individual file owner can prevent unauthorized access to his permanent file. When he catalogs a file, he can stipulate the degree to which the file is to be protected from read, write, rewrite or purging. Once a file is cataloged, it can be accessed only through the necessary passwords when a request is made to attach the file.

Four basic access permissions are associated with a permanent file:

READ	To permit the read access of the file
MODIFY	To allow the user to change the content of an existing permanent file
EXTEND	To make permanent an extension written beyond the end-of-information mark
CONTROL	To catalog an additional cycle or to purge an existing cycle of a file

Depending upon the degree of privacy required, the file owner may specify up to five passwords to be associated with a file when it is cataloged in the permanent file directory. Subsequent cycles of the file assume the same passwords as the first cycle cataloged. Passwords can be any string of 1 to 9 alphanumeric characters. Each password implies one type of access permission for subsequent ATTACH or new cycle CATALOG attempts. TURNKEY, the fifth permission/password, provides an extra measure of control over file access. When the turnkey password is cataloged for a file, no permission is granted requestors unless the turnkey password is included on the file ATTACH card.

If a password is not cataloged for READ, MODIFY, EXTEND or CONTROL, access for that permission is given automatically to requestors. Multiple-read access is granted when either the user, after permission checking, is left with only read permission or the MR=1 parameter is specified. Multiple-read access permission is specified by the MR parameter, as described under the table of parameters in this section.

An installation-optional feature, Universal Permission, may be activated whereby an installation can define combinations of one or more permissions that may be granted. A nine-character password will have been defined as the Universal Permission password by an installation. When given on an ATTACH request, the associated permissions will be granted.

FUNCTION MACROS

The permanent file functions CATALOG, ATTACH, RENAME, EXTEND and PURGE are available for use as either control cards or running program macro calls. The same parameters are used for both; the difference is in the format of the call, and in the capability to test status in the running program.

MACRO REQUESTS

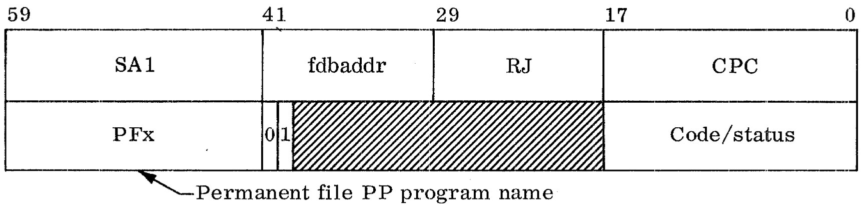
All permanent file macro requests share a common format:

name fdbaddr,RC,RT

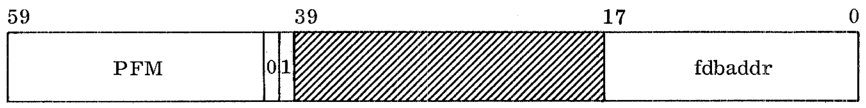
name	Macro name written in the mnemonic field of a COMPASS instruction.
fdbaddr	Address of the fifth word in a file definition block (FDB)
RC and RT	Optional parameter. RC will return control to the requestor on non-fatal errors. RT permits the PF manager to place a return code in the FDB and return control to the requestor on an unsatisfied real time call. At the present, no real time response will be received from CATALOG, RENAME and EXTEND macro requests; and it is not required in the PERM macro request. Use of RT implies that RC is also set.

MACRO REQUEST CALLS

Each permanent file request macro expansion (except FDB) generates the following two-word call to CPC:



An RA + 1 call to PFM has the following format:



PERMANENT FILE CONTROL CARDS

All permanent file control cards are written in the form:

function name (parameter list)

The five function names are: CATALOG, ATTACH, EXTEND, RENAME and PURGE.

All parameters in the list are written in the forms:

file name	(logical or permanent file name)
code=	(two-letter parameter code)
code=value	(decimal value for parameter)
code=password	(password for parameter)

Permanent file control statements may be continued on as many cards as needed to contain the function parameters. If a card has no parameter list terminator (right parenthesis), column 1 of the next card is considered as the immediate continuation of column 80.

The colon (:) should not be used on permanent file control cards or in permanent file macros.

PARAMETERS

With the exception of the lfn and pfn file names, the parameters described below may be written in any order on control cards. File name parameters are position dependent, and omitted file name parameters must be indicated by commas.

lfn	Logical file name, 1 to 7 characters, the first of which must be alphabetic. After a file is attached to a requesting job, it is referenced by this name.
pfn	Permanent file name under which a file is cataloged; 1 to 40 alphanumeric characters assigned by the file creator.
ID=name	Name, 1 to 9 alphanumeric characters, which identifies the file creator (or owner). This parameter is used in either of either of two modes, depending upon an installation's accounting procedures. In the normal mode, which is standard in SCOPE, the requestor is required to identify himself as the file owner. In the automatic mode, the installation stores the user ID with each job in the SCOPE control point area. (ID modes are described under Terms and Concepts at the beginning of this section.)
CY=n	The cycle number, 1 to 63, assigned by the file creator. The default value on an initial CATALOG is 1; on ATTACH, it is the highest complete cycle number cataloged.
RP=n	Retention period in number of days, 0 to 999, as specified by the file creator; indefinite retention is indicated by 999. An installation can define a default value.
PP=name	The 1 to 9 characters specified for name represent information to be passed to an installation-defined privacy procedure.

TK=pw	Turnkey password	
RD=pw	Read password	
EX=pw	Extend password	1 to 9 alphanumeric characters
MD=pw	Modify password	
CN=pw	Control password	

PW=list A maximum of five passwords on an ATTACH control card which establish user's access permission for a file.

PW=pw1,pw2,...,pw5

PW is required also on the CATALOG card when a new cycle is added, and on the PURGE card under the permanent file name mode. PW is used to submit passwords; TK, RD, EX, MD and CN are used to define passwords.

SD=n Subdirectory number, 1 to 999.

RN=n If n is zero, unique permanent file name generation is inhibited; if n is one, a name will be generated automatically. An installation default value may be defined.

MR=n If n is one, the file can be attached for multi-read access only, regardless of other access permissions given by default. Any other value for n inhibits multi-read access.

PERMANENT FILE FUNCTIONS

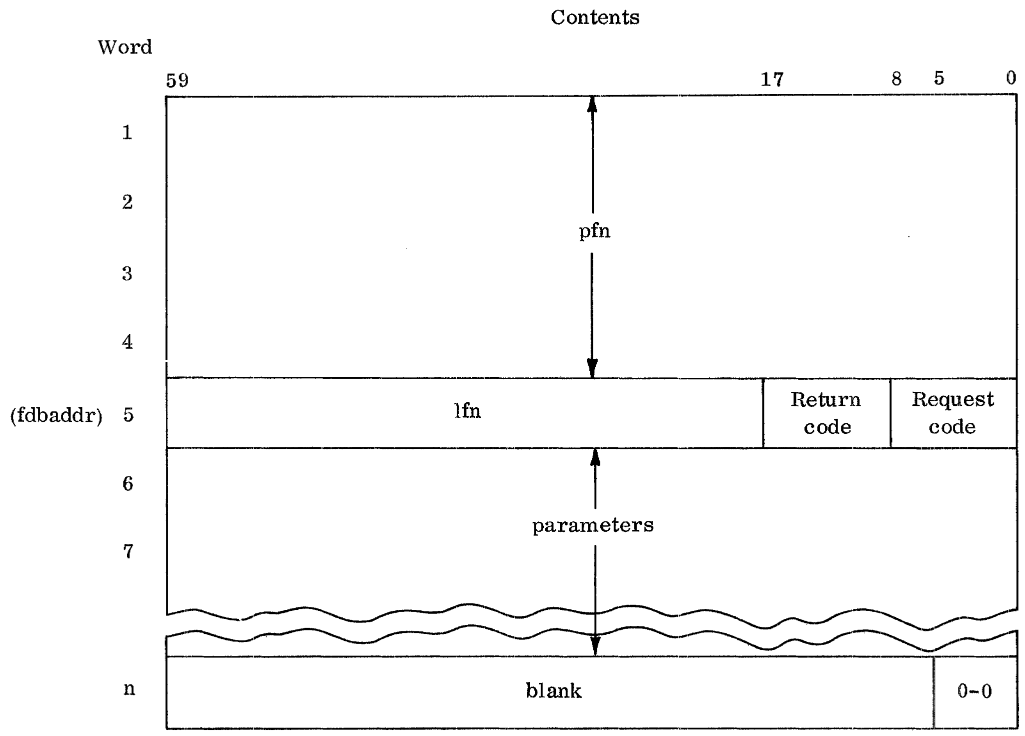
The permanent file functions described in this section are written as SCOPE control cards. They are also available for use as COMPASS program macro calls. All parameters may be given on control cards; however, only those shown in the text will be recognized as meaningful and others not pertinent to the function will be ignored. The functions and their effective parameters are summarized below:

	lfn	pfn	CY	ID	TK	RD	EX	MD	CN	PW	RN	SD	RP	PP	MR
CATALOG	+	+	*	+	*	*	*	*	*	*	*	*	*	*	
ATTACH	+	+	*	*						*		*		*	*
EXTEND	+														
RENAME	+	*	*	*	*	*	*	*	*		#		*		
PURGE	#	#	*	*						*		*		*	

+	Parameter required	#	Special case, see text
*	Parameter optional	blank	Meaningless, ignored if present

FILE DEFINITION BLOCK

Parameters necessary for the execution of a permanent file function are contained in a central memory table called the file definition block (FDB). Error codes are returned to the user via the FDB. The FDB is generated automatically for control card calls, but it must be generated by an FDB macro for use by other permanent file macro calls. The format of the FDB follows:



pfn	Up to 40 display code characters, left justified with zero fill.
lfn	Logical file name of 1 to 7 alphanumeric characters, left justified with zero fill.

The macro for generating an FDB has the following format:

fdbaddr FDB lfn,pfn,parameter list

fdbaddr Symbol to be associated with word 5 of FDB; it must be present in the location field. The FDB will be generated in-line during assembly whenever the macro is called.

Parameters are separated by commas, and the list is terminated by a blank. Parameters may include any of those indicated by the 2-letter codes given above. Parameters will be entered into the FDB as they are encountered in the list.

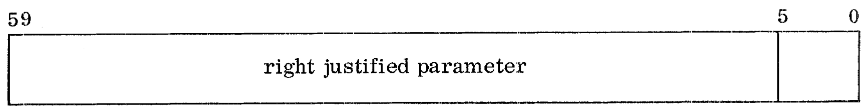
If the RC or RT parameter is specified in a function call referencing an FDB, a return code will be available to the user in fdbaddr (bits 17-9).

Completion of request is indicated by setting bit zero to one.

The PFM request codes are detailed below:

010	ATTACH	Return control to user on non-fatal error (for macro, call with RC)
020	CATALOG	
030	EXTEND	
040	PURGE	
050	RENAME	
060	PERM	
110	ATTACH	Abort on any error (all other calls)
120	CATALOG	
130	EXTEND	
140	PURGE	
150	RENAME	
160	PERM	
210	ATTACH	Return control to user on non-fatal error or real time request unsatisfied (for macro, call with RT)
220	CATALOG	
230	EXTEND	
240	PURGE	
250	RENAME	
260	PERM	

The parameter words of the FDB have the following format:



Parameters are stored, one per word in any order. FDB values are shown below:

Value (Octal)		
00	End of FDB list	
01	PP	Privacy procedure parameter (display code)
02	RP	Retention period in days (binary)
03	CY	Cycle number (binary)
04	TK	Turnkey password (display code)
05	CN	Control password (display code)
06	MD	Modify password (display code)
07	EX	Extend password (display code)
10	RD	Read password (display code)
11	MR	Multi-read access only (binary)
12	SD	Sub-directory (display code)
13	null	
14	ID	User identification
15	RN	Automatic rename (binary)
20-24	PW	Submitted passwords

All other values (except 50 and beyond) are reserved for future system use.

Word 5, Bits 9-17: The 9-bit return code can take the following values.

User Return Codes (Octal)	
0	Function successful
1	Identification error
2	Logical file name already assigned (ATTACH)
3	Logical file name not found (CATALOG, EXTEND, PURGE, RENAME)
4	Blank permanent file name (CATALOG, ATTACH)
5	Directory full (CATALOG) (PFD)
6	Catalog full (CATALOG, EXTEND) (RBTC)
7	Permanent file device unavailable
10	Latest index not written for random file (CATALOG, EXTEND)
11	Illegal device for file residence (CATALOG)
12	ATTACH request for unknown file
13	Cycle referenced does not exist (ATTACH, PURGE, EXTEND)
14	Invalid cycle number
15	Duplicate name/cycle or no slot (CATALOG, RENAME)

User Return Codes (Octal)	
16	Attempt to recatalog existing permanent file
17	Attempt to CATALOG non-local file
20	Function attempted on non-permanent file
21	Function attempted on purged file
22	CATALOG attempt; no word pairs in RBT chain for file
23	Cycle incomplete on an ATTACH
24	Duplicate ATTACH request
25	Real Time request unsatisfied

On control card requests all errors are fatal; on macro requests, unless the RC or RT parameter is specified, the job is terminated.

On any breach of privacy, the job is terminated.

CATALOG FUNCTION

A local file attached to a job is made permanent with the CATALOG function. Cataloging consists of locking out the recording area on which the file is written to prevent its reassignment and entering the file's name, location, and control information into the permanent file directory.

The two modes of cataloging are: Initial in which a new permanent file is created with a unique name, and Newcycle in which a new cycle is created for an existing permanent file. The mode of operation is implicit in the CATALOG call. Newcycle mode is entered when a CATALOG request is received for an existing permanent file in which a cycle number is specified by the CY parameter.

The control card is written:

CATALOG(lfn,pfn,parameters)

The macro is written:

CATALOG fdbaddr,RC,RT

If RC is specified, the user is notified of a non-fatal error condition by an error return code at fdbaddr in the FDB.

RT specifies a real-time call. Currently, no real-time response will be received from Catalog.

Specifying the RT option forces the RC option, if it is not selected.

For both RC and RT options, informative and diagnostic messages to the dayfile are suppressed.

The lfn, pfn, and ID parameters are required in both the control card and macro; the other parameters described below may be included:

ID=name	File creator/owner's identification must be given for both Initial and Newcycle cataloging. The names PUBLIC and SYSTEM are reserved.

PW = list	Password list has meaning only for Newcycle cataloging.
RP = n	Retention period in number of sequential days; if not given, zero is assumed.
CY = n	Cycle number; required in Newcycle mode. In Initial mode, CY may be given; if it is not, cycle number 1 is assumed. The presence of the CY parameter establishes Newcycle mode; its omission implies Initial Mode.
RN = n	Automatic rename option. When n is 1 and a duplicate name has been cataloged, the name specified by pfn will be made unique with a numeric prefix. The new unique name will be output in a message to the dayfile. If n is 0, unique name generation is inhibited.
TK = pw	Turnkey password; required only if definition of a turnkey password is requested during an initial catalog.
RD = pw	Read password; required only if definition of a read password is requested during an initial catalog.
EX = pw	Extend password; required only if definition of a modify password is requested during an initial catalog. Permission allows cataloging of local extensions.
MD = pw	Modify password; required only if definition of a modify password is requested during an initial catalog. Permission allows content of the file to be changed.
CN = pw	Control password; required only if definition of a control password is requested during an initial catalog. Permission allows the cataloging of additional cycles and the purging of permanent files.
SD = n	Subdirectory number, which has meaning only in Newcycle mode. If omitted, the subdirectories are searched until the pfn entry is located.
PP = name	This parameter may be used for any purpose defined by the installation.

The cataloging process always searches the directory for a duplicate name. If one is found, the action taken depends upon the rename (RN) parameter or the system default action. A duplicate name may cause conflicts under the following conditions:

1. When no CY parameter is given, Initial mode is assumed. If the rename option is RN = 1, a new permanent file name is created and cycle 1 is cataloged. If RN = 0 or is not permitted by installation option, the job is terminated.

2. When the cycle number is a duplicate of an existing file cycle, the rename option is consulted. If renaming is permitted, a new file name is created and the cycle is cataloged as an initial cycle. If renaming is not permitted, the job is terminated.
3. If five cycles of a file (maximum capacity of a directory entry) have been cataloged, the rename option is consulted. If renaming is not permitted, the job is terminated; otherwise, a new file name is generated and the cycle number is cataloged as an initial cycle.

In each instance where automatic renaming takes place, the new name is returned to the user in a dayfile message.

Before requesting the CATALOG function for SCOPE random files, the user must ensure that the index is written out as the last logical record of the file after the last data. When SCOPE Indexed Sequential (SIS) files are cataloged, both the data file and its index are cataloged as one file; they cannot be treated as separate files.

Once a file is cataloged, it remains attached to the job as a local file as if it were an attached permanent file with all access permissions granted to the creator.

CATALOG EXAMPLES

The following examples illustrate establishing file privacy and providing automatic access permission:

```
    CATALOG(STAT1,PAYSTAT,ID=BJC,EX=PS1,MD=PS2,CN=PS3)
```

Initial mode always is assumed until a permanent file name is matched in the directory. At that time, the cataloging mode becomes Newcycle. Since the CY parameter does not appear in the example, the permanent file directory is searched to determine that the name PAYSTAT is unique. If it were not, the job would be terminated since the rename option (RN) is not specified. Otherwise, the local file called STAT1 is cataloged for owner BJC. The three passwords, PS1, PS2 and PS3 provide access permissions for EXTEND, MODIFY and CONTROL, respectively. When another job requests that the permanent file PAYSTAT be attached, the read access permission is given automatically as no password is cataloged for that type of access. Permission to extend, modify or add/delete a cycle can be granted only when the correct password is submitted using the PW parameter.

```
    CATALOG(STAT2,PAYSTAT,ID=MCY,CY=2,PW=PS3)
```

In this example, Newcycle mode is established when the file name PAYSTAT is matched in the permanent file directory. A control password was cataloged for permanent file PAYSTAT, so the password list (PW) is checked for password PS3 which permits a new cycle to be added. The correct password is required for control access; read access is given automatically as no password was required. The local file STAT2 is then cataloged as cycle number 2. The name MCY given with the ID parameter identifies the user who accessed the PAYSTAT catalog entry and created the new cycle of the file.

```
    CATALOG(ALPHA,MYFILE,RN=1,ID=X7830,TK=SESAME,RD=AR,MD=AM,EX=AX,RP=60)
```

In this example, Initial mode is assumed and the directory is searched to determine that the name MYFILE is unique. If not, the rename option (RN = 1) is used; the new name created is returned in the dayfile for the user's information. A turnkey password is cataloged along with the passwords for read, modify and extend permission. A retention period of 60 days is established and the logical file ALPHA is cataloged as a permanent file. On subsequent ATTACH requests, the turnkey password must be given as well as the passwords for read/modify/extend access. The turnkey password must also be given to obtain the no-password permission, control. For example:

Password List	**Permission Granted**
PW = SESAME,AR	Control and Read
PW = SESAME,AR,AX	Control, Read, Extend
PW = SESAME	Control only
PW = AR,AM	None (Turnkey password missing)
PW = AR,AM,SESAME	Control, Read and Modify

ATTACH FUNCTION

The ATTACH function is necessary to request attachment of a permanent file to a job and to establish the requestor's legal access to the file. Evaluation of the cataloged passwords and permissions, as well as the password list submitted with the request, establishes the type of access granted to the user. Once access legality has been ascertained and approved, the permanent file is attached to the job as a local file and may be used only as specified by the ATTACH function. For example, if the file was attached with only read permission, the user may not modify or extend the file. At job termination, the file is detached automatically; the file may be detached before job termination with the RETURN request.

The ATTACH control card is written:

ATTACH(lfn,pfn,parameter list)

The macro is written:

ATTACH fdbaddr,RC,RT

The RC parameter is the same as for CATALOG. If RT is specified, and octal code 25 will be returned in the FBD if the file is currently in use.

The logical file name, lfn, and the permanent file name, pfn, must be specified; pertinent optional parameters are:

CY=n	Cycle number of file to be attached. If this parameter is omitted or is 0, the cycle with the highest number is implied, provided it is a complete cycle. If this cycle is purged, the cycle with the next highest cycle number is implied, provided it is a complete cycle.
ID=name	Name identifies the file owner or user. In the automatic mode, the parameter may be omitted; it need be specified only with attempts to attach a file not cataloged under the user's ID.
SD=n	Subdirectory number, 1 to 999, where file control information is cataloged. If omitted or if the file is not found in the specified subdirectory, the entire directory will be searched; the correct subdirectory number will be noted in a dayfile message for the user.
PW=list	One to five passwords which establish the requestor's access permissions.
PP=name	Parameter to be passed to an installation defined privacy procedure.
MR=n	Multi-read access specification. MR=1 should be specified to obtain multi-read access. This specification allows READ permission to be generated by the user, and prevents other permissions from being granted by default. Multi-rewrite permission is possible when only READ and MODIFY permissions have been established at attach time, and then only if it is allowed by an installation. If another job already has the file attached and has inhibited multi-read access, the permanent file manager will wait until multiple access to the file is permitted before allowing the file to be attached.

RENAME FUNCTION

With this function, changes can be made to the name, cycle number, passwords and ID cataloged for a permanent file. The RENAME function cannot be performed unless the file has been attached and all four permissions (READ, EXTEND, MODIFY and CONTROL) have been granted.

The control card is written:

RENAME(lfn,pfn,parameter list)

The macro is written:

RENAME fdbaddr,RC,RT

RC and RT are the same as for CATALOG.

Only the logical file name (lfn) parameter is required. Remaining parameters are optional; if none are specified, the RENAME function has no effect and the control card is processed as a no-operation.

The optional parameters are grouped into two classes: those which could cause replacement of cataloged information for the file, and those which do not. Parameters which cause replacement if new values differ from the cataloged information for the file are listed below:

pfn	Permanent file name to replace the pfn used to attach the file.	
ID=name	New file owner identification. Under ID automatic mode, a user can change a file ID only to the ID supplied by his own installation. If the name specified in the RENAME function does not match the name supplied in the control point area, the new ID will be ignored; no change will be made.	
TK=pw	New TURNKEY password	1 to 9 characters
RD=pw	New READ password	
EX=pw	New EXTEND password	
MD=pw	New MODIFY password	
CN=pw	New CONTROL password	
CY=n	New cycle number for attached cycle When ID permanent file name, or passwords are changed for any cycle, the changes apply to all cycles cataloged for the file.	
RP=n	New retention period for the cataloged file cycle.	

To remove a cataloged password without substituting a replacement, the parameter should be the permission code and an equal sign followed by a delimiter such as comma or right parenthesis. For example, to remove the password for read permission, the parameter would be:

```
RD=,
```

The following parameter does not result in replacement:

RN=n	When n is 1, automatic generation of a unique new permanent file name is permitted using the lfn parameter on the RENAME card as a basis for the name. When n is 0, unique name generation is inhibited. When a new name is generated, a dayfile message informs the user of the new name.

Example:

Assume that file PFILE was cataloged by owner ABC with X as the read password, Y as the extend password, and Z as the modify password. Control is given automatically.

```
.
.
.
ATTACH(LFILE,PFILE,ID=ABC,PW=Y,Z,X)
RENAME(LFILE,PFILE2,RN=1,RD=,CN=W)
.
.
.
```

The permanent file name PFILE will be replaced by PFILE2, which will be made unique, if necessary, by the appearance of the rename option, RN=1. The read password will be removed (succeeding users will be given read permission automatically) and a password for control permission will be cataloged. The existing passwords for extend and modify will remain unchanged. Since the changes involve the permanent file name and passwords, the changes apply to all cataloged cycles of the file.

EXTEND FUNCTION

To catalog extensions to a permanent file, the file must be attached to a job which has been granted extend permission. Also, a file cataloged by a given job may be extended by that job. Information written at the end- of-information of the file is made permanent by the EXTEND function.

The control card is written:

EXTEND(lfn)

The macro is written:

EXTEND fdbaddr,RC,RT

The lfn parameter is required; it is the logical name of the attached file to be extended. Since the newly written section is an extension of the permanent file, it will acquire the privacy controls of the permanent file. If lfn refers to an indexed file, the current index will be rewritten at the end of the file, invalidating any prior index. When an EXTEND function is requested for a SCOPE random file, nothing must have been written on the file since the index was last written.

PURGE FUNCTION

With the PURGE function, a cycle of a file can be removed from the catalog of permanent files. Control permission must be established before a request for a purge function is granted.

The control card is written:

PURGE(lfn,parameter list)

The macro is written:

PURGE fdbaddr,RC,RT

The RC and RT parameters are the same as for CATALOG.

The logical file name is required on the control card and in the FDB macro; optional parameters are:

pfn	The permanent file name is given when the file cycle to be purged is not attached. When a purge by permanent file name is performed, lfn is required but serves no functional purpose. When the purge is by logical file name, pfn is not required. It may be a null parameter indicated by writing only the comma that would separate it from the parameter list (lfn,,parameters).
CY=n	The cycle of the file to be purged is designated by n; if omitted, the complete cycle with the highest number is purged.
SD=n	Number of the subdirectory containing the file. If not known, the SD parameter can be omitted, in which case the entire directory will be searched.
PW=list	Passwords that establish the requestor's access permissions. Control permission must be obtained, either by password or automatically.
ID=name	Name of the owner or user.
PP=name	Parameter to be passed to an installation privacy procedure, if implemented in the system.

PURGE can be complete or partial.

When a purge is performed, the permanent file is removed from the permanent file directory. A purge will be performed only on a permanent file attached with at least control permission or when the passwords submitted via the PW parameter on the PURGE card grant at least control permission.

Upon completion of the purge, the file reverts to the status of a local file and retains those permissions granted at ATTACH or PURGE time. A purged file can be recataloged only if it has all permission as a local file. Since the file reverts to local, its disk space is released as for any other local file.

A purge can be attempted in two modes: purge by local file name and purge by permanent file name. Purge by local file name implies the permanent file is already attached under the local file name specified on the PURGE card. Purge by permanent file name implies the permanent file is not already attached, and the local file name given on the PURGE card has not been used.

The partial purge is also useful in preserving file privacy when the universal permission password option is elected by an installation. The option can be set to grant control permission only, which allows the file to be purged. Since no other permission would be granted, no other access, such as read, extend or modify, is possible. To some installations, this feature may not seem important, however activation of the Universal Permission for control only is a threat only to file permanency and not to file privacy.

PERM FUNCTION

This function is available only as a COMPASS macro. PERM allows a running program to determine if a file is a non-permanent local file or which permissions have been granted to a currently attached permanent file. The macro request has the format:

PERM fdbaddr,RC,RT

The macro causes a 5-bit code to be returned in the fdbaddr return code field, as follows:

```
              y yyyy
             /   |
            /    |
           /      \
          v        \         Permissions granted:
0 Non-permanent file  -----> 1000  CONTROL permission
1 Permanent file             0100  MODIFY permission
                             0010  EXTEND permission
                             0001  READ permission
```

A return code of zero signifies that the lfn was not in the FNT for the control point (a non-existent file) or some other error was detected.

Example:

```
                .
                .
                .
FDBA            FDB DFLN,PFILE,CY=1,PW=XXX,ID=ABCD
                .
                .
                .
                ATTACH FDBA,RC
                .
                .
                .
                PERM FDBA
                .
                .
                .
```

Assuming the file had been cataloged with passwords required for both control and modify permissions, the ATTACH request would have generated control permission by the password XXX and read and extend permissions by default. Subsequently, the PERM request would cause an octal value 13 to be returned to the FDB. This code indicates a permanent file is attached with the permissions read, extend, and control.

EXAMPLES OF PERMANENT FILE CONTROL CARD USAGE

INITIAL CATALOGING

Example 1:

```
jobcard
REQUEST,FILE1,*PF.
COMPASS.
LGO.
CATALOG(FILE1,PFILE,ID=ABC,RN=1,CN=XXX,MD=YYYY)
7/8/9
(Program that creates logical file FILE1)
6/7/8/9
```

In the above job, a COMPASS program is assembled and executed. The program writes logical file FILE1, which is assigned to a permanent file device by the *PF parameter on the REQUEST card.

After the program terminates, FILE1 is made permanent by the CATALOG control card which specifies the name PFILE. Since the CY parameter is not specified and the pfn is found to be unique in the directory, cycle number 1 is assigned and an initial catalog attempt is made. If file name PFILE is found in the permanent file directory, the RN = 1 parameter will permit the system to generate a new name for the file so it can be cataloged.

The file will be assigned a control password of XXX and a modify password of YYYY; no protection is given against read or extend access.

Example 2:

```
jobcard
REQUEST,TAPE1.
REQUEST,OLDPL,*PF.
COPYBF(TAPE1,OLDPL)
CATALOG(OLDPL,PROGLIB1,CY=50,ID=ABC,RN=0,CN=XX)
REWIND(TAPE1)
6/7/8/9
```

This job copies a binary library file named TAPE1 from magnetic tape to a mass storage file, called OLDPL, on a permanent file device. The CATALOG request causes file OLDPL to be made permanent under the name PROGLIB1 as cycle number 50. Since the rename option is inhibited by RN = 0, no new name will be generated if a duplicate name already exists; in which case the job will be terminated. If the initial catalog attempt is successful, the file will be cataloged with a control password of XX.

NEWCYCLE CATALOGING

Example 3:

```
jobcard
REQUEST,LOC,*PF.
REQUEST,TAPEX.
COPYBR(TAPEX,LOC)
CATALOG(LOC,PROGLIB1,SD=6,CY=21,PW=XX,ID=ABC)
EXIT.
CATALOG(LOC,SCRATCH,TK=ABC,RD=DEF,RN=1,ID=ABC)
6/7/8/9
```

The above job copies file TAPEX from a tape to file LOC on a permanent file device. Assuming the job in example 2 was successful, this CATALOG request will be a newcycle attempt. Subdirectory 6 will be scanned first for the PROGLIB1 file entry. Control permission will be granted by the password XX. If no problems arise, the file will be cataloged as cycle 21.

If any errors occur in the catalog attempt, the control card after EXIT. will be executed; an attempt will be made to catalog the file under the name SCRATCH. This will be an Initial catalog attempt, and the file will be given turnkey and read protection. If necessary, the permanent file name SCRATCH will be made unique by the rename option RN = 1.

ATTACH EXAMPLES

Example 4:

```
jobcard
ATTACH(FILE,PROGLIB1,PW=XX,ZZZ,ID=ABC)
(remainder of job)
6/7/8/9
```

The permanent file PROGLIB1, created by examples 2 and 3, is to be attached to the requesting job and given the logical file name FILE by which it will be subsequently referenced. As the subdirectory parameter SD was not specified, all subdirectories will be searched for the file. Only a control password was cataloged; therefore extend, modify and read permissions are implied. In addition, control permission will be granted because the correct password, XX, was stated in the password list. The extra password ZZZ will be ignored and will not cause an error.

Omission of the CY parameter implies the cycle with the largest number; therefore cycle 50 will be attached. If example 2 had been unsuccessful but example 3 had been successful, cycle 21 would have been attached.

Example 5:

```
jobcard
ATTACH(OLDPL,PROGLIB1,CY=50,MR=1,ID=ABC)
UPDATE(Q)
RETURN(OLDPL)
COMPASS.
7/8/9
*IDENT ABC
(Update data)
*COMPILE DEF
6/7/8/9
```

This example specifies cycle 50 of permanent file PROGLIB1, created in example 2. Only a control password was cataloged, therefore the requestor automatically is granted modify, extend and read access permission. The multi-read option is inhibited in this example by the MR=1 parameter, and access permissions will be limited to read only. An UPDATE could be done as illustrated. The RETURN request logically detaches the file before the job is terminated. In this case, the file would be then available to other jobs for non-read-only attaching.

RENAME EXAMPLES

Example 6:

```
jobcard
.
.
.
ATTACH(LFILE,PFILE,PW=XXX,YYYY,ID=ABC)
RENAME(LFILE,PFILE2,RN=0)
.
.
6/7/8/9
```

PFILE is the permanent file cataloged in example 1, presuming that unique name generation did not occur at that time. Renaming of the file, in this example, will be permitted as the file user was granted all four access permissions: control and modify by password, read and extend automatically. The old file name is to be replaced with PFILE2 and no other, as the unique name generation feature is inhibited by the RN=0 parameter. If another file is already cataloged with the name PFILE2, this job will be terminated without changing the current name, PFILE. If renaming is successful, the file will retain the same passwords and cycle number.

Example 7:

```
jobcard
.
.
.
ATTACH(LFNAME,PFILE2,ID=ABC,PW=XXX,YYYY)
RENAME(LFNAME,,CY=10,CN=CCCC)
.
.
6/7/8/9
```

Here, the file created in example 1 and renamed in example 6 is attached with all four access permissions specified. In the RENAME request, the cycle number is to be changed to 10 and the current control password is to be changed to CCCC. In this example, the cycle number change applies only to the attached file cycle; yet the password change applies to all cycles that may be cataloged for file PFILE2. The permanent file name parameter may be omitted; since lfn and pfn are position-dependent parameters, the second comma must be present.

EXTEND EXAMPLE

Example 8:

```
jobcard
ATTACH(LF1,PROGLIB1,ID=ABC)
INPUT.
EXTEND(LF1)
7/8/9
(Object program deck)
7/8/9
(Data deck)
6/7/8/9
```

This job requests file PROGLIB1, created by example 2, to be attached; read, extend and modify permissions are granted automatically. The object program deck is loaded from the INPUT file and executed, using the data deck supplied. At the end of execution, extensions written to the permanent file will be cataloged by the EXTEND function.

PURGE EXAMPLES

PURGE BY LOGICAL FILE NAME

Example 9:

```
jobcard
ATTACH(LFN1,PFILE2,CY=10,ID=ABC,PW=CCCC)
PURGE(LFN1)
.
.
.
6/7/8/9
```

The file created in example 1 and renamed in examples 6 and 7 is attached to this job. Read and extend permissions are given automatically; control permission is granted by the password, CCCC. The PURGE card carries out the purge of the cataloged permanent file information before the remainder of the job is executed. The file then becomes a local file to this job, and cannot be modified or recataloged. When the job ends, this purged file will disappear as would any temporary file.

Example 10:

```
jobcard
ATTACH(LFN1,PFILE2,CY=10,ID=ABC,PW=CCCC,YYYY)
PURGE(LFN1)
CATALOG(LFN1,QFILE,ID=ABC,RN=0)
.
.
6/7/8/9
```

This example differs from example 9 in that the modify password is included; thus, all four access permissions are granted and logical file LFN1 can be recataloged as shown.

PURGE BY PERMANENT FILE NAME

Example 11:

```
jobcard
PURGE(DUMMY,PFILE2,PW=CCCC,ID=ABC)
.
.
.
6/7/8/9
```

When the PURGE request is issued for an unattached file, a logical file name is required for use by the permanent file manager. Since the modify permission is not granted (that password was omitted), the local file will not have that permission.

Example 12:

```
jobcard
PURGE(DUMMY,PROGLIB1,CY=21,SD=6,PW=XX,ID=ABC)
.
.
.
6/7/8/9
```

Cycle 21 of permanent file PROGLIB1, which is cataloged in subdirectory 6, is to be purged. An unattached logical file name parameter is required. Control permission is granted by password XX; read, extend and modify permissions are granted automatically. Only cycle 21 will be purged. In example 2, cycle 50 was cataloged in PROGLIB1; this purge will not affect cycle 50 in any way.

Example 13:

```
jobcard
PURGE(LFN,PROGLIB1,CY=50,SD=6,ID=ABC,PW=XX)
.
.
.
EXIT.
CATALOG(LFN,PROGLIB1,CY=50,CN=XX,ID=ABC)
6/7/8/9
```

This example is the same as example 12, except for the specification of the logical file name. It illustrates the capability to recatalog the purged file by respecifying the same cycle number and passwords, if program execution terminates abnormally. The CATALOG request after the EXIT. card would be honored only when the program using logical file LFN terminated abnormally.

RELOCATABLE PROGRAM LOADER 6

The relocatable loader loads relocatable user programs as well as relocatable system programs from job and library files into central memory for execution. CPLOADR may be called specifically by the LOADER control card; if the loader default option in SCOPE is set, CPLOADR will be called automatically.

Relocatable binary subprograms, assembled or compiled independently, are loaded into central memory by CPLOADR components together with other subprograms supplied by the user and/or obtained from library files. The action of the loader is to bind together all loaded subprograms to form a single executable program. A core (load) map for the bound program is produced by the loader and written out, as well as any diagnostic messages that may occur.

The loader can be used to divide a bound program into independent segments or overlays which may be called and executed as needed. The generation and writing of program overlays is handled by one component of the loader, LOADERV; another component, LOADERS is used to load and link program segments into memory for execution.

Loading is accomplished in the following general manner:

1. As a result of input to SCOPE, two PP routines, LOQ and LDQ, are called into assigned PP's and the relocatable program loader is called into central memory. LOQ does the initial loading of the main module, LOADERQ, into the upper end of the user's field length and performs certain initialization tasks. LDQ does physical input/output and loading of the relocatable programs into the user's field length. The five components of CPLOADR process the user's loader control cards, generate overlays, prepare the load map, perform linking and delinking operations and other related operations.

2. When all user-supplied object decks have been loaded, system libraries are searched to fill unsatisfied external references. Upon completion of the search, any remaining unsatisfied references are filled with out-of-bounds addresses.

3. A load map of the user's field length is produced and written to the job OUTPUT file. Program execution begins at the last loaded transfer address, or at any entry point specified on the EXECUTE control card.

The debugging aids SNAP and TRACE rely on the relocatable loader to insert program traps and to load required debugging subprograms from the system library. Both SNAP and TRACE may be used when loading and executing overlays and segments.

In summary, the following functions are performed by CPLOADR and its component parts:

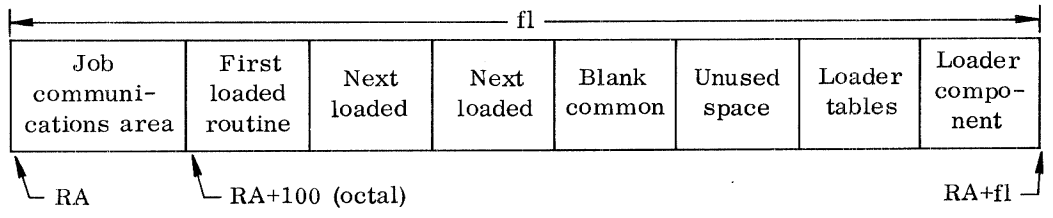
Relocatable program loading
Library searching and loading
Subprogram linking
Labeled and blank common assignment
Overlay/segment preparation
Overlay/segment loading
Core load map preparation
Error detection and diagnostic message preparation

LOADER INPUTS

SCOPE control cards describe the basic functions to be performed by the relocatable loader. Loader directive cards describe specific sequences the loader must follow in performing given functions. Both types of cards are processed by SCOPE, and the information they convey is either used by SCOPE to determine the loader components required or is passed on as input to the loader components.

Other inputs to the loader are the job and system files which supply relocatable binary subprograms, overlays, and segments. Associated files are indicated in the discussion of each loader card and directive in this section.

As shown below, loading of the user's field length proceeds from RA + 100 toward the upper end of the assigned memory space. If a blank common area is to be assigned, it is given the unused space following the last subprogram loaded. If labeled common is to be assigned, it is given space preceding the subprogram that first defines it. After loading is complete, if the program does not make calls to the loader during execution, the space occupied by the loader component and the loader tables may be released.



LOADER SELECTION

LOADER(name)

This control card is used to select a loader to be used with the job; name indicates the loader needed. An unrecognized name will produce a diagnostic message, and the job will be aborted. A loader default option is defined by installation parameter so that a given loader is selected automatically when this control card is omitted from a job deck.

The loader selection remains in effect until a loader card is encountered, or until the end of the job. When the default option is set for CP loader and the absolute (PP) loader is required by the job, the card is written:

```
LOADER(PPLOADR)
```

When the default option is set for the PP loader and the CP loader is required by the job, the card is written:

```
LOADER(CPLOADR)
```

When PPLOADR is used, a minimum field length of 4000(octal) words is required to contain LOADER. When CPLOADR is used, a minimum field length of 5400(octal) words is required to contain LOADERQ, LOADERS and buffer. The two loaders are externally compatible except for file positioning at the end of selective load operations; neither recognizes absolute code generated by the other loader.

LOADER CONTROL CARDS

PROGRAM EXECUTION (LOAD-AND-GO) CARD

The load-and-go control card provides the CP loader with a name which may be a file containing relocatable or absolute programs to be loaded into central memory. If the loader cannot find the name in the table of files attached to the job, it searches the system library for a program having that entry point name. If found, it is then loaded into memory and placed in execution.

The program execution card is written:

name(plist)

name is a binary program file name such as INPUT or LGO. It may also be the name of a library program, such as COMPASS, to be loaded and placed in execution. Parameters given on a user's program load-and-go card are passed to the loaded program for execution. plist is the optional parameter list which can be enclosed in parentheses or separated from name by a comma and terminated by a period.

Object (binary) decks produced by compilers and assemblers are output to a file named LGO, unless specifically directed to another file named in a parameter on the assembler/compiler control card. A binary deck included in the job automatically becomes a record on the job INPUT file.

If name is INPUT, loading begins at the current position of the file. All other files are rewound by the loader before loading. Loading terminates at the first empty record (double 7/8/9 end-of-record mark) or at end-of-file (6/7/8/9).

Example:

```
JOB.
FTN.
LGO.
7/8/9
(FORTRAN source deck)
7/8/9
(data deck)
6/7/8/9
```

The name FTN is not found in the job's file name table; the system program library is searched and the FORTRAN Extended compiler, identified by FTN, is found, loaded, and placed into execution. FORTRAN compiles the source deck and an object (binary) deck is output to the LGO file. The LGO. control card directs the loader to the file from which the object deck is loaded and placed into execution. While in execution, the program processes the data from the INPUT file. This example illustrates a compile-load-and-go operation.

Example:

```
JOB.
FTN(B=SAM)
SAM.
7/8/9
(FORTRAN source deck)
7/8/9
(data deck)
6/7/8/9
```

This example is a variation of the preceding example. The FORTRAN source deck is compiled and the object deck is output to a file called SAM, as directed by the parameter (B=SAM). The program execution card SAM. directs the loader to file SAM from which the object deck is loaded and placed in execution.

LOAD CARD

LOAD(lfn)

This card directs the loader to the file name given for lfn, and that file is loaded into memory. This card is load-only; no execution is implied. Generally, this card is used when several user-supplied binary subprograms located on different files are to be loaded and bound together before execution can begin.

When lfn is given as INPUT, reading of a binary subprogram begins from the current position of the file. All other files designated by lfn will be rewound automatically. Loading terminates at the first empty record (double 7/8/9 end-of-record mark) or at an end-of-file mark (6/7/8/9).

Loader directives may appear in the lfn file, but they must be placed before the binary subprogram. Such is the case with partitioned programs where the directives specify whether overlay, segment, or section processing is required of the loader. Loading is not completed; an EXECUTE, NOGO, or program call card must follow the LOAD card to complete the load. Also, another LOAD card may follow.

Example:

```
JOB.
FTN(B=SAM)
LOAD(INPUT)
SAM.
7/8/9
(FORTRAN source deck)
7/8/9
(object deck)
7/8/9
(empty record)
7/8/9
(data deck)
6/7/8/9
```

The object deck produced from the FORTRAN source deck is output to file SAM. The object deck supplied in the job could be from a previously compiled main program and contains its own 7/8/9 card which must be followed by a second 7/8/9 card in the job deck. The source deck could represent a subroutine being recompiled for use with the main program in the subsequent program loading and execution. The LOAD(INPUT) card directs the loader to the INPUT file from which it is to load the binary deck from the record immediately following the FORTRAN source deck. The control card SAM. is a load-and-go program execution card, therefore file SAM is rewound, the subprogram is loaded, and when the loading process is completed, execution is started.

LOAD (lfn) may not be used if lfn is an absolute file.

EXECUTE CARD

EXECUTE(name,plist)

This card indicates loading is to be completed and the bound program is to be put in execution. The EXECUTE card is required when it is not practical to use a load-and-go program execution card. Such would be the case when loading the main level of a segmented program or when loading a program and requesting that execution begin at an alternate entry point.

Both name and plist are optional; name is the entry point symbol at which execution is to begin. If not specified, the last transfer (entry point) address encountered by the loader will be used. Parameters to be passed to the program are given in plist.

Example:

```
    JOB.
    COMPASS(B=JOE)
    LOAD(JOE)
    LOAD(INPUT)
    EXECUTE(ENTR2)
    7/8/9
    (COMPASS source deck)
    7/8/9
    (object deck containing a 7/8/9 card)
    7/8/9
    (data deck)
    6/7/8/9
```

The COMPASS source deck is assembled and the resultant object deck is output to file JOE. The object deck supplied in the job containing its own 7/8/9 card and followed by a second 7/8/9 card is loaded by the LOAD(INPUT) card after the object deck on file JOE is loaded. Execution is to begin at the entry point ENTR2, specified in the EXECUTE card, after the loading operation is complete.

NO EXECUTION CARD

NOGO.

This card permits loading and binding a program and preparation of a load map, but it inhibits execution of the bound program. It is most useful in mapping large production programs or partitioned programs. When this control card is used to create overlays, the absolute overlays will be generated on the specified files, which can be used for execution or saved for later execution in another job. After loading has been completed, execution is bypassed; and the next control card in the job is processed.

MAPPING CONTROL

MAP(option)

The MAP control card may appear in any job deck prior to an EXECUTE card or a program execution card. An installation default option will prevail when a MAP control card does not appear in a job deck. The selected or default option will remain in effect until the end of the job or until replaced by another MAP control card. A load map, with explanations of its contents, is included at the end of this section. The MAP options are:

MAP(ON)	This card specifies a load map is to be produced and written on the job's OUTPUT file when loading is completed. Load maps are never produced for system library programs, such as FORTRAN or COMPASS, or for any absolute program.
MAP(OFF)	The OFF option indicates a load map is not to be produced. This option is of value when loading production programs or when the debugging of frequently used programs is not anticipated.
MAP(PART)	This option causes a partial load map to be produced. It differs from the ON option in that entry point addresses are omitted from the load map.

FIELD LENGTH REDUCTION

REDUCE.

When SCOPE encounters the REDUCE card, it sets a flag for the loader to reduce the job field length after loading is completed and before execution is begun. The field length is reduced to the highest word address assigned to the bound program, rounded upward to the next higher multiple of 100(octal). The reduced field length will remain in effect until the end of the job or until an RFL card is encountered. The REDUCE card is not honored if an external reference is made to LOADER, or if the system is loading an absolute program (such as COMPASS) from the library, or when overlays are being created.

The REDUCE card may be placed anywhere before an EXECUTE control card. Any succeeding runs in the job that require a larger field length will be terminated unless an RFL card is used to increase the field length.

Example:

```
LOAD(ABLE)
REDUCE.
EXECUTE.
RFL(60000)
LOAD(BAKER)
REDUCE.
EXECUTE.
CHARLIE.
```

The field length into which program ABLE was loaded is reduced before ABLE is executed. Before program BAKER can be loaded, the field length is increased to 60,000(octal) words to accommodate the loader tables and other routines. After BAKER is loaded, the field length is reduced once again. The reduced field length is known to be sufficient to contain the loader and tables to load program CHARLIE. Such knowledge can be obtained by previously loading CHARLIE with the NOGO option to obtain a load map in which the required field length is indicated.

OVERLAYS AND SEGMENTED PROGRAMS

A segmented program is made up of a group of interrelated program sections, any number of which may be linked together and executed as a whole. An overlay program is made up of a maximum of three related program sections which may be loaded and executed as a whole program. Program segments are linked dynamically at execution time; overlays are bound together into a fixed composition when they are generated.

SEGMENTATION

A segment is a group of relocatable subprograms to be handled as a program unit. A segment may contain sections wherein the user defines the subprograms to be included in given sections. The program dynamically selects the segment required to form an executing program, linking additional segments as they are required and unlinking segments no longer needed for further execution of the program.

Some pre-planning on the part of the programmer will be required to determine the combination of subprograms and sections that form segments. Program segmentation is costly in terms of linking/delinking time and the space required to keep the loader components in memory throughout segmented program execution.

SEGMENT LEVELS

Segments are assigned octal level numbers by the user, in the range of 0 to 77. Level numbers identify the program level at which each segment is to be linked or delinked during program execution. Each segmented job must contain a main segment at level zero. It must reside in central memory at all times and it may not be delinked. All other segments are called into central memory at execution time by the user program, and may be assigned in any order of level number.

Care must be exercised in the sequencing of segment levels. Loading a segment at a level which is numerically less than or equal to a level currently occupied will cause all segments at and above that level to be delinked. As an example: a segment is to be loaded for execution at level 5. Segments have been previously loaded and linked to the program at levels 2, 4, 6, 7, and 8. The level 6, 7 and 8 segments will be delinked from the program, and the new segment loaded and linked at level 5.

Labeled common in segments other than the main level may not be used to pass data between segments having higher level numbers, but it may be used to pass data between subroutines within the same segment. Blank common can be used to communicate between segments; however, its maximum size must be specified in the level zero segment.

Segment level numbers need not be assigned in consecutive order. For instance, if the user requests segments to be loaded at levels 2, 3 and 5, they will be loaded in that order. The number of segments in the program at any one time is limited only by the amount of memory available in the user's field length.

SEGMENT LINKING

When a segment is loaded into core, any unsatisfied external references from lower level segments are linked to entry points in the newly loaded higher level segment, if possible. External references in the newly loaded segment are linked to entry points in the lower level segments, if possible. When a segment is linked to a program that already contains segments at an equal or higher level, those segments are delinked and removed. Thus, previously satisfied external references may become unsatisfied, and an attempt will be made to link them to the newly loaded segment.

Example:

Segments loaded at levels 0, 1, 2, and 3 are linked in memory as a program. A routine at level 1 references routine DOIT in level 3. Then a new segment is requested to be loaded at level 2. The segments at levels 2 and 3 are delinked and removed, leaving external reference DOIT unsatisfied in level 1. However, the newly loaded level 2 segment contains a routine called DOIT which is then linked to the external reference in the level 1 segment.

SEGMENTATION DIRECTIVES

All segmentation directives must appear before any of the subprogram decks to be incorporated into segments or sections.
The first segment of a program must be the main segment, having level number zero. Therefore, the SEGZERO directive is written:

SEGZERO(sn,pnlist)

In the parameter string, sn is a segment name of up to seven characters; and pnlist consists of one or more subprograms and/or section names which make up the main level segment. All sections named in the list must have been defined prior to the SEGZERO directive. Subprogram names in the list may be user or library subprograms.

Additional segments are defined by the directive:

SEGMENT(sn,pnlist)

The parameters in this directive are the same as described for SEGZERO. All subprograms named must reside in the same file. A segment defined in the user's program need not be defined by a SEGMENT directive; however, a SEGZERO directive is always required.

The SECTION directive is used to define a section of a segment.

SECTION(sn,pnlist)

sn is the name of the section and pnlist names one or more subprogram which is to belong to the section. If more than one card is needed to list all the subprogram names, consecutive SECTION directives with the same sn name may appear. All subprograms named in pnlist are loaded whenever the named section is loaded. All SECTION directives must appear prior to the SEGMENT cards which refer to the named sections.

As an example, the following directives describe the main and subordinate segments as well as the sections which are to be made part of the segments when they are loaded:

```
SEGZERO(MAIN,ALFA)
SECTION(ABLE,AA,AB)
SECTION(BAKER,BB,BC)
SECTION(CHARLIE,AA,CC)
SEGMENT(A,ABLE,BAKER)
SEGMENT(B,CHARLIE,BAKER)
Subprogram ALFA  \
Subprogram AA     |
Subprogram CC     |
Subprogram AB      > binary decks
Subprogram BB     |
Subprogram BC     /
```

The main segment is to contain the object subprogram ALFA; segment A is to contain sections ABLE and BAKER which are made up of the subprograms AA, AB, BB and BC. Segment B is to contain subprograms AA, CC, BB and BC which make up the sections CHARLIE and BAKER.

SEGMENT LOADING

Segments may be called for loading and linking by program execution (call) control cards to load segment zero or by loader function requests issued from within the program. In FORTRAN-compiled programs, the CALL SEGMENT statement requests loading and linking of a segment. The statement is described in both the FORTRAN and FORTRAN Extended Reference Manuals. In COMPASS language programs, the LOADER macro is used.

LOADER MACRO

LOADER param

59	29	17 0
Zeros	RJ	LOADER
		param

A program may request service from the loader with this macro. Param is the location at which the user has established a parameter list for the load sequence. LOADER is an external symbol which is satisfied by the loader and which will ultimately reference an entry point in LOADERQ if the CP loader is being used, or LOADER if the PP loader is being used.

Unlike control card requests for loader activity, user requests do not cause a file to be rewound. Instead, the user is responsible for positioning all files before issuing a user request. A request for a full file load loads programs only to the end-of-file. In all other cases, the file is searched end-around for the specified programs. If all programs are located, the file will be positioned immediately following the last program loaded. If all programs are not located, a fatal error flag is returned to the user; CPLOADR will leave the file positioned at end-of-file. PPLOADR will leave the file positioned at its original starting point.

59	53	47	43		41		39		37		35		17 0
lfn (logical file name)													sec
seg1	seg2		r	p	u	v	m	k	s	f	c	lwa	fwa

lfn — Field contents may be one of the following:

Name of file from which programs will be loaded (sec may or may not be 0)

Name of entry point in a program (sec = 0)

Program name (sec = 0)

Zero (sec $\neq 0$)

sec — If non-zero, the location of a list of sections, a segment name, or a list of subprograms to be loaded as a segment; if segment loading is not requested, a list of subprograms to be loaded from file lfn. Names may not exceed seven characters. The list may be empty. It is terminated by a word of zeros.

Each entry in the sec list has the following format:

59	17 0
subprogram name	Zeros

seg1	Segment level if s ≠ 0 and v = 0. Primary overlay level if s = 0 and v ≠ 0.
seg2	Secondary overlay level if v ≠ 0.
r	Reset bit. If r ≠ 0 all loader tables are cleared before loading; normal loading only.
p	Partial map bit. If p ≠ 0 a one-line partial core map is given.
u	Library flag. When u ≠ 0, and v = 0, sec refers to a list of externals to be loaded from the system library. When u ≠ 0 and v ≠ 0, an overlay will be loaded from the system library. In this case sec = 0 and lfn = program name. When u ≠ 0, v = 0, s ≠ 0, and lfn = 0, sec points to a list of sections, a segment or list of programs to be loaded as a segment from the library.
v	Overlayflag. If v ≠ 0, an overlay load operation is requested.
m	NOMAP flag. If m ≠ 0, all maps are suppressed.
k	Search key. If k ≠ 0, lfn is the name of an entry point, and the key is used to find the address of a previously loaded entry point; no loading is performed.
s	Segment flag. If s ≠ 0, segment loading is requested.
f	Fill flag. If f ≠ 0 unsatisfied external symbols are filled with out-of-bounds references.
c	Complete flag. If c ≠ 0, loading is to be completed by loading relocatable subroutines from the system library. The origin and length of blank common will be established. Until loading has been completed, the length may vary between subprograms.
lwa	Last word location, relative to RA, available for the loading operation. If lwa = 0, the limit of program loading is the first word of the loader tables stored at descending addresses starting at fwa of LOADER. For blank common declarations, lwa is designed as RA + fl-20(octal).
fwa	Initial location, relative to RA, at which loading is to begin. If fwa = 0, loading begins at the next available location as determined by the current state of the loading operation.

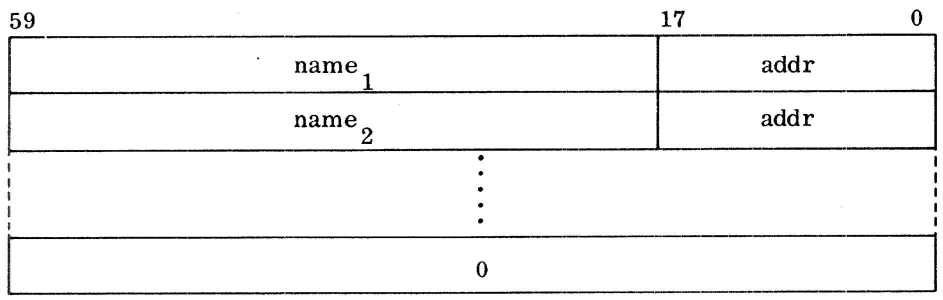
REPLY FROM LOADER

When LOADER has completed the requested operation (loading is not necessarily complete), LOADER signals the caller by setting the parameter list as follows:

	59	53	37	36	35	17 0
Word 1	0					
Word 2	lev		ne	fe	aa	ea

lev	Level at which segment was loaded. lev = 0 if segment loading not requested.
ne	Non-fatal error flag. ne ≠ 0 if the following loading errors are detected by LDR: Unsatisfied externals if c = 1. Duplicate occurrences of named program; second and subsequent occurrences are ignored. No entry point for a named transfer.
fe	Fatal error flag. fe ≠ 0 if the following loader errors are detected: Improper deck structure. Improper parameter specification. Requested file name, program name, or entry point not found.
aa	Entry address. aa = 0 if less than two named XFER's were encountered. aa = address of next-to-last name if more than one named XFER was encountered.
ea	Entry address. If k = 0, ea is the location (relative to RA) of last encountered named entry specified in a XFER table. If more than one named XFER is encountered, the last one is in ea, and the preceding entry in aa. If k = 1, ea is the location (relative to RA) of the named entry point. If v ≠ 0, ea is the entry point to the overlay. If ea = 0, no name was found.

If sec ≠ 0, the list of entry points and/or subroutines to be loaded from the library contains the address at which each name is loaded. If the name was not loaded, the address is zero. The list then has the form:



USER REQUEST PROCESSING

Examples of parameter lists to be processed by the loader are given below:

Load from File:

```
lfn = name of file
sec = 0
v = 0
k = 0
s = 0
```

The file is not rewound before loading. If no file is found, the system library is searched as in the following example. Subprograms will be loaded from lfn until end-of-information is encountered. seg1 and seg2 are ignored. If c $\neq 0$, loading will be completed and unsatisfied externals printed.

Load Named Entry:

```
lfn = name of entry point in a subprogram or the name of a subprogram.
sec = 0
u = 0
v = 0
k = 0
s = 0
```

The file name table is searched first; if it contains the lfn, a load from file is performed as in the first example; if not, the system library is searched. seg1 and seg2 are ignored. If c $\neq 0$, loading will be completed.

Load Segment from File:

The segment defined by the list at sec will be loaded from lfn at level seg1. If seg1 $>$ current segment level, the segment will be loaded at the current level $+1$. If seg1 $\leq$ current level, segments at a higher level will be delinked. If a subprogram specified in the segment list is not located after the entire file has been searched, the fatal error flag will be set. lfn is not rewound prior to loading.

```
lfn = name of file
sec = address of list containing a segment name or section names and subprogram
names only
seg1 = level at which segment is to be loaded
u = 0
v = 0
k = 0
s = 1
```

If c $= 0$, loading will be completed with the origin and length of blank common established. If c $\neq 0$, loading will be completed normally. f $\neq$ will cause unsatisfied external references to be set to out-of-bounds references.

Load Named Subprograms from File:

The list of subprograms specified by the list at sec will be loaded from lfn and the system library.

```
lfn = name of file
sec = address of list
seg1 and seg2 ignored
u = 0
v = 0
k = 0
s = 0
If c ≠0, loading will be completed.
```

Load Overlay from File:

```
lfn = name of file
sec = 0
seg1 = primary level
seg2 = secondary level
u = 0
v = 1
r, p, m, k, s, f and c are ignored.
```

The overlay file (constructed during the initial load from overlay cards and binary text) is searched for the unique identifier segl, seg2. If fwa = 0, the overlay will be loaded at the address where it was created, otherwise the load will be at fwa. The absence of such an overlay will cause the loader to set the fatal error flag. No map is produced.

Load Overlay from System Library:

```
lfn = program name
sec = 0
seg1 = primary level
seg2 = secondary level
u = 1
v = 1
r, p, m, k, s, f, and c are ignored.
```

SEGMENTATION EXAMPLE:

```
(job card)
REQUEST(SEGFILE,MT)
FTN(B=SEGFILE)
SEGFILE.
UNLOAD(SEGFILE)
7/8/9
SEGZERO(HELP,STARTA)
SEGMENT(HELPA,ALLAN,SAM)
SEGMENT(HELPB,JACK,JOHN)
SUBROUTINE SAM     \
SUBROUTINE JACK    |
SUBROUTINE JOHN    > FORTRAN Source Decks
PROGRAM STARTA     |
SUBROUTINE ALLAN   /
7/8/9
(data deck)
6/7/8/9
```

In the above job, a magnetic tape is requested, and it is to have the logical file name SEGFILE. The FORTRAN compiler will output object decks to this file. After compilations are completed, the program execution control card SEGFILE will cause segment zero to be loaded for execution. Following the first 7/8/9 card is a group of loader directives and FORTRAN source decks. Since the directives are not recognized by the compiler, they are copied directly to the binary output file SEGFILE from which they will be read by the loader. This example includes only source decks; however, if object decks were interspersed with source decks, these too would be copied directly to the SEGFILE. The compiler handles only what it recognizes; all else is copied to the binary output file.

OVERLAY GENERATION

An overlay is a portion of a program that is bound together into a fixed composition and written out to a retention file. Because of program size and complexity, it may be beneficial to organize a program into overlays to make more efficient use of the user's field length and available storage. Since the overlay is always loaded into the field length at the same relative address at which it was created, the overlay loader is greatly reduced in size as normal loading functions are completed when the overlay was created.

When using overlays, the programmer must organize his program so as to retain the more commonly used subprograms in the main level overlay and the lesser used subprograms in the overlays which will reside in memory as temporary overlays. When each overlay is created, the loader will complete the loading operation by loading library and user subprograms and binding them together. The resultant overlay will be in a fixed format, in that the internal members are fixed in their relationship to one another. The entire overlay has a fixed origin address within the field length and therefore is not relocatable within the field length. The overlay loader, therefore, simply reads the required overlay from the overlay file and places it into the field length at its pre-established origin.

Overlay jobs are self-contained bound programs that can be loaded and executed without requiring execution time for linking and delinking purposes; linking was done by the loader when the overlays were created.

OVERLAY LEVELS

Each overlay is made up of a main program and, if desired, one or more subprograms. Each overlay is identified by an ordered pair of integer numbers (i,j) where i represents the overlay's primary level and j represents its secondary level. A main overlay denoted by level (0,0) must exist in every overlay job. This overlay is always in core and may call into core any primary level overlay. A primary overlay is denoted by a non-zero primary level number and a zero secondary level number: (1,0), (2,0),... Primary overlays must be called into core by the main overlay. A father-son relationship exists between primary and secondary overlays. All secondary overlays related to a primary overlay are denoted by the parent primary level number and a non-zero secondary level number. Secondary overlays associated with primary overlay (1,0) might be: (1,1), (1,2), (1,5),... A secondary overlay may be called into core by its associated primary overlay or by the main overlay.

OVERLAY LINKAGES

Overlays are created by the loader and placed on a mass storage device or tape file in their bound, or absolute, form. Linkage within an overlay is established during this generation. Linkage with subroutines in other overlays are established in one direction only. A routine in a secondary overlay may call any subroutine in itself, in its parent primary overlay or in the main overlay. A routine in a primary overlay may call only other routines in itself or in the main overlay. When a modification is made to a subroutine, these linkages may require the entire set of overlays to be regenerated.

Communication between overlays can be accomplished only by using labeled or blank common. Any element of a labeled common block in the main (0,0) overlay may be referenced by any higher level overlay. Any labeled common declared in a primary overlay may be referenced only by the primary overlay and its associated secondary overlays and not by the main overlay. If blank common is used for communicating between overlays, the user must ensure that sufficient field length is reserved to accommodate the largest loaded overlay plus blank common. Each level of the overlay must use data stored in blank common in exactly the same format, since no linkage is provided between the different levels of overlay and blank common at execution or load time.

OVERLAY DIRECTIVES

The loader is instructed to create overlays by the overlay directives encountered in the input file to the loader. An OVERLAY directive is placed in front of the main program of each group of one or more subprogram decks that comprise the overlay. The directive format is:

OVERLAY(lfn,i,j,Cnnnnnn)

lfn is the logical file name of the retention file on which the generated overlay is to be written and i and j are the primary and secondary level numbers to be given to the overlay. In segmentation, the level number is assigned dynamically; in overlay creation, the level numbers are assigned when the overlay is created and will not change. The first overlay card must have a named lfn. Subsequent cards may omit it, indicating that the overlays are related and are to be written in the same lfn.

A different lfn on subsequent cards results in generation of overlays to the new lfn.

The source file for overlay generation may not be used as the lfn on any overlay card. The loader starts writing on the first overlay to the output file before all overlays on the source file have been processed.

Cnnnnnn is an optional parameter consisting of the letter C and a six-digit octal number, which indicates the overlay is to be loaded nnnnnn words from the start of blank common. With this method, the programmer can change the size of blank common at execution time. Cnnnnnn cannot be included on the 0,0 overlay loader directive. If this parameter is omitted, the overlay is loaded in the normal manner.

OVERLAY DECKS

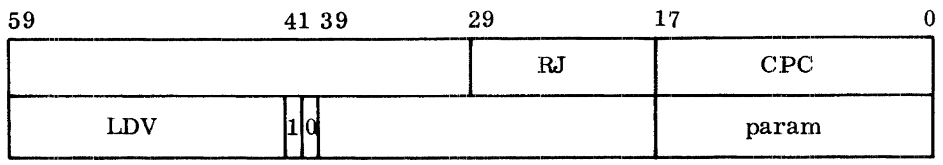
The relocatable binary decks immediately following OVERLAY, up to the next OVERLAY control card or an end-of-file, comprise the overlay deck. When the overlay deck has been loaded, loading is completed by satisfying undefined external references from the system library. The overlay and its identification are written as the next logical record in the file. Writing to the overlay file takes place when a directive is encountered which specifies a level which would overlay a level currently residing in memory. Writing also takes place when the last overlay has been created.

Each overlay has a unique entry which is the last transfer address (XFER) encountered in the overlay subprograms during preparation. External references which cannot be satisfied, even by the system library, result in job termination after loading is completed and maps are produced for all overlays. References to entry points in the main overlay may be made from primary and secondary overlays. References to entry points in a primary overlay may be made only from an associated secondary overlay. Similarly, common blocks defined in a lower level overlay can be referenced from a higher level overlay. Data can be preloaded into a labeled common block if the overlay which defined the common block has not been written to the overlay file.

OVERLAY LOADING

Each overlay may reside on a different file, or all on the same file. The file name is specified as the lfn parameter in the OVERLAY directive. The main overlay must be brought into memory by the loader by the appearance of a program execution card in the job deck. Thereafter, additional overlays are called into memory by the executing program. FORTRAN-compiled programs use the CALL OVERLAY statement, described in the FORTRAN and FORTRAN Extended Reference Manuals. COMPASS assembled programs may use the LOADREQ system function request.

LOADREQ param



This request is used to load overlays from a user file or the system library.

The LOADREQ request results in a call to the PP overlay LDV. The functions of LDV depend upon the value of param:

1. A non-zero param has the same significance as when used in the LOADER request. LDV examines the two-word entry pointed to by param which has the same format as the entry described under the LOADER service function.

   If the v bit in word 2 indicates that this operation is not an overlay load, LDV calls LDR. Results are unpredictable.

   If this is an overlay and the u bit in word 2 indicates a user file, LDV calls LDR to perform the operation.

   If this is an overlay and the u bit in word 2 indicates the system library, LDV loads into the user's area the overlay named in bits 18-59 of word 1 of the two-word entry pointed to by param. If the named overlay does not appear in the system library, LDR is called to handle the error condition.

2. If param is zero or omitted, this call is to the loader for loading for a file. The name of the file must be specified in RA + 64. LDV calls the loader for the load-and-go operation selected by the most recent LOADER control card encountered in this job; or if no LOADER control card had been used for this job, the loader is selected by default.

After the overlay has been loaded, LOADREQ generates reply information for the user in the same format as the LOADER request.

OVERLAY JOB EXAMPLES

Creation of an overlay tape file:

```
(job card)
FTN(B=OVERBIN)
REQUEST(ABCD,MT)OUTPUT
LOAD(OVERBIN)
NOGO.
UNLOAD(ABCD)
7/8/9
OVERLAY(ABCD,0,0)
(FORTRAN Source Deck)
6/7/8/9
```

This simple example shows the creation of a tape containing a one-overlay binary program file. The FORTRAN compiler will copy the OVERLAY(ABCD,0,0) directive and the compiled object deck out to the temporary file OVERBIN. The REQUEST(ABCD,MT)OUTPUT control card directs that a magnetic tape is to be made available to contain the file ABCD. The comment, OUTPUT, can be displayed on the display console as a signal to the operator that the tape to be mounted is to be made available for output. The LOAD(OVERBIN) control card requests the loader to load the program from file OVERBIN. As the loader reads file OVERBIN, it encounters the OVERLAY directive, which instructs it to create a main level (0,0) overlay from the program and write it out to file ABCD, which is the magnetic tape file. The NOGO. card indicates that execution is not to take place after loading is completed. When the loader has completed writing the overlay to file ABCD and the job is advanced beyond the NOGO card, the tape is unloaded as directed by the UNLOAD control card after the file is closed and rewound.

Once the overlay tape file has been created, the user can submit it along with his data for execution at a later time. In effect, it becomes the user's production program tape. The deck setup for such a job might be:

```
(job card)
REQUEST(ABCD,MT)
ABCD.
UNLOAD(ABCD)
7/8/9
(Data deck)
6/7/8/9
```

Deck structure for overlay generation and multiple executions:

```
(job card)
COMPASS.
LGO.
FRANK1.
7/8/9
OVERLAY(FRANK1,0,0)
(Source decks)
OVERLAY(JOHN,1,0)
(Source decks)
OVERLAY(JOHN,1,1)
(Source decks)
7/8/9
(Data deck)
7/8/9
(Data deck)
6/7/8/9
```

In this job, the COMPASS control card will cause the source decks to be assembled; the standard system default is used to write the object decks to the file LGO, interspersed with the OVERLAY loader directives. The LGO. control card causes loading from the LGO file and the loader will process the OVERLAY directives. The main overlay is to be written on file FRANK1; the primary overlay (1,0) and its secondary overlay (1,1) are to be written to file JOHN. Both files FRANK1 and JOHN are temporary files to this job and will be released when the job is terminated; therefore, it is reasonable to assume that this job is for preliminary overlay creation and checkout before a removable overlay file is prepared.

When all overlays have been created, execution begins using the first data deck in the job. Since a second LGO card would cause the entire process to be repeated including unnecessary re-creation of all overlays, processing of the second execution can be expedited by using the control card FRANK1, instead of LGO. Control card FRANK1. causes the main overlay to be loaded quickly from file FRANK1 and execution of the job will begin to process the second data deck.

LEGEND FOR CORE MAP:

1. Time of day (24-hour clock) when program was loaded.
2. Type of loading: NORMAL, OVERLAY, or SEGMENT.
3. Overlay or segment level numbers (blank for normal loading).
4. TYPE is CONTROL if load was requested via control card; otherwise TYPE is address of user's call to loader.
5. Meaningless if TYPE is CONTROL; otherwise a 2-word loader request appears above USER---+ +---CALL.
6. First word address of loaded user program.
7. Last word address + 1 of loaded user program.
8. First word address of blank common.
9. Length of blank common.
10. First word address of loader and loader tables. Table address may be reassigned to blank common.
11. Name and first word address of all subprograms loaded. These may be user and/or library subprograms.
12. Labeled common name and first word address of each block. Each is listed in the same line as the subprogram in which the block is first defined.
13. Entry point name and address for each subprogram.
14. List of subprograms referring to entry points and addresses at which each reference is made.
15. List of unsatisfied external references and address at which each was made.

```
                (1)       (2)      (3)       (4)             (5)        (5)                      (6)      (7)       (8)       (9)
CORE MAP   16.52.31.   NORMAL              CONTROL                                              000100    011076    000000    000000
 ---TIME---LOAD MODE --L1--L2-----TYPE--------------USER---+---CALL--------------FWA LOAD--LWA LOAD--BLNK COMN--LENGTH--
 (10)  FWA LOADER  054010  FWA TABLES  052230
       -PROGRAM----ADDRESS-                        --LABELED---COMMON--
       GEN         000720    (11)                   (12)  FP          000100
                                                          DP          000244
                                                          IN          000554
       DUMMY       003045
       DEBUG       003052
       SYSTEM      005154                                 SCOPE2      005154
       OUTPTC      006231
       SIOS        006322
       KODER       007651
       GETBA       011057
       --ENTRY-----ADDRESS-                                      REFERENCES
       GEN         000721    (13)
       DUMMY       003045          GEN       000763    (14)

       POINT1      003046
       POINT2      003047
       POINT3      003050
       TSCARD      003052
       SETADR      003722
       WRDEBUG     004261
       SNACE       004666
       STORE       004440
       RESTORE     004542
       RBO         004603
       RAO         004613
       RXO         004623
       TSSEGUP     004362
       OVFLAG      003713
       ZZGETFL     004426
       PCNTR       004244
       ZSQUIZH     004253
       QBNTRY      005155          GEN       000722

       SYSTEM      005361          OUTPTC    006306
                                   KODER     010663
       ---UNSATISFIED EXTERNALS-----    (15)                  REFERENCES
    -2499     -2496     -2491     -2484     -2475     -2464     -2451     -2436
    -2419     -2400     -2379     -2356     -2331     -2304     -2275     -2244
    -2211     -2176     -2139     -2100     -2059     -2016     -1971     -1924
    -1875     -1824     -1771     -1716     -1659     -1600     -1539     -1476
    -1411     -1344     -1275     -1204     -1131     -1056      -979      -900
     -819      -736      -651      -564      -475      -384      -291      -196
      -99         0       101       204       309       416       525       636
      749       864       981      1100      1221      1344      1469      1596
     1725      1856      1989      2124      2261      2400      2541      2684
     2829      2976      3125      3276      3429      3584      3741      3900
     4061      4224      4389      4556      4725      4896      5069      5244
     5421      5600      5781      5964      6149      6336      6525      6716
     6909      7104      7301      7500
```

DEBUGGING AIDS 7

Debugging aids include routines named DUMP, SNAP, and TRACE; requests for their use are submitted with normal jobs. User programs or subroutines should not use these names as any user references to them will reference the system routine.

DEBUG

The DEBUG control card is required when a labeled or change format dump is requested or when debugging aids are used with overlay or segment jobs. The DEBUG control card applies to all subsequent loads and executions within a job. Dump formats are described under the DUMP section. Any absolute program loaded completely from the system library cannot use debugging aids. Such programs can be debugged only when they have been loaded from a user file.

DEBUG FILE

Upon completion of loading, a local file is created. This file, named DEBUG, contains the loader table information necessary for formatted dumps. It is updated as each segment is loaded. During user loading, execution does not affect or is not affected by the DEBUG card except for user segment loading.

DEBUG CONTROL CARD

DEBUG (C,T,S)

C	A labeled dump followed by a change dump is output when the DMP card is encountered; if C is absent, only a labeled dump is produced. The change dump is not produced for overlay or segment loading.
T	In overlay mode, TRACE and SNAP routines are loaded with the (0,0) overlay; in segment mode, TRACE with SEGZERO.
S	In overlay mode, TRACE and SNAP routines are loaded with the (0,0) overlay; in segment mode, loaded SNAP with SEGZERO.

DEBUG cards with both C and T parameters or with both C and S parameters cannot appear in the same job. Such cards do not terminate the job, but the change dump is not processed. Only a DEBUG card with no parameters produces a labeled dump without a change dump.

DEBUG USE

OVERLAY LOADING

If a DEBUG card is included when an overlay is prepared, the loader inserts a record on the overlay file following the overlay. This record contains loader table information necessary for traces, snapshot, and formatted dumps. When the overlay is loaded, table information is extracted from the overlay file and placed on the DEBUG file.

DUMP

The function of the DUMP program is to enable a programmer to obtain as printed output when the job terminates which represents the contents of selected memory areas containing job-related information. The dump can be obtained in a variety of formats, and can be produced upon either normal or abnormal termination of the job.

TYPES OF DUMPS AND DMP CONTROL CARDS

Five types of dumps are possible: relative, control point area, exchange package, absolute, and ECS. The first four types are identified accordingly at the beginning of each dump printout.

The dump control card is written in any of four forms:

DMP(,from,,to)
DMP(from,to)
DMP(to)
DMP.

Portions to be dumped are selected by specifying a first word address (from parameter) and a last word address (to parameter) on a DMP control card, in a console entry, or from a CP program. If the from parameter exceeds the field length, no dump results unless an absolute dump has been requested.

RELATIVE DUMP

This dump may include all or any part of the job field length (FL) from RA through the last word address (RA+FL). The format of the dump may be unlabeled, labeled, or change. If the last word address exceeds FL, FL will be substituted instead.

Format: *DMP(x,y)*
DMP(y)
DMP(,x,,y)

If the to/from addresses specified by x and y are numeric, they must be expressed as octal numbers. If only y is specified, x is assumed to be zero and the dumped area extends from RA through the address specified by y relative to RA. When x or y represents a common block name, it must be preceded by an empty parameter.

Example:

```
DMP(500,6500)
```

Produces a dump of locations 500 through 6500 relative to RA. If a DEBUG card is included, a labeled dump of this area is produced.

```
DMP(6000)
```

Produces a dump of the locations from 0 through 6000 relative to RA.

```
DMP(2340,CAT)
```

Produces a labeled dump of the area starting at relative address 2340 through program CAT, provided a DEBUG card is present in the job's control card stream.

CONTROL POINT AREA DUMP

This dumps 200(octal) words of the control point area. The values specified for the x and y parameters must be equal and must not be zero.

Format:	*DMP(x,y)*
Example:	`DMP(1,1)`

EXCHANGE PACKAGE DUMP

The resulting dump consists of the following three parts:

The exchange jump package consisting of P, RA, FL, RAECS, FLECS, EM, and registers A0...A7, B0...B7, X0...X7. The contents of the locations to which the A and B registers point are dumped also if they are within the FL. If not, the message **OUT OF RANGE** appears.

The first 100(octal) locations (RA to RA+77) of the FL.

100(octal) locations before and after the address to which the P register points, provided they are within the FL.

If the P register equals zero, the P address in bits 30-47 of RA+0 will determine the locations to be dumped.

If the P register or the P address in RA+0 is less than 200(octal), the first address dumped will be 100(octal).

If the P register and P address in RA+0 are both zero, part 3 will not be dumped.

Format:

DMP.
DMP(0)
DMP(0,0)

ABSOLUTE DUMP

All locations in central memory may be dumped whether they are within the FL or not. The x parameter must be six octal numbers and the first must be 4, 5, 6, or 7. DMP subtracts 4 from this digit to determine the absolute address to be dumped.

If the y parameter is less than 400000(octal), it is treated as y+400000. If only the y parameter is specified, it must be greater than 400000; x is assumed to be 400000. For example, DMP(400000,405000) dumps the first 5000 words in central memory. If the parameters exceed central memory (CM) core size, they will be adjusted to avoid wrapping around central memory.

x Parameter	Maximum CM Address Dumped
4xxxxx	77 777 (32K)
5xxxxx	177 777 (65K)
6xxxxx	277 777 (98K)
7xxxxx	377 777 (131K)

Format: *DMP(x,y)*

No absolute dumps are permitted if the installation parameter IP.DEBUG (debug permission bit) is turned off (zero).

ECS DUMP

A labeled dump is produced of the specified area of ECS.

Format: *DMPECS(x,y,f,lfn)*

ECS from location x to y is dumped where x is the closest multiple of 10(octal) less than or equal to x, and y is the closest multiple of 10(octal) greater than y-1.

f selects the print format per line according to the value:

0 or 1	4 words in octal and in display code.
2	2 words in octal parcels and in display code.
3	2 words in octal bytes and in display code.
4	2 words in octal and in display code.

lfn specifies the dump file; if absent or zero, file OUTPUT is assumed.

DUMP FORMATS

Upon normal or abnormal termination, three dump formats are available: unlabeled, labeled, or change. Each dump printout on the job OUPTUT file is prefaced by a line stating the type of dump and including the DMP control card that requested it.

UNLABELED DUMP

This standard dump option does not require use of the DEBUG control card. This option will produce an octal core dump when the DMP control card is encountered. Each line of storage printed contains an address and the contents of one to four central memory words starting at that address. Printing of a central memory word is suppressed when that word is identical to the last word printed. When the next non-identical word is encountered, its location is printed and marked by a right-pointing arrow.

LABELED DUMP

The format of this dump printout differs from the unlabeled format in two ways. As the origin of a common block or subprogram is encountered, the associated name is printed. In addition, two relative addresses appear. The first is relative to the origin of the subprogram or common block; the second is relative to the reference address (RA).

A labeled dump cannot be obtained without a DEBUG control card in the job's control card stream before the program is loaded. The DEBUG file is used to locate the origin and names of the subprograms and common blocks. Parameters on the DMP card for a labeled dump may be either symbolic or octal numbers; the two may be combined. (A common block name is preceded by an empty parameter.) Entry point names and unlabeled common blocks may not be used as parameters.

The dump begins at the origin of the first parameter name and continues through the space occupied by the subprogram (or common block) given as the second parameter. The second parameter origin must be greater than the first parameter origin.

Example:

```
DMP(ALPHA,CAT)
```

Produces a labeled dump of program ALPHA and all locations through program CAT, provided a DEBUG card is present in the control card stream. Intermediate programs encountered are identified.

When a job is in overlay or segment mode, the DEBUG file is updated with the loading of each overlay or segment.

CHANGE DUMP

The change dump prints out a list of core locations which have changed from their initial values. When a job begins an execution phase, a core image of the entire field length is written on the DEBUG file. The image is compared with the contents of memory at the time of termination. The contents of changed locations are listed on the job OUTPUT file. A labeled dump always precedes a change dump.

Because the DEBUG file is needed to produce this dump, a change dump can be obtained only if a DEBUG(C) control card is in the job's control card stream before the program is loaded. A change dump will not be produced during overlay or segment mode.

Change dumps permit swift analysis of entered subprograms, changed data, and modified instructions. Large areas of instructions or data which have remained unchanged need not be considered.

NORMAL AND ABNORMAL PROGRAM TERMINATION

To obtain a dump at normal program termination, DMP cards may be placed anywhere after the control card that executes the programs. To obtain a dump at abnormal program termination, DMP control cards must follow the EXIT control card.

DMP EXAMPLES

The following DMP examples are interpreted as a standard dump request except they can be labeled (with or without a change dump) if appropriate DEBUG cards are present.

```
DMP(1000)

DMP(100,200)
```

The following dump requests must be for labeled format, produced by inserting a DEBUG control card prior to the card that loads or executes the programs to be dumped.

Request format	Dump from	Dump to
DMP(CPC,IO)	Start of program CPC	End of program IO*
DMP(COPYL)	Reference address (RA)	Beginning of COPYL
DMP(100,COPYL)	RA+100(octal)	End of COPYL*
DMP(COPYL,2000)	Beginning of COPYL	RA+2000(octal)
DMP(COPYL,COPYL)	Start of program COPYL	End of program COPYL*
DMP(,RED)	Reference Address (RA+0)	Start of labeled common block RED*
DMP(,RED,,WHITE)	Start of labeled common block RED	End of labeled common block WHITE*
DMP(,RED,,RED)	Start of labeled common block RED	End of labeled common block RED*
DMP(100,,RED)	RA+100(octal)	End of labeled common block RED*
DMP(IDA,,RED)	Start of program IDA**	End of labeled common block RED*
DMP(,WHITE,ELLA)	Start of labeled common block WHITE**	End of program ELLA*
DMP(,WHITE,70000)	Start of labeled common block WHITE	RA+70000(octal)

*One word beyond dump end is dumped also.
**Because of manner in which labeled common blocks are loaded, care must be taken when program names and labeled common block names are specified as the first word and the last word address.

SAMPLE DECK STRUCTURE

Normal execution to produce a labeled dump if job terminates abnormally:

```
job card
DEBUG.                  DEBUG card remains in force throughout job.
COPYBR(INPUT,LGO)
COMPASS.
LGO.
EXIT.                   If program execution terminates abnormally, control cards
                        following this card will be executed. Otherwise, the job
                        will proceed to the end of the control card stream.
DMP(6000)
DMP(PA,PA)              Dump locations occupied by program PA. PA must have been
                        loaded for this card to be accepted.
```

7/8/9

(Binary deck of previously assembled program)
7/8/9
(COMPASS source decks including program called PA)
6/7/8/9

If COMPASS terminates abnormally, the dump produces labels reflecting programs loaded for COPYBR. COMPASS, an overlay program, has no loader tables to update the DEBUG file. Sample dump outputs are given on pages 7-9 and 7-10.

SNAP

The SNAP (or snapshot) dump capability provides selective area printouts upon execution of specified instructions. Printing frequency is established by parameters. The dump format is variable.

When SNAP cards are encountered, the system prepares SNAP tables, which are located in front of the loader tables and extend toward the reference address (RA). During subsequent loading, snap parameters are inserted which apply to newly loaded programs. The SNAP control card may specify an entry point to a user supplied routine which is entered before the snap output is written.

Prior to execution, the word of instructions at a SNAP initial address (IA) is trapped and its contents are replaced by a return jump to the SNAP routine. The replaced instruction word is saved in the SNAP tables for execution after the snap is taken. When the trapped address (IA) is encountered during execution, the SNAP routine stores all registers. Upon return from the SNAP routine, the snap dump is written on the system-assigned local file SNACE or on an alternate file if an FET address is specified by the user. To obtain listings, the dump written on file SNACE must be rewound and copied onto file OUTPUT.

Following the dump, the saved instructions are executed before control is passed to the trapped location + 1. If an alternate address is placed in the communications area through the use of the user's entry point parameter specification, SNAP will return to that location after executing the saved instructions.

SNAP cannot be used for programs loaded entirely from the system library or for absolute programs.

SNAP Control Card

The following SCOPE control card initiates SNAP:

SNAP(ID=i,IA=ia,FWA=fwa,LWA=lwa,INT=n, F=f,F1=n,F2=n,F3=n,UR=params)

Parameters may appear in any order except as noted below. If more than one SNAP card is used within a job, the cards must appear contiguously within the job's control card stream. SNAP cards must appear before the control card which loads only or loads and executes the program.

PARAMETER SPECIFICATIONS

In the descriptions of parameter formats, the slashes indicate choices to be made by the programmer.

SNAP Identifier (ID): An optional identifier which is printed with the dump. If continuation cards are used, the identifier must appear somewhere on the first card and as the first parameter on all continuation cards.

EXCHANGE JUMP PACKAGE DUMP

```
DMPX.
 P   004645  A0  000111  B0  000000
 RA  125300  A1  000001  B1  000006     C(A1)=  0102  2400  0000  0000  0000      C(B1)=  0000  0000  0000  0000  0000
 FL  050000  A2  000006  B2  002122     C(A2)=  0000  0000  0000  0000  0000      C(B2)=  1725  2420  2524  0074  0035
 EM  070000  A3  002124  B3  001625     C(A3)=  0000  0000  0000  0000  2317      C(B3)=  5555  5555  5555  2324  1720
 RE  000000  A4  002125  B4  000315     C(A4)=  0000  0000  0000  0000  2317      C(B4)=  5555  5555  5555  5555  5555
 FE  000000  A5  002122  B5  000001     C(A5)=  1725  2420  2524  0074  0035      C(B5)=  0102  2400  0000  0000  0000
 MA  002064  A6  000001  B6  777776     C(A6)=  0102  2400  0000  0000  0000      C(B6)=
             A7  000001  B7  000001     C(A7)=  0102  2400  0000  0000  0000      C(B7)=  0102  2400  0000  0000  0000
 X0  7777  7777  7777  7777  0000
 X1  0000  0000  0000  0000  0000
 X2  0000  0000  0000  0000  0000
 X3  0000  0000  0000  0000  2317
 X4  0000  0000  0000  0000  5105
 X5  1523  0700  0000  0000  0000
 X6  1523  0700  0000  0000  5105
 X7  0102  2400  0000  0000  0000
```

The following information is stored in the 16-word exchange jump package.

P	Program register contents
RA	Central memory address of beginning of user field length
FL	Central memory field length limit
EM	Error mode
RE	ECS reference address
FE	ECS field length
MA	Monitor address applicable only to machines with monitor exchange jump instructions:
	A1-A7 Contents of A registers 1-7
	B1-B7 Contents of B registers 1-7
	X0-X7 Contents of X registers 1-7

When the exchange jump package is dumped, the following information is given also if addresses are within the field length:

C(A1)-C(A7)	Contents of addresses listed in registers A1-A7
C(B1)-C(B7)	Contents of addresses listed in registers B1-B7

```
DMP.

 LOW CORE

000000  00000   00000 00000 00000 00000        05160 40000 00000 00000        00000 00000 00000 00000        06200 00000 00000 00000
000004  00004   00000 00000 00000 00000  00006→17252 42025 24000 00120        00000 00000 00000 00004
000010  00010   00000 00000 00000 10000        00000 00000 00000 00000  00014→00000 00000 00000 60000        14071 70000 00000 00000
000016  00016   00000 00000 00000 00000  00020→04000 14013 00000 00000        14071 70000 00000 00047        02000 00000 00100 31774
000023  00023   00000 00000 00000 32022  00025→24110 62000 01000 55432        00000 00000 00000 00000  00042→55051 60455 03012 20455
000043  00043   15112 32311 16070 00000
000044  00044   55555 55555 55555 55555  00051→00000 00000 00000 00000  00053→23312 32405 30240 00000
000054  00054   00000 00000 00000 00000        00000 00000 00000 31774        00000 00000 00000 00002        00000 00000 00000 55277
000060  00060   00000 00000 00000 00003  00062→00000 00000 00000 00000        00000 00000 00000 60000
000064  00064   04152 00000 00000 00002        00000 00000 00000 11076        00052 23010 60000 00720        00000 00000 20000 54010
000070  00070   04152 05156 06205 20000        00000 00000 00000 00000  00077→00000 00000 00000 00000


 FP  (COMMON)

000000  00100   60420 53217 77777 77777


CHANGE DUMP
  LOC         ENDING VALUE                STARTING VALUE              LOC         ENDING VALUE                STARTING VALUE

   00001    05160 40000 00000 00000     00000 00000 00000 00000      00003    06200 00000 00000 00000      00000 00000 00000 0000
0
   00004    00000 00000 00000 00000     00000 00000 00000 02022      00064    04152 00000 00000 00002      14071 70000 00000 0000
0
   00070    04152 05156 06205 20000     14071 75700 00000 00000


 GEN   (PROGRAM)

000000  00720   07051 60000 00000 00102    61100 00720 61200 00766    01000 05155 00000 00000    71600 00001 46000 46000
000004  00724   51600 01022 10166 46000    52010 00553 27001 42600    26306 71207 73073 36723    54700 27001 42700 42607
000010  00730   26506 71407 42547 36645    51600 01012 62110 00242    53011 10266 27202 24202    51300 01004 44623 43700
000014  00734   54600 50760 00001 46000    52010 00077 27001 42700    42607 26206 36642 27706    24707 72110 00001 46000
000020  00740   72017 77632 54700 46000    03300 00725 61300 00773    61200 01021 67202 66100    01000 06232 07000 00720
000024  00744   51200 01005 63220 46000    61100 00554 01000 06232    61107 77776 01000 06232    61300 00776 61200 01021
000030  00750   67202 66100 46000 46000    01000 06232 07000 00720    51300 01005 63230 46000    61100 00100 01000 06232
000034  00754   61107 77776 01000 06232    61300 01001 61200 01021    67202 66100 46000 46000    01000 06232 07000 00720
000040  00760   51400 01005 63240 63242    61100 00244 01000 06232    61107 77776 01000 06232    01000 03045 07000 00720
000044  00764   51500 01006 10755 46000    01000 05250 07000 00720    00000 00000 00000 00103    00000 00000 00000 00000
000050  00770   00000 00000 00000 10000    17252 42025 24000 02022    24012 00541 00000 00001    34000 00000 00000 00000
000054  00774   51343 05643 11343 35255    00000 00000 00000 00055    35000 00000 00000 00000    51343 05643 05343 65737
000060  01000   52000 00000 00000 00055    36000 00000 00000 00000    51343 05643 04343 65737    52000 00000 00000 00055
```

```
DMPX.

P   004645  A0  000111  B0  000000
RA  125300  A1  000001  B1  000006   C(A1)=  0102  2400  0000  0000  0000    C(B1)=  0000  0000  0000  0000  0000
FL  050000  A2  000006  B2  002122   C(A2)=  0000  0000  0000  0000  0000    C(B2)=  1725  2420  2524  0074  0035
EM  070000  A3  002124  B3  001625   C(A3)=  0000  0000  0000  0000  2317    C(B3)=  5555  5555  5555  2324  1720
RE  000000  A4  002125  B4  000315   C(A4)=  0000  0000  0000  0000  2317    C(B4)=  5555  5555  5555  5555  5555
FE  000000  A5  002122  B5  000001   C(A5)=  1725  2420  2524  0074  0035    C(B5)=  0102  2400  0000  0000  0000
MA  002064  A6  000001  B6  777776   C(A6)=  0102  2400  0000  0000  0000    C(B6)=
            A7  000001  B7  000001   C(A7)=  0102  2400  0000  0000  0000    C(B7)=  0102  2400  0000  0000  0000
X0  7777  7777  7777  7777  0000
X1  0000  0000  0000  0000  0000
X2  0000  0000  0000  0000  0000
X3  0000  0000  0000  0000  2317
X4  0000  0000  0000  0000  5105
X5  1523  0700  0000  0000  0000
X6  1523  0700  0000  0000  5105
X7  0102  2400  0000  0000  0000

  00000    00000 00000 00000 00000         01022 40000 00000 00000        11162 02524 00000 00100        17252 42025 24000 02122
  00004    24012 00540 00000 00100         24012 00541 00000 02122        00000 00000 00000 00000        00000 00000 00000 00007
  00010    00000 00000 00000 00000  00012→00000 00000 00000 47777        00000 00000 00000 47651
  00014    00000 00000 00000 47631         00000 00000 00000 40713        00000 00000 00000 00000
  00020   →04000 30725 00000 00000         00000 00000 00000 00000        00000 00000 00000 00001        00000 00000 00000 00000
  00024    00000 00000 00000 00015         00000 00000 00000 40547
  00030   →00000 00000 00000 40713         00000 00000 00000 00000        00000 00000 00000 47432        00000 00000 00000 00000
  00035   →00000 00000 00000 00001         00000 00000 00000 00000        00000 00000 00000 00040
  00040    00000 00000 00000 00000  00043→00000 00000 00000 47576
  00044    00000 00000 00000 00000  00046→00000 00000 00000 47777        00000 00000 00000 47651
  00050    77777 77777 77777 77776         00000 00000 00000 00000        00000 00000 00000 00001        00000 00000 00000 00000
  00054    00000 00000 00030 00000         00000 00000 00000 00000        20001 51722 07010 70500        00000 00000 00000 00000
  00061   →00000 00000 00000 47432         00000 00000 00000 00000        14551 72255 06270 10000
  00064    14071 70000 00000 00000         00000 00000 00000 12156        00042 15210 00000 00100        00000 00000 20000 44010
  00070    00000 00000 00000 00000
  00100   →11162 02524 00000 00021

  04545    03310 05010 63210 46000         07210 04547 76110 46000        52510 05132 43014 20010
  04550    20510 11405 63140 20510         11405 63240 20514 20004        11405 63340 76770 20514        20730 51300 05076 56000
  04554    51400 05077 76000 46000         05100 04560 04200 04557        11405 63140 05300 04563        11553 04000 04565 46000
  04560    05120 04562 04300 04557         11545 56070 71600 10000        36557 04000 04565 46000        04270 04564 04120 04557
  04564    56070 71600 10000 46000         37556 11050 72707 70000        03270 04750 76770 36557        10655 20624 54650 46000
  04570    03020 04527 64100 53010         04100 04627 63320 46000        51100 05113 43060 10611        51600 05545 56230 46000
  04574    15320 10622 54227 54667         03130 04574 51100 05114        74300 10711 54767 46000        61100 00004 01000 05013
  04600    51600 05071 20630 54117         12616 51500 05123 46000        51200 04527 54677 10655        51400 05066 21236 63340
  04604    52427 77776 54667 10744         51700 05075 46000 46000        43014 11504 43752 20506        63150 05170 04625 54357
  04610    53140 11271 20252 12623         54667 63210 20036 11304        03030 04671 51200 05112        61100 00004 21322 10722
  04614    54767 01000 05013 46000         54677 05200 04616 64217        61500 05544 65165 64467        71217 77553 03220 04722
  04620    56220 20206 72127 77773         67337 07370 04625 46000        03110 04625 20230 46000        52427 77776 51500 05107
  04624    10655 56640 04000 04606         61500 05544 65165 66200        05100 04722 46000 46000        50200 05132 73020 20250
  04630    03100 04650 03220 04650         51100 05101 03010 04634        76600 51600 05101 46000        04000 04651 46000 46000
  04634    01000 04763 46000 46000         01000 05030 46000 46000        51100 05071 71700 02255        20160 20760 71501 52307
  04640    12671 51600 05106 75467         20552 12645 46000 46000        56170 03110 04642 56670        71700 10224 20752 46000
  04644    56170 03110 04644 56770         00000 00000 46000 46000        11770 13774 03070 04542        04000 04537 46000 46000
  04650    10622 20201 03320 04742         20223 51500 05067 53050        03000 04527 51300 04527        21336 52437 77776 53340
  04654    63330 04300 04655 54537         53050 61100 05072 46000        51500 04664 51300 04525        53130 73307 12753 46000
  04660    50757 77776 03360 04665         05300 04662 64337 46000        56430 10644 53600 46000        00000 00000 00000 00000
  04664    46000 46000 04000 00000         71600 04670 71500 00400        20652 12756 20760 53700        04000 04663 46000 46000
  04670    61700 00001 04000 04631         05200 04672 64217 46000        53340 63430 63540 77345        61100 00006 76600 46000
  04674    71200 00007 20352 46000         20303 11432 72540 00033        20606 36665 67117 46000        05100 04675 51200 05111
  04700    10722 20614 71300 02355         20360 12636 54767 54677        04000 04616 46000 46000        26312 63730 71600 00014
  04704    51100 05065 71600 00014         51600 05773 01000 06060        76000 61700 00001 46000        61700 00001 04000 04651
```

Identifier format:

ID=i   i is 1-7 alphanumeric characters

Example:

```
SNAP,p1,p2,ID=SNAP1,p3.         first SNAP card

SNAP,ID=SNAP1,p4,...,pn.        continuation SNAP card
```

Initial Address (IA): The address where a trap (a return jump to SNAP) is placed. This parameter is required. Restrictions on the specification of this parameter follow.

IA formats:

$$\left.\begin{array}{l}\text{IA}=\text{e/e+n/a}\\ \text{IA1}=\text{e-n}\end{array}\right\}\quad\text{for entry points}$$

$$\left.\begin{array}{l}\text{IAC}=\text{c/c+n}\\ \text{IAC1}=\text{c-n}\end{array}\right\}\quad\text{for common blocks}$$

e	Entry point name
c	Labeled common block name
n	Octal integer not greater than 777777
a	Absolute address (relative to RA) expressed as six octal digits

First Word Address (FWA): The first word address of the area to be dumped. No restrictions are placed on FWA with respect to IA; it may be the same or greater or less than the location specified by IA. This parameter is optional.

FWA formats:

$$\left.\begin{array}{l}\text{FWA}=\text{e/e+n/n/a}\\ \text{FWA1}=\text{e-n/n}\end{array}\right\}\quad\text{for entry points}$$

$$\left.\begin{array}{l}\text{FWAC}=\text{c/c+n/n}\\ \text{FWAC1}=\text{c-n/n}\end{array}\right\}\quad\text{for labeled common blocks}$$

FWAB=b   for blank common blocks

e	Entry point name
c	Labeled common block name
b	Octal number denoting the bth location in blank common block
n	Octal integers not greater than 777777. If e or c has appeared as a previous parameter on the SNAP card, it is assumed that n is to be added to or subtracted from e or c. Thus, the address will be e+n, e-n, c+n, or c-n. If e or c has not yet appeared on this card, the number specified is assumed to be a (see following).
a	Absolute address (relative to RA) expressed as six octal digits.

Last Word Address (LWA): The last word of the area to be dumped. LWA must be specified if FWA is specified.

LWA formats:

LWA=e/e+n/n/a
LWA1=e-n/n
	for entry points

LWAC=c/c+n/n
LWAC1=c-n/n
	for labeled common blocks

LWAB=b	for blank common blocks

Interval (INT): Interval between words dumped. This parameter is optional.

INT format:

INT=n	n is a positive octal integer; if not specified, 1 is assumed. For a D dump, the value of n is doubled.

Dump format (F): Format of output dump. If FWA and LWA are present and F is not specified, octal(O) format is assumed; also a register (R) dump is taken. If FWA and LWA are not present and F is not specified, only a register dump is taken.

Dump Format:

F=Rx or F=xR or F=x

One of the following must be specified for x.

O	Octal dump
M	Octal dump with mnemonic operation code
I	Integer dump
S	Single precision floating point dump
F	I format if exponent = 0; otherwise, S format
D	Double precision floating point dump (two words)
C	Display code dump
R	Denotes register dump. It is optional and may suffix or prefix any of the above designators.

Frequency (Fi): An optional set of parameters which permits control of the SNAP output volume when an IA is within the range of an iteration. n is a positive octal integer which cannot be zero. If not specified, 1 is assumed for an omitted parameter.

Format:

F1=n	When the program is executed, a counter is set to 0 and incremented by 1 each time the IA address is encountered. When the counter value reaches n, the initial SNAP output will be taken.
F2=n	When the IA counter exceeds n, no further SNAP output will be taken.
F3=n	The increment value n determines when the next SNAP output will be taken.

Example:

```
F1=10,F2=30,F3=5,IA=ALPHA
```

In an iterative operation, a SNAP will be taken when address ALPHA is encountered the 10th time, and it will be taken every 5th time after that until the counter exceeds 30. Therefore, SNAP output will be produced for the 10th, 15th, 20th, 25th and 30th time address ALPHA is encountered.

User Entry Point (UR): Optional; but if specified, it must be last on the SNAP card.

Format:

UR=p,r1,...rn

p	User's entry point to be called before SNAP dump is taken.
r	Parameters passed to user's routine: Alphanumeric string, 1-10 characters, terminated by a zero byte. If the string contains 9 or 10 characters, an extra word is required. Decimal integer, converted to binary, stored right-justified. The parameter list is terminated by -0 (word filled with octal sevens) which is used optionally by the user's routine. It has no meaning for SNAP.

SNAP enters the user entry point in the following manner:

L	RJ	P	(user entry address)
L+1		TADDR	(FWA of loader SNAP tables) User parameters begin at FWA-10 (octal) and extend toward the reference address.
L+2		RB0	(FWA of register storage area)
L+3			The user program must increment the return address by two so that return to SNAP will be at L+3.

Registers are stored one per word in the first 24 words of the register storage are a as follows:

B0-B7, A0-A7, X0-X7

RB0+24 has the following format:

Bits	
59	No-dump flag; if bit 59 is set, SNAP output is suppressed. This bit is cleared on entry to the user routine.
18-58	Not used.
0-17	FET address can be inserted to designate an alternate file for SNAP output. If the user supplies an FET, the buffer defined by that FET must be 27 words greater than the size of one PRU for the receiving device.

RB0+25 has the following format:

Bits	
18-59	Not used.
0-17	Address to which SNAP returns (address+1 of the trapped instruction). Return address can be changed by replacing the address in these bits.

RESTRICTIONS AND CAUTIONS ON SNAP USE

Only one FWA, LWA range may be specified on a SNAP card. More than one SNAP card may be used within the same job, but the IA specified for each must be for a different location.

The instruction at IA must not be modified during program execution (a subroutine entry point called by a return jump is modified).

A subroutine called by a return jump instruction at IA must return to the word following the return jump. The following sequence of instructions will cause incorrect results:

```
      RJ     subroutine
   +  JP     *+n+1
      VFD    60/parameter1
      VFD    60/parameter2
      .
      .
      .
      VFD    60/parameter n
```

A subroutine called from IA will result in an infinite loop.

SNAP IN OVERLAY OR SEGMENT MODE

SNAP triggers are inserted as each segment is loaded. The SNAP routine is loaded with the SEGZERO segment or with the (0,0) level overlay. The DEBUG(S) card must be included in the loader input stream when the overlay file is prepared; it must appear immediately before the initiation of a segment load.

SNAP CONTROL CARD EXAMPLES

```
SNAP(IA=TAG)
```

The first time location TAG is encountered, a dump is produced of the contents of all registers as they appeared before execution of TAG.

```
SNAP(IA=HOOK,F=M,ID=SYM,FWA=C,LWA=C+30,UR=IN,1,A,2)
```

When location HOOK is executed, control passes to a user subroutine (entry point = IN). Parameters 1, A, and 2 are passed to the user subroutine. If the user routine returns control to SNAP, and if the no-dump flag is not set (in RB0 + 24, bit 59), a mnemonic dump format (F = M) is taken (labeled SYM) of locations C through C + 30.

```
SNAP(ID=AX,IA=L,FWA=B,LWA+B+150,F1=10,F2=35,F3=2,F=O)
```

Produces a dump in octal format labeled AX and consisting of all locations from B to B + 150. The dump is triggered and the 8th time (octal 10) the instruction at location L is encountered, and is taken every 2nd time thereafter through the 34th time.

```
SNAP(ID=AX,IA=L,FWA=B,LWA=B+150,INT=5,F1=10,F2=35,F3=2,F=O)
```

Same as above, except that the dump is taken when location L has been encountered in intervals of five times.

EXAMPLES OF DECK SETUP STRUCTURES

SNAP run with SNACE output going to tape; tape to be listed by a second job:

```
job card
REQUEST,SNACE,*MT.
REWIND(SNACE)
SNAP(params)              SNAP card must be placed before program load or load and
                          execute control card.
INPUT.
COPYBF(X,SNACE)           These COPY cards write an end-of-file on the SNACE tape.
                          Normally, this method of running SNAP and TRACE is used when
                          output to SNACE is extensive.
COPYBF(X,SNACE)
7/8/9
(object deck)
6/7/8/9
job card                  This job is used to list the extensive SNACE output from the
                          preceding job.
REQUEST,SNACE,MT.
REWIND(SNACE)
COPYCF(SNACE,OUTPUT)
7/8/9
6/7/8/9
```

SNAP OUTPUT EXAMPLES

SNAP output examples are given on pages 7-16, 7-17 and 7-18.

TRACE

The tracing function provides a capability for analysis of program execution. Instructions based on storage references, operand references, register usage, and branch instructions are analyzed. Output is written on a local file named SNACE. If trace output is to be listed, SNACE must be rewound and copied to the standard output file OUTPUT. Trace output always includes a dump of the contents of the P register, all operand registers involved, and the result register.

Each instruction within the designated range (IA through LA) is scanned for triggers (conditions which can cause trace output) as established by the TRACE control card parameters. A trap, in the form of a return jump instruction to TRACE, is placed at each instruction containing a trigger. As each trap is encountered during execution, the designated instruction is executed in the area used by TRACE; and the specified output is written on SNACE. TRACE continues until the parameters are satisfied and as long as the program remains inside the designated range. Tracing ranges can overlap and multiple outputs can be triggered.

TRACE may be used with all system loading schemes, except programs loaded entirely from the system library. Overlay/segment mode has special requirements. When TRACE cards are encountered, the system prepares TRACE tables to be referenced during subsequent loading. Calls for SNAP features cannot be traced because they are not considered part of the user program.

OUTPUT PRODUCED FOR: SNAP(ID=SNAP,FWA=ODDR+11,IA=ODDR+4,F=M,F2=2)
RELATIVE ADDRESS OF ODDR=5514(OCTAL)

```
        SNAP 1.0   SNAP            P = 005520                                02/11/70    PAGE 001

005521       SA6   A1+B0          LX6   73                 PL    X6,005523

005522       SB1   B1+000001      NOP                      NOP
005523       SB3   B3+000001      LX6   01                 NOP

005524       NE    B3,B2,005520   SX7   B1+B0              NOP
005525       SA7   B0+011450      EQ    B0,B0,005514
```

```
        SNAP 1.0   SNAP            P = 005520                                02/11/70    PAGE 002

005521       SA6   A1+B0          LX6   73                 PL    X6,005523

005522       SB1   B1+000001      NOP                      NOP
005523       SB3   B3+000001      LX6   01                 NOP

005524       NE    B3,B2,005520   SX7   B1+B0              NOP
005525       SA7   B0+011450      EQ    B0,B0,005514
```

SNAP(ID=OCTAL,IA=ODDR+4,F1=25,F2=26,INT=3,FWAB=1,LWAB=35,F=IR)

```
        SNAP 1.0  OCTAL          P = 005520                                  02/10/70    PAGE 002

        A0  040000                                 B0  000000   X0  77777777777777000000
        A1  011475  C(A1)= 00000000000000000037    B1  000013   X1  00000000000000000055
        A2  003454  C(A2)= 00000000000000000034    B2  000043   X2  00000000000000003472
        A3  003455  C(A3)= 00000000000000000000    B3  000037   X3  00000000000000002022
        A4  003457  C(A4)= 17252420252400002022    B4  000000   X4  17252420252400002022
        A5  032410  C(A5)= 76000000000011000000    B5  005535   X5  00000000000000010000
        A6  011475  C(A6)= 00000000000000000037    B6  000000   X6  00000000000000000037
        A7  003512  C(A7)= 00000000000000000000    B7  000001   X7  00000000000000000000

011451       11                        14                           17                       20
011465       23                        26                           29                       45
011501       45                        45
```

SNAP(ID=FLOAT,IA=POINT1,FWAC=FP,LWAC=FP+143,F=S)

```
        SNAP 1.0  FLOAT              P = 006404                                                                02/09/70    PAGE 001

003436     -1.49990000000000E+04           -1.49920000000000E+04           -1.49730000000000E+04           -1.49360000000000E+04

003442     -1.48750000000000E+04           -1.47840000000000E+04           -1.46570000000000E+04           -1.44880000000000E+04
003446     -1.42710000000000E+04           -1.40000000000000E+04           -1.36690000000000E+04           -1.32720000000000E+04
003452     -1.28030000000000E+04           -1.22560000000000E+04           -1.16250000000000E+04           -1.09040000000000E+04
003456     -1.00870000000000E+04           -9.16800000000000E+03           -8.14100000000000E+03           -7.00000000000000E+03
003462     -5.73900000000000E+03           -4.35200000000000E+03           -2.83300000000000E+03           -1.17600000000000E+03
003466      6.25000000000000E+02            2.57600000000000E+03            4.68300000000000E+03            6.95200000000000E+03
003472      9.38900000000000E+03            1.20000000000000E+04            1.47910000000000E+04            1.77680000000000E+04
003476      2.09370000000000E+04            2.43040000000000E+04            2.78750000000000E+04            3.16560000000000E+04
003502      3.56530000000000E+04            3.98720000000000E+04            4.43190000000000E+04            4.90000000000000E+04
003506      5.39210000000000E+04            5.90880000000000E+04            6.45070000000000E+04            7.01840000000000E+04
003512      7.61250000000000E+04            8.23360000000000E+04            8.88230000000000E+04            9.55920000000000E+04
003516      1.02649000000000E+05            1.10000000000000E+05            1.17651000000000E+05            1.25608000000000E+05
003522      1.33877000000000E+05            1.42464000000000E+05            1.51375000000000E+05            1.60616000000000E+05
003526      1.70193000000000E+05            1.80112000000000E+05            1.90379000000000E+05            2.01000000000000E+05
003532      2.11981000000000E+05            2.23328000000000E+05            2.35047000000000E+05            2.47144000000000E+05
003536      2.59625000000000E+05            2.72496000000000E+05            2.85763000000000E+05            2.99432000000000E+05
003542      3.13509000000000E+05            3.28000000000000E+05            3.42911000000000E+05            3.58248000000000E+05
003546      3.74017000000000E+05            3.90224000000000E+05            4.06875000000000E+05            4.23976000000000E+05
003552      4.41533000000000E+05            4.59552000000000E+05            4.78039000000000E+05            4.97000000000000E+05
003556      5.16441000000000E+05            5.36368000000000E+05            5.56787000000000E+05            5.77704000000000E+05
003562      5.99125000000000E+05            6.21056000000000E+05            6.43503000000000E+05            6.66472000000000E+05
003566      6.89969000000000E+05            7.14000000000000E+05            7.38571000000000E+05            7.63688000000000E+05
003572      7.89357000000000E+05            8.15584000000000E+05            8.42375000000000E+05            8.69736000000000E+05
003576      8.97673000000000E+05            9.26192000000000E+05            9.55299000000000E+05            9.85000000000000E+05
```

SNAP(ID=DOUBLE,FWAC=DP,LWAC=DP+143,F=D,IA=POINT2)

```
        SNAP 1.0  DOUBLE            P = 006405                                                            02/09/70    PAGE 002

003602     -2.1427142857142753200605511665E+03     -2.1417142857142753200605511665E+03     -2.1390000000000000000000000000E+03

003610     -2.1337142857142753200605511665E+03     -2.1250000000000000000000000000E+03     -2.1120000000000000000000000000E+03
003616     -2.0938571428571303840726613998E+03     -2.0697142857142753200605511665E+03     -2.0387142857142825960181653500E+03
003624     -2.0000000000000000000000000000E+03     -1.9527142857142825960181653500E+03     -1.8960000000000000000000000000E+03
003632     -1.8290000000000000000000000000E+03     -1.7508571428571376600302755833E+03     -1.6607142857142825960181653500E+03
003640     -1.5577142857142825960181653500E+03     -1.4410000000000000000000000000E+03     -1.3097142857142825960181653500E+03
003646     -1.1630000000000000000000000000E+03     -1.0000000000000000000000000000E+03     -8.1985714285714129800908267498E+02
003654     -6.2171428571428259601816534996E+02     -4.0471428571428441500756889582E+02     -1.6800000000000000000000000000E+02
003662      8.9285714285714220750378444791E+01      3.6800000000000000000000000000E+02      6.6900000000000000000000000000E+02
003670      9.9314285714285506401211023331E+02      1.3412857142857101280242204666E+03      1.7142857142857101280242204666E+03
003676      2.1130000000000000000000000000E+03      2.5382857142857101280242204666E+03      2.9910000000000000000000000000E+03
003704      3.4720000000000000000000000000E+03      3.9821428571428550640121102333E+03      4.5222857142857101280242204666E+03
003712      5.0932857142857101280242204666E+03      5.6960000000000000000000000000E+03      6.3312857142857101280242204666E+03
003720      7.0000000000000000000000000000E+03      7.7030000000000000000000000000E+03      8.4411428571428405120968818665E+03
003726      9.2152857142856810241937637329E+03      1.0026285714285681024193763733E+04      1.0875000000000000000000000000E+04
003734      1.1762285714285681024193763733E+04      1.2689000000000000000000000000E+04      1.3656000000000000000000000000E+04
003742      1.4664142857142840512096881866E+04      1.5714285714285681024193763733E+04
```

SNAP(ID=INTEGER,FWAC=IN,LWAC=IN+143,F=I,IAPOINT=3)

```
         SNAP 1.0  INTEGER          P = 006406                                                          02/09/70   PAGE 003
004112        1152921503533109248           1152921503533109248           1152921503533109248           1152921503533109248
004116        1152921503533109248           1152921503533109248           1152921503533109248           1152921503533109248
004122        1152921503533109248           1152921503533109248           1152921503533109248           1152921503533109248
004126        1152921503533109248           1152921503533109248           1152921503533109248           1152921503533109248
004132        1152921503533109248           1152921503533109248           1152921503533109248           1152921503533109248
004136        1152921503533109248           1152921503533109248           1152921503533109248           1152921503533109248
004142        1152921503533109248           1152921503533109248           1152921503533109248           1152921503533109248
004146        1152921503533109248           1152921503533109248           1152921503533109248           1152921503533109248
004152        1152921503533109248           1152921503533109248           1152921503533109248           1152921503533109248
004156        1152921503533109248           1152921503533109248           1152921503533109248           1152921503533109248
004162        1152921503533109248           1152921503533109248           1152921503533109248           1152921503533109248
004166        1152921503533109248           1152921503533109248           1152921503533109248           1152921503533109248
004172        1152921503533109248           0                             101                           204
004176        309                           416                           525                           636
004202        749                           864                           981                           1100
004206        1221                          1344                          1469                          1596
004212        1725                          1856                          1989                          2124
004216        2261                          2400                          2541                          2684
004222        2829                          2976                          3125                          3276
004226        3429                          3584                          3741                          3900
004232        4061                          4224                          4389                          4556
004236        4725                          4896                          5069                          5244
004242        5421                          5600                          5781                          5964
004246        6149                          6336                          6525                          6716
004252        6909                          7104                          7301                          7500
```

TRACE CONTROL CARD

The following SCOPE control card initiates TRACE:

TRACE,p1,p2,...,pn.

TRACE cards must appear immediately before the control card which loads or loads and executes a program. If more than one TRACE card is used, they must be placed contiguously within the job's control card stream. The order of the parameters is not significant except as noted. Parameters must be specified unless indicated as optional. The (/) used in the parameter specification denotes that a choice is to be made by the programmer.

PARAMETER SPECIFICATIONS

Identification: Optional; but when specified, the identifier is printed on TRACE output. If continuation cards are used, the ID must appear somewhere on the first card and as the first parameter on all continuation cards. For example:

```
TRACE,p1,ID=TRACE,p3,p4.            First TRACE card

TRACE,ID=TRACE,p5,...,pn.           Continuation TRACE card
```

Identification format:

ID=i	i is 1 to 7 alphanumeric characters.

Initial Address (IA): Designates the beginning of the tracing range. Restrictions on the specification of this parameter are given at the end of this section.

IA formats:

$$\left.\begin{array}{l}\text{IA}=\text{e/e+n}\\ \text{IA1}=\text{e-n}\end{array}\right\}\ \text{for entry points}$$

$$\left.\begin{array}{l}\text{IAC}=\text{c/c+n}\\ \text{IAC1}=\text{c-n}\end{array}\right\}\ \text{for labeled common blocks}$$

e	Entry point name
c	Labeled common block name
n	Octal integer no greater than 777777

Last Address (LA): Designates the end of the tracing range. LA must be greater than IA. Further restrictions applying to the specification of the parameter are given at the end of this section.

LA formats:

$$\left.\begin{array}{l}\text{LA}=\text{e/e+n}\\ \text{LA1}=\text{e-n}\end{array}\right\}\ \text{for entry points}$$

$$\left.\begin{array}{l}\text{LAC}=\text{c/c+n}\\ \text{LAC1}=\text{c-n}\end{array}\right\}\ \text{for labeled common blocks}$$

e	Entry point name
c	Labeled common block name
n	Octal integer not greater than 777777

Three trigger specifications are described below; at least one must appear on a TRACE card.

Register Trigger (TR): Each instruction is examined to determine whether or not the specified trigger is used as a result register. The P register measures satisfactory completion of conditional jump. It must be placed before other triggers on the TRACE card; otherwise, traps are set for previously set traps.

TR format:

```
TR=P/An/Bn/Xn
```

n	Register number 0-7

Masking Trigger (TM): A mask (Boolean AND) of each instruction in the range is compared with all match keys for that mask. If equality is found, the instruction is used as a trigger.

TM format:

```
TM=m,k1,k2,...,kn
```

m	Mask of 5 or 10 octal digits
k	Match key associated with mask m

Location Trigger (TL): Each instruction making an A register reference to the specified location is used as a trigger. Only the unmodified 18-bit address of that instruction is used as a trigger. The traps for TRACE are planted before execution so TRACE does not know the value of the registers at that time. In an instruction such as SA2 COMON+B2, where COMON is the trigger, an A-register reference to COMON is made, even through the operand may be modified during execution. Restrictions applying to this parameter specification are given at the end of this section.

TL formats:

```
TL=e/e+n  }
TL1=e-n   }     For entry points

TLC=c/c+n }
TLC1=c-n  }     For labeled common blocks

TLB=b           For blank common blocks
```

e	Entry point name
c	Labeled common block name
n	Octal integer not greater than 777777
b	Octal number representing the bth location in blank common block

Register Dump: Optional; but when specified, an octal dump of all A, B, and X registers is included in the output. It must immediately follow a trigger: TR, TM, TL or any form of TM or TL on a TRACE card.

Register dump format:

```
RD
```

TRACE output control includes the optional specification of frequency, and output trigger specification.

Output Frequency: TRACE output is controlled by a frequency counter (FC) which is initialized to one and a flag (FT). When the instruction at the beginning of the tracing range (IA) is encountered, the flag FT is set. When the instruction at the end of the tracing range (LA) is encountered, the flag FT is cleared and FC is incremented.

Output is generated only if all of the following conditions are true:

FT is set,
$F1 \leq FC \leq F2$, and
(FC-F1)/F3 has a remainder of zero.

If a trigger occurs outside the range $IA \leq P \leq LA$, even though the above conditions are met, no output is generated.

Frequency format:

F1=n,F2=n,F3=n

F1=n	When the program is executed, a counter is set to 0 and incremented by 1 each time the IA address is encountered. When the counter value reaches n, the initial SNAP output will be taken.
F2=n	When the IA counter exceeds n, no further SNAP output will be taken.
F3=n	The increment value n determines when the next SNAP output will be taken.

Storage location (OL): When a trigger is encountered, i words beginning with the specified location are written in octal format on SNACE.

OL format:

$$\left.\begin{array}{l}\text{OL=e,i/e+n,i}\\\text{OL1=e-n,i}\end{array}\right\}\quad\text{for entry points}$$

$$\left.\begin{array}{l}\text{OLC1=c,i/c+n,i}\\\text{OLC1=c-n,i}\end{array}\right\}\quad\text{for labeled common blocks}$$

OLB=b,i   for blank common blocks

i	Octal number of words (less than 100) to be written out, i must be specified.
e	Entry point name
n	Octal integer not greater than 777777
c	Labeled common block name
b	Octal number representing the bth location in blank common block.

Register Designation (OR): When a trigger is encountered, i words beginning at the location specified in the designated register are written in octal format on SNACE.

OR format:

OR=r,i

r	Register designator: An/Bn/Xn and n is the register number 1-7.
i	Number of words to be written out and is an octal integer less than 100 (octal). i must be specified.

RESTRICTIONS AND CAUTIONS ON TRACE USE

Instructions that are program modified or not executable cannot be traced. This restriction applies to IA, LA (and all their related formats), and all words between IA and LA.

Caution must be used if macro expansion, data, or parameter lists are contained within the tracing range. If a trigger is detected within any constants, a return jump to TRACE is placed in these locations, thus modifying the constant while actively tracing within the range.

A return jump which does not return to L + 1 cannot be traced.

If frequency parameter F1 is specified, F2 must be greater than or equal to F1.

The same location cannot be specified as the IA (and all formats of IA) on two or more TRACE control cards within the same job.

TRACE IN OVERLAY OR SEGMENT MODE

In overlay or segment mode, the DEBUG(T) card must be present when the overlay file is generated. As the TRACE routine is loaded with SEGZERO or with the (0,0) level overlay, TRACE cards must appear just prior to the program call card which causes loading of the (0,0) level overlay or the SEGZERO segment.

When TRACE cards are encountered, the system prepares trace tables to be referenced during subsequent execution. The loader tables for overlays are read from the overlay file. As each overlay is loaded, applicable TRACE controls are established. Similarly, segment loading causes TRACE traps to be inserted.

TRACE EXAMPLES

```
TRACE,ID=Q,IA=S,LA=E,F1=3,F2=7,F3=2,TM=00700,00600,OR=B4,7.
```

ID=Q	range identifier	

IA=S
LA=E
	range limits	

TM=00700,00600	trap trigger	00700=octal mask 00600=match key When the third digit of any instruction is six, the designated output (OR=B4,7) occurs if the frequency requirements are met.
F1=3,F2=7,F3=2	frequency requirements	Output is not to begin until the third time through the range. It is to be repeated each second time thereafter through the seventh time the trap is encountered.
OR=B4,7	output trigger	Output consists of seven words of data starting at the address in register B4.

TRACE SAMPLE DECK STRUCTURE

TRACE RUN:

```
 job card
COMPASS.
TRACE(params)


LGO.
REWIND(SNACE)
COPYCF(SNACE,OUTPUT)
7/8/9
(COMPASS source deck)
6/7/8/9
```

TRACE loader cards appear immediately before program call card that initiates the run.

TRACE OUTPUT EXAMPLES:

TRACE output examples are given on pages 7-24 and 7-25.

SAMPLE DECKS FOR COMBINED DEBUGGING AIDS

COMBINED TRACE/SNAP RUN:

```
 job card
COMPASS.
COPYBR(INPUT,LGO)
TRACE(params)
 .
 .
 .
TRACE(params)
SNAP(params)
 .
 .
 .
SNAP(params)
LOAD(INPUT)
LGO.
REWIND(SNACE)
COPYCF(SNACE,OUTPUT)
7/8/9
(COMPASS source deck)
7/8/9
(Binary deck from previously assembled
program)
6/7/8/9
```

TRACE and SNAP cards appear immediately before loader card that initiates run execution. Multiple SNAP and TRACE cards must be placed contiguously.

```
                                                    TRACE                    PAGE 001

TRAP FOR PMONE   AT  004655

OPERAND REGISTERS,      X2 =  00000000000000000001
   X2 =  00000000000000000001

RESULT REGISTER IS      X7 =  00000000000000000000

   B0   = 000000
   B1   = 000001
   B2   = 004654
   B3   = 000100
   B4   = 004654
   B5   = 000000
   B6   = 042356
   B7   = 004654
   A0   = 050000
   A1   = 004620
   A2   = 004620
   A3   = 042353
   A4   = 004424
   A5   = 004352
   A6   = 004653
   A7   = 004657
   X0   = 77000000777777777777
   X1   = 50600073370000000000
   X2   = 00000000000000000001
   X3   = 77777777777777777776
   X4   = 00042300000000000000
   X5   = 00000000000000004654
   X6   = 00000000000000000000
   X7   = 00000000000000000000
```

```
                                                  TRACE                     PAGE 002
TRAP FOR FMONE    AT  004656

OPERAND REGISTERS,      X3 =  77777777777777777776
   X3 =  77777777777777777776

RESULT REGISTER IS      X7 =  00000000000000000000

   B0   = 000000
   B1   = 000001
   B2   = 004654
   B3   = 000100
   B4   = 004654
   B5   = 000000
   B6   = 042356
   B7   - 004654
   A0   = 050000
   A1   = 004620
   A2   = 004620
   A3   = 042353
   A4   = 004424
   A5   = 004352
   A6   = 004662
   A7   = 004663
   X0   = 77000000777777777777
   X1   = 50600073370000000000
   X2   = 00000000000000000001
   X3   = 77777777777777777776
   X4   = 00042300000000000000
   X5   = 00000000000000004654
   X6   = 00000000000000000000
   X7   = 00000000000000000000
```

OVERLAY PREPARATION

Execution run using both SNAP and labeled dumps:

```
job card
REQUEST,FILE,MT. LOADER INPUT FILE
REQUEST,OVFILE,MT. OVERLAY FILE WRITTEN BY LOADER
DEBUG.
DEBUG(S)
LOAD(FILE)
NOGO.
SNAP(params)
OVFILE.
UNLOAD(OVFILE)
DMP(1000)
REWIND(SNACE)
COPYCF(SNACE,OUTPUT)
EXIT.
DMP(10000)
REWIND(SNACE)
COPYCF(SNACE,OUTPUT)
6/7/8/9
```

FILE contains overlay directives and binary decks which comprise the overlays to be written on file OVFILE when LOAD(FILE) is processed. Once execution of the overlay file begins, dumps will be labeled because of the DEBUG card. The following rules apply:

The DEBUG(S) card must precede the LOAD(FILE) cards, so that the loader tables will be placed on the overlay file as it is generated, and so that in (0,0) overlay the debugging routines will be loaded.

The NOGO card must appear in the overlay preparation part of the job; otherwise, the SNAP routine is written in the last overlay on the OVFILE file.

The SNAP cards must appear just before the OVFILE. card which re- initializes the loader tables and causes loading of the (0,0) overlay.

SEGMENT RUN

Using both TRACE and labeled dumps:

```
job card
DEBUG.                                    The DEBUG cards must appear be-
DEBUG(T)                                  fore any TRACE or SNAP cards;
TRACE(params)                             TRACE and SNAP cards must ap-
INPUT.                                    pear immediately before the
DMP(40000)                                load to which they apply.
REWIND(SNACE)
COPYCF(SNACE,OUTPUT)
7/8/9
(Segmentation loader directive cards and
object decks)
6/7/8/9
```

SNAP RUN

With SNACE output going to tape with tape to be listed in the next job:

```
job card
REQUEST,SNACE,*MT.
REWIND(SNACE)
SNAP(params)
INPUT.
COPYBF(X,SNACE)
EXIT.
COPYBF(X,SNACE)
7/8/9
(object deck)
6/7/8/9
job card
REQUEST,SNACE,MT.
REWIND(SNACE)
COPYCF(SNACE,OUTPUT)
6/7/8/9
```

The COPYBF(X,SNACE) cards write an end-of-file on the SNACE file. Normally, this is a faster method of running SNAP and TRACE when output to SNACE is extensive.

UTILITY PROGRAMS 8

The SCOPE library contains a set of peripheral processor (PP) and central processor (CP) utility programs which can be called by control cards or by keyboard entries.

Card-to-tape. tape-to-tape. tape-to-print. card-to-central storage. and central storage-to-punch operations, as well as, general file manipulation are possible. Utility operations can be performed with named files, each of which designates a specific peripheral device. such as a card reader. tape unit. printer. card punch or mass storage unit.

Before the first reference to any named file, a device should be assigned to it by the operator with the ASSIGN statement or by the programmer with the REQUEST statement; otherwise, the system will assign the file to a mass storage unit. All files. except mass storage, specify a unique peripheral device and all references to a specific device are made through the file name.

Utility jobs conform to the normal deck structure. The job deck contains the following cards:

Job card	First control card
Request cards	Equipment assignment
Program cards	Data operations
6/7/8/9	End of job

The job card includes name. priority. time limit and field lengths. If only utility programs are to be executed, a short field may be specified. In all copy operations, the central memory buffer is set up automatically to use the entire field length of the job. Operations between high speed devices may be accelerated with a larger field length.

The operator should be requested to assign equipment to all necessary files which do not reside on mass storage. Tapes can be rewound and positioned upon request. Each utility program is called by specifying its name starting in column 1. Parameters for execution of the program appear in a list after the name.

FILE COPYING

COPY TO END-OF-INFORMATION

COPY(file1,file2)

File1 is copied onto file2 until a double end-of-file (end-of- information) is detected on file1. Then both files are backspaced over the last file mark if a backspace is legal on the device. If parameters are omitted, INPUT, OUTPUT are assumed. COPY will not operate on S or L tapes, on labeled or BCD tapes. It is useful for binary files, disk files, and SCOPE standard tapes.

SPECIAL- PURPOSE COPY

The system library contains four special purpose copy routines: COPYCR, COPYCF, COPYBR, COPYBF. The following parameter information is pertinent for the four copy control cards that follow:

file1 and file2 name the input and output files. Information is copied from file1 onto file2. If names are not specified, INPUT and OUTPUT are assumed.

n is a decimal number indicating how many files or records are to be copied. If n is omitted only one file or record is copied.

COPY BINARY FILE

COPYBF(file1,file2,n)

COPY CODED FILE

COPYCF(file1,file2,n)

COPY BINARY RECORD

COPYBR(file1,file2,n)

COPY CODED RECORD

COPYCR(file1,file2,n)

The first two routines terminate when the specified number of files are read, or when an end-of-information is encountered.

The other two routines terminate when the specified number of records are read or when a file mark is encountered. For example, if the card specifies 100 records but the file contains only 50 records, the copy operation terminates after 50 records. When an end-of-reel is detected, the next reel is obtained, label checking/writing is performed if the tape is labeled, and the function continues normally on the next reel. If an end-of-file is encountered on the input file before the record count is exhausted, the copy operation will cease (but not abort) at that point. A message is entered in the dayfile. An EOF is written on file2 and backspaced over, and the file remains open.

If an end-of-information is encountered on the input file before the record/file count is exhausted, the copy operation will cease (but not abort) at that point. A message is entered in the dayfile; an EOF is written on file2; both files are closed.

The copy routines open the files specified on the copy control card. At the conclusion of the copy operation, only those files on which EOI has been encountered are closed.

Although not primarily implemented for that purpose, the copy routine is capable of limited format conversion. The following matrix shows format conversion copies that can be handled successfully:

USE OF UNQUOTED STRINGS

		OUTPUT		
		SCOPE	S	L
INPUT	SCOPE	Yes	Yes①	Yes
	S	Bin Yes / BCD Yes①	Yes	Yes
	L	Bin Yes / BCD Yes①	Yes①	Yes

NOTE:

① This kind of conversion cannot be guaranteed because of possible truncation of the input record. Maximum record size for S tape output files is 512 words (5120 coded characters). Maximum physical record size for coded SCOPE tapes is 1280 characters. If these sizes are exceeded, the output record is truncated; but the copy continues after a message is entered into the dayfile.

COPY BCD FILE

COPYBCD(file1,file2,n)

This routine copies packed output files to a magnetic tape where each line image is a discrete physical record, so the tape may be listed off line.

Control card formats and the interpretation follow:

COPYBCD.	*(INPUT,OUTPUT,1)*
COPYBCD(n)	*(INPUT,OUTPUT,n)*
COPYBCD(file 1)	*(file1,OUTPUT,1)*
COPYBCD(file1,n)	*(file1,OUTPUT,n)*
COPYBCD(file1,file2)	*(file1,file2,1)*
COPYBCD(file1,file2,n)	*(file1,file2,n)*

Any other format or illegal file names will cause the job to terminate with the dayfile message CONTROL CARD ERROR.

COPY SHIFTED BINARY FILE

COPYSBF(file1,file2)

A single file is copied from file1 to file2, each line is shifted right one character, and a leading blank character is added at the beginning of the line. If parameters are omitted, INPUT, OUTPUT are assumed.

This routine is used to format a print file where the first character of each line is to be printed rather than treated as a control character. The added blank character will result in single line spacing when the file is printed.

Examples:

Control cards to print a Hollerith card file. The Hollerith card file read by the operator-assigned card reader will be printed on OUTPUT file of job CARDCPY.

```
CARDCPY,P1,T100,CM10000.
REQUEST,CARDS,CR.
COPYSBF(CARDS,OUTPUT)
6/7/8/9
```

Copy utility programs are not limited to copy functions. They also may be used to position tape files to access desired data. Files or records may be skipped over by copying them to dummy files. For example:

Assume that a magnetic tape contains files in print format (each record contains a carriage control character in column one) and the third and fourth files are to be printed, to print the desired files:

```
TAPEPRT,T520,CM10000,P6,TP1.   (job card)
```

Assign a unique file name MAGTAPE with a REQUEST control card to a tape unit:

```
        REQUEST,MAGTAPE,MT.                Operator would assign specific
                                           tape unit.

        REWIND(MAGTAPE)                    Rewind tape unit to be sure of
                                           position.

        COPYCF(MAGTAPE,XX,2)               Skip tape to beginning of third
                                           file by copying first two files
                                           to an unused dummy file XX.

        COPYCF(MAGTAPE,OUTPUT,2)           Copy the two coded files to the
                                           output file, OUTPUT is automat-
                                           ically printed at end of job.

        6/7/8/9                            End-of-file card completes the
                                           job.
```

If the files are not in print format, they may be converted to a list format by using the COPYSBF utility. This routine adds the necessary control character to any file of coded information.

MISCELLANEOUS FILE UTILITIES

COPYXS

Since X tapes are not supported by SCOPE 3.3, the COPYXS routine is provided to convert binary X tapes to SCOPE standard format tape. A binary X tape has a logical structure containing 512-word PRUs with short PRUs of sizes that are variable multiples of CM words.

COPYXS(xlfn,scplfn,n)

xlfn	logical file name of the input X tape
scplfn	logical file name of the output SCOPE tape
n	number of files to be copied (default value is 1)

The COPYXS card is to be used in the following manner:

```
   .
   .
   REQUEST(xlfn,S)         Both tapes must be requested as S format.
   REQUEST(scplfn,S)
   COPYXS(xlfn,scplfn,n)
   .
   .
```

CAUTION: The output tape will be produced in SCOPE standard format, but will be flagged in the SCOPE tables as in S format. to be able to read the output tape in the same job, the following control cards would be needed:

```
UNLOAD(scplfn)
REQUEST(scplfn,MT)
```

The COPYXS routine cannot determine when the end-of-information occurs on an X tape; therefore, at least n files to be copied must exist on the X tape. Neither the input nor the output tape is rewound after conversion. After the requested number of files have been copied, the output tape is backspaced and positioned directly in front of the first tape mark preceding the SCOPE EOF trailer label. Subsequent files may be copied to the output tape; however, the block count in the trailer label will be incorrect.

Error recovery is handled by SCOPE. If a parity error persists after a number of recovery retries, a parity message appears on the display console and the copy operation may be abandoned by the operator because of the indeterminate state of the data.

COPYN

Files may be consolidated or merged with COPYN. Logical records from up to ten binary input files may be extracted and written on one output file (out). Input may be from tape, card, or mass storage files. Output may be to a tape, card, or mass storage file. Input or output files not defined by REQUEST cards are assumed to be mass storage files.

COPYN(p1,out,in1,in2,...,in10)

Text cards associated with the COPYN routine determine the order of the final tape. Several tapes may be merged to create a composite library tape. A routine may be selected from a composite tape, temporarily written on a scratch tape, and transmitted as input to a translator, assembler, or programmer routine, eliminating the need for tape manipulation by the second program. In its most basic form, COPYN can perform a tape copy.

Records to be copied may or may not have an ID prefix control number (12 bits), number of words in the prefix (12 bits), or the name associated with the logical record. A record ID format consists of the first seven characters of the first word of each logical record. Record format is indicated in the COPYN card by p1; a non-zero value indicates that ID of the logical records is to be omitted from the output file, zero indicates that records are to be copied verbatim. If records do not contain an ID, they are copied verbatim.

If logical records do not contain an ID, record identification cards must specify the record number–the position of the logical record from the current position of the file.

The file names (in) reference binary files on tape, mass storage, or cards; these are the input files. A binary tape file consists of the information between the load point and a double end-of-file; the tape file may contain any number of single end-of-file marks. A mass storage file ends at EOI, and a card deck must be terminated with a record separator (7,8,9 punch in column one). A file mark for an output tape is written by using a WEOF card in the desired sequence; or it may be copied in a range of records and counted as a record.

Text cards that may be used with COPYN are REWIND, SKIPF, SKIPR, WEOF (Write End-Of-File), and record identification cards. These cards are read from INPUT and terminated by a 7/8/9 punch in card column one. The text cards are free-field; they may contain blanks, but must include the separators indicated in each card description.

REWIND (p)

This card generates a rewind of file p which must be one of the input or output file names given on the COPYN control card. File p may not be the system INPUT file.

SKIPF (p, ± n)

With this card, n file marks on a tape file p may be skipped forward (+n) or backward (-n). Requests for other types of files will be ignored. No indication is given when SKIPF causes a tape to go beyond the double end- of-file or when the tape is at the load point.

SKIPR (p, ± n)

With this card, n records may be skipped forward (+n) or backward (-n) on tape file p. Zero length records and file marks must be included in n. Requests for other types of files will be ignored.

WEOF (p)

This card writes a file mark on file p, which must be an input or output file named on the COPYN control card.

The record identification card contains the parameters which identify a record or set of records to be copied from a given file.

p1,p2,p3

p1	Record to be copied or the beginning record of a set. The name associated with the record or a number giving the position of the record relative to the current position of the file may be specified.
p2	Last record to be copied in a set of records:

name	Logical records p1 through p2 are copied.
decimal integer	Number of logical records to be copied, beginning with p1. Zero length records and file marks are counted.
*	p1 through an end-of-file mark are copied.
**	p1 through a double end-of-file mark are copied.
/	p1 through a zero length record are copied.
0 or blank	Only p1 is copied.

p3	Input file to be searched. If p1 is a name, and p3 is omitted, all input files declared on the COPYN card are searched until the p1 record is found. If it is not located, a diagnostic is issued. If p1 is a number and p3 is omitted, the last input file referenced is assumed. If this is the first text card, the first input file on the COPYN card is used.

RECORD IDENTIFICATION CARD EXAMPLES

`SIN,TAN,INPUTA`	Copies all logical records from SIN through TAN from file INPUTA.
`SIN,10,INPUTA`	Copies 10 logical records from file INPUTA, from SIN through SIN+9.
`SIN,TAN`	Searches all input files beginning with current file. (If this is the first text card, the first input file named on the COPYN card is used.) When SIN is encountered, all logical records from SIN through TAN are copied.
`SIN,INPUTA`	Copies logical record SIN from file INPUTA.
`1,TAN,INPUTA`	Copies the current logical record through TAN from file INPUTA.
`1,10,INPUTA`	Copies 10 logical records, beginning with the current logical record on file INPUTA.
`1,*,INPUTA`	Copies the current logical record through the first file mark encountered on file INPUTA.

FILE POSITIONING FOR COPYN

Files manipulated during a COPYN operation are left in the position indicated by the previously executed text card; they are moved only during a search. If file name (p3) is omitted from the record identification card, all files on the COPYN card will be searched end-around. The end of a file is determined by a double end-of-file for tape or a single end-of-file for mass storage. The first input file declared is searched until either p1 or the original position of the file is reached, whereupon a search of the second input file begins. In this manner, all files remain effectively in the same position except the file containing p1, which is positioned at p2 + 1.

The output file is not repositioned after a search; so that the COPYN routine may be re-entered, if desired. Therefore, the programmer is responsible for any REWIND, SKIP, or WEOF requests referencing the output file.

Examples:

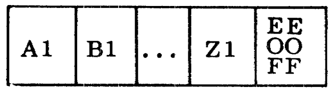
1. Record identification card: `REC,,INPUT1`

Input file INPUT1	ABLE	BAKER	. . .	REC	SIN	TAN	ZEE	EOF EOF

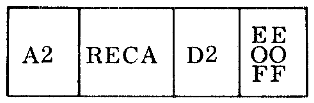
If INPUT1 were positioned at TAN, TAN and ZEE would be examined for REC. The double end-of-file would cause ABLE to be the next logical record examined, continuing until REC is read and copied to the output file. INPUT1 would then be positioned at SIN.

2. Record identification card: **RECA**

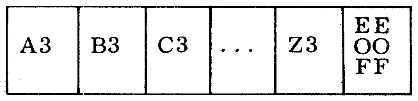
Input file INPUT1, positioned at B1



Input file INPUT2, positioned at loadpoint



Input file INPUT3, positioned at loadpoint



All routines from B1 through A1 are compared to RECA, and INPUT1 is repositioned at B1. A2 is compared, then RECA is copied to the output file, and INPUT2 is positioned at D2. INPUT3 is not searched.

3. Record Identification cards and binary logical records on INPUT file.

```
REC,,INPUT
JOB1,JOB3,INPUT
ABLE,,IN2
7/8/9  (Record separator)
REC (binary)
7/8/9
JOB1 (binary)
7/8/9
JOB2 (binary)
7/8/9
JOB3 (binary)
7/8/9
```

Because the INPUT file is not searched end-around, REC and JOB1-JOB3 must directly follow the requesting record identification cards in the order specified by those cards. An incorrect request for an INPUT record terminates the job.

ERROR MESSAGES

The text cards are written on the system OUTPUT as they are read and processed. When an error occurs, the abort flag is set, and a message is printed on OUTPUT followed by the card in error. This card is not processed, but an attempt is made to process the next text card. When the last text card is processed, the abort flag is checked; if it is set, the job is terminated. Otherwise, control is given to the next control card.

COPYL

COPYL(file1,file2,file3,program)

This routine allows selective replacement of one or more routines. File1 is the existing file; file2 contains the replacement routines; file3 is the resultant updated file. File1 is not rewound; file 2 is rewound. Processing continues until the end-of-file on file1 unless the fourth parameter is specified; in which case, it is the name of the last record to be copied from file 1 to file 3. The fourth parameter (a program name) may not contain any special character, such as $. Routines on file2 need not be in any order.

Example:

```
COPYL(OLD,CORR,NEW)
```

COPYL can be used to determine the contents and order of routines on a deadstart tape by declaring the deadstart tape to be the correction file for a dummy file, but this routine should not be used to edit the deadstart tape.

Example:

```
(job card)
REQUEST,SYSTAPE,HI.   Assign deadstart tape
REWIND(SYSTAPE)
COPYL(DUMMY,SYSTAPE,DUM)
6/7/8/9
```

COPYL will list all the routines on SYSTAPE in order as none exist on the file DUMMY.

Listing Messages:

COPYL DONE

COPYL DID NOT FIND xxxxxxx.

ILLEGAL COPYL PARAMETER
xxxxxxx UPDATED.

Display Message:

UPDATING xxxxxxx.

FILE MANIPULATION

REWIND

REWIND(file1,...filen)

All files specified are rewound.

UNLOAD

UNLOAD(file1,...filen)

When an UNLOAD card is encountered, a CLOSE, UNLOAD is performed on each file named. Magnetic tape units are returned to the control point pool. The units may be used by another job, but the number of units logically reserved by this job is not decreased. In contrast, the RETURN card results in a decrease of the number of units logically reserved.

SKIP OPERATIONS

SKIPF(lfn,n,lev,m)

One or more logical records will be bypassed in a forward direction. The request may be initiated at any point in a logical record.

SKIPB(lfn,n,lev,m)

One or more logical records will be bypassed in a reverse direction. The request may be initiated at any point in a logical record.

lfn	Logical file name (1-7 alphanumeric characters beginning with a letter).
n	Decimal number of logical records, or record groups, to be skipped; maximum value is 777776 (octal). A value equivalent to 777777(octal) will be treated as a rewind for SKIPB; for SKIPF a tape file will not be positioned and a disk file is positioned at end of information.
lev	Octal number. Logical records are skipped until the number of end-of-records with level numbers greater than or equal to the requested level is reached; the file is positioned immediately following (for SKIPF) or preceding (for SKIPB) the last record.
m	B for binary files, or C for coded files.

Default values are:

```
SKIPB(FILE,1,0,B)

SKIPF(FILE,1,0,B)
```

BACKSPACE LOGICAL RECORD

BKSP(file,n)

Multiple logical records are backspaced as specified by the decimal n. Backspacing terminates if it results in a rewound file.

COMBINE

COMBINE(f1,f2,n)

For this operation, n(decimal) logical records are read from file f1 and written as one logical record (level 0) onto file f2. The file is not positioned prior to initiating this operation. If the files f1 and f2 have not been previously defined by REQUEST cards, they will be assumed to be on mass storage.

COMPARE

One or more consecutive records on one file may be compared with the same number of consecutive records on another file to determine if they are identical. COMPARE is often used to verify a tape copy operation.

COMPARE(f1,f2,n,l,e,r)

fl,f2	Files to be compared
n	Number (decimal) of records in f1 to be compared to f2
l	End-of-record level number (octal)
e	Number (decimal) of non-matching words to be written to the OUTPUT file
r	Number (decimal) of counted records to be processed during the comparison. Included in non-matching record OUTPUT file if e parameter is given.

Comparison begins at the current position of each file and continues until the number of ends-of-records of the level specified or a higher level has been passed over. If all pairs of records are identical, the dayfile message is GOOD COMPARE; otherwise, it is BAD COMPARE. Discrepancies listed on file OUTPUT depend on the e and r parameters.

Example:

```
COMPARE(RED,BLUE)
```

Compares next record on file RED with next record on file BLUE.

```
COMPARE(RED,BLUE,6)
```

Compares next six records regardless of level of end-of-record marks; but each end-of-record on file RED must have the same level as the corresponding end-of-record on file BLUE for a GOOD COMPARE.

```
COMPARE(RED,BLUE,3,2)
```

Compares two files from their current positions up to and including the third following end-of-record with level number of at least 2. Both the records and end-of-record levels must match for a GOOD COMPARE.

A bad comparison will result only in the message BAD REC.n on OUTPUT; n is the record number, counting the first one read on each file as number 1. To obtain more information, errors and records must be specified as parameters.

```
COMPARE(GREEN,BLACK,3,2,5,1000)
```

This will do the same comparison as the previous example; but will list, on OUTPUT, the first five discrepancies between corresponding words in the files, together with their positions in the record. Positions are indicated in octal, counting the first word as 0. 1000 is the limit of pairs of records to be read in which discrepancies are found. 1000 is chosen as an infinitely large number greater than the number of records to be compared, because details are wanted about all discrepant records. If two long files were to be compared, for instance, 20 might be used as the records parameter, so that a reasonably large number of discrepancies would be described in detail; but if, through an error, the two files were completely different, an enormous and useless listing would not be produced. Furthermore, the comparison will be abandoned if this limit is exceeded, and the files will be left positioned where they stand.

A discrepancy between the levels of corresponding ends-of-records will be listed on OUTPUT, and the comparison will be abandoned, leaving the files positioned immediately after the unlike ends-of-records.

Mode need not be specified in the COMPARE card. It is handled in the following manner. The first record of the first-named file (GREEN) is first read in the binary mode. A redundancy check is produced; the file is backspaced and re-read in coded mode. If another redundancy check occurs, the fact is noted in file OUTPUT, the corresponding record of the second-named file (BLACK) is skipped over, and the process begins again. If the coded read is successful, the corresponding record of file BLACK is read in coded mode. If this record of BLACK produces a redundancy check, the fact is noted in file OUTPUT, and nothing further is done with that record. Each record of file BLACK will be read in the same mode as that in which the corresponding record of GREEN was successfully read; but if the record of GREEN was unsuccessfully read in both modes, the record of BLACK will be read in the same mode as the preceding record of BLACK. Once a record of GREEN has been read without redundance in one mode or the other, following records of GREEN are read in the same mode until a change is forced by a redundancy check. Mass storage records can be read indifferently in either mode, so that the above strategy imposes no difficulty if a tape file is being compared with a mass storage file; as long as the tape file is named first on the COMPARE card.

LISTMF

LISTMF,M=mfn,P=p.

LISTMF requests a list of the contents of multifile set tape, mfn. The multifile set must have existing status.

LISTMF rewinds the multifile set, mfn, and then positions to file p. If p is absent, a value of one is assumed. The contents of the label are extracted from the buffer and written to the OUTPUT file. Each succeeding file is positioned, specifying the previous position number plus one. Files are positioned and labels are listed until end-of-multi-file-set status is returned (code 21). The set remains positioned at end-of-set.

LABEL UTILITY

The LABEL utility program may be used to write labels, to check labels, and to position within a multifile set.

$$LABEL,lfn,\begin{Bmatrix}R\\W\end{Bmatrix},Y,L=x,V=x,E=x,\ T=x,C=x,F=x,\ M=x,P=x,D=x,\ N=x,X=x,VSN=vsn.$$

lfn	Logical file name, 1 to 7 characters; lfn must be specified, and must be the first parameter.
R or W	Read or write label. R or W is required and may appear anywhere following lfn.
Y	Indicates non-standard label is used. The non-standard label processor provided in SCOPE 3.3 processes CDC 3000 series labels (section 3). The LABEL utility program accepts parameters which conform to ANSI standard. The form in which the label is written depends upon the label processor. ANSI standard specifies a 17- character label name; the 3000 label processor truncates the name to 14 characters.
L=	File label name, 1 to 17 characters; up to 14 characters for Y labels. If absent, a default name is assumed. The existing tape is not checked. The special characters $, .) may be not be used.
V=	Reel number, 1 to 4 digits; 1 or 2 digits for Y label. If absent, 0001 is assumed and the tape is not checked.
E=	Edition number, 1 or 2 digits. If absent, edition 01 is assumed and the tape is not checked.
T=	Retention cycle, 1 to 3 digits. If absent, an installation defined value is written and the existing tape is not checked. When 999 is specified, an expiration date of 99999 is written on the new tape, establishing permanent retention.

C=	A 5-character Julian creation date. If absent, today's date is used and the tape is not checked.
F=	Format of the file data. If absent, data is written in SCOPE standard format. S specifies S tape format, L specifies L tape format.
M=	Multifile set name, 1 to 6 characters; mfn is a required parameter on utility calls that process multifile set tapes.
P=	Position number, 1 to 3 digits; specifies the position of a file in a multifile set. P may not be declared when the REQUEST card contains the N parameter.
D=	Density at which data is to be recorded or read. If absent and a 7-track tape is assigned, data will be recorded at the installation defined density.

LO	200 bpi	7-track tape
HI	556 bpi	
HY	800 bpi	
HD	800 cpi	9-track tape
PE	1600 cpi	

N= Character set to be used for coded data conversion on 9-track tapes. If absent, the installation-defined default character set is used.

US	USASCII code
EB	EBCDIC code

X= Disposition code. If absent, no disposition code is assumed.

IU	inhibit unload
SV	save
CK	checkpoint dump tape
CI	checkpoint tape and inhibit unload
CS	checkpoint tape and save

VSN = Volume Serial Number

SINGLE-FILE TAPES
(neither M nor P declared)

When R is declared, the LABEL utility may be used to read and check labels. When W is specified, labels may be written. Only standard or Y labeled tapes may be read or written. 7-track tape will be assumed unless a 9-track parameter was given.

Normally, LABEL is the first reference to lfn within a job; in which case, a REQUEST card is not needed. LABEL issues a REQUEST function for a standard or Y-labeled tape recorded at the last specified density and data format. A 7-track tape will be requested unless a 9-track parameter is used. If the specification is incorrect, the job will be terminated with a message to the dayfile. The contents of the label read or written will appear in the dayfile.

The tape remains positioned at the beginning of information. The label may be inspected later by issuing an OPEN function using a 13-word FET.

MULTI-FILE TAPES

(M declared; P may or may not be declared)

LABEL may be used to position and check a label on a file of a multifile set. Y-labeled multifile sets are not defined. The multifile set must have been previously assigned as the result of a REQUEST card/function. P may not be declared when a multi-file is being created. Data format and density fields will be ignored as these fields must have been declared on the REQUEST card.

If P is declared in conjunction with R, the set will be positioned at the beginning of information of the specified file by issuing a POSMF using parameters declared in the FET label fields. The field content of the label read will be noted in the dayfile.

If P is declared in conjunction with W, positioning will occur as above, and the label will be written. If the referenced file already exists, it will be evicted and the new label written by issuing a POSMF using parameters declared in the FET label fields. If end-of-set is encountered before reaching the referenced file, or if P is not declared, the label will be written at the end-of-set. The new file number will be one greater than the preceding file. In any case, the field content of the label written will appear in the dayfile.

If P is not declared, files will be positioned in ascending order, beginning at the current position + 1 until the correct file is located or until the entire set has been searched. The search is circular.

JOB UTILITIES

REQUEST FIELD LENGTH

With the RFL card, the user can request a different field length during job execution; nfl is the new octal field length.

RFL,nfl.

RFL should be employed to obtain optimal usage of central memory. For example, a FORTRAN program that requires 45000 words of memory to compile will need much less prior to compilation. RFL may be used as follows:

Example:

```
JOB,T300,CM45000,P7.
RFL(100)                       Reduce FL to process control card
REQUEST(TAPE,MT)
RFL(10000)                     Assign FL sufficent to do following steps
REWIND(TAPE)
COPYBF(TAPE,DISK)
UNLOAD(TAPE)
REWIND(DISK)
RFL(45000)                     Increase FL for compilation
FTN(I=DISK)
```

OCTAL CORRECTION ROUTINE

LOAD OCTAL CORRECTIONS (LOC)

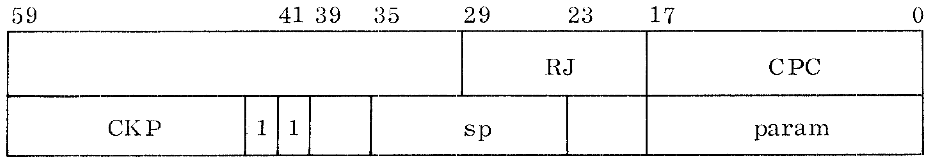
This peripheral program may be called with a control card or from the keyboard. Octal corrections are read from the INPUT file and entered in central storage. The octal correction cards must be in the following format:

```
1      7
23001  45020   04000   00042   00044
```

CHECKPOINT REQUESTS

Checkpoint dumps may be requested by CKP. control cards placed in the job stream where checkpoint dumps are desired. The CKP. control card does not contain parameters. Checkpoint dumps, however, are more often requested by an executing program or a console message entered by the operator. An executing program would request CHECKPOINT at various logical points, such as end-of-file, x logical records processed, x seconds of elapsed time, etc. CHECKPOINT requests may be issued more than once. The request takes the following form:

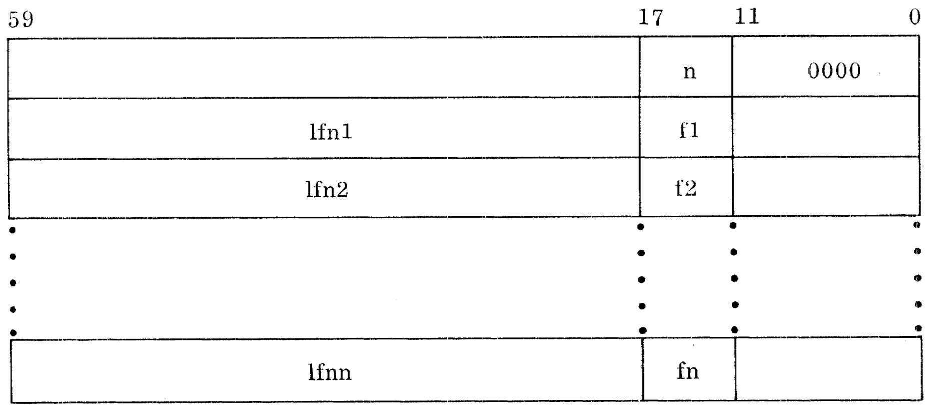
CHECKPT param,sp.



sp = zero indicates all mass storage files are to be processed

sp = nonzero indicates a limited set of files

param	Address of a parameter list within user's relocatable code; format follows:



n	defines number of terms in the list (number of lfn entries) to a maximum of 42(decimal).
lfn	name of user mass storage files in list.
f	flag indicating specific manner in which lfn is to be processed.

For a general call to CHECKPOINT using the CHECKPT macro, the sp field will be zero. Also:

If n = 0, all mass storage files assigned to the control point, including INPUT, OUTPUT, PUNCH, PUNCHB, and LGO, will be copied to the CHECKPOINT dump tape in the manner determined by the last code/status (f flags).

If n ≠ 0, all mass storage files named in the lfn list will be copied to the CHECKPOINT dump tape in the manner determined by the f flags; however system mass storage files will be copied as determined by the last operation performed on each file.

CHECKPOINT/RESTART 9

A job may be ended as the result of machine malfunction, operator error, or program error. Such an abnormal end may occur at any time during program execution.

Valuable machine time would be lost if an abnormal end occurred during one of the last steps of a long program and the program had to be restarted from the first job step.

CHECKPOINT/RESTART is a system facility which captures the total environment of a job on magnetic tape so that the job may be restarted from a checkpoint, rather than from the beginning of the job. Total environment includes local files associated with the control point of the job. For mass storage files (drum or disk), the complete file is captured as well as the relative position within that file. For magnetic tape files, only the relative position on the tape is captured, so the tape may be properly re-positioned during restart.

Checkpoint/Restart cannot handle:

Rolled-out jobs

Random files

Common files

Multi-file reels

Permanent Files

Each time a checkpoint dump is taken during execution of a job, a file is written containing all information needed to restart the job at that point. Each checkpoint dump is automatically numbered in ascending order by the system. It is recommended that a number of checkpoint dumps be taken during a long job, so that the user may return to any one of the previous checkpoints to restart his job. The maximum number of checkpoint dumps is limited only by the capacity of the checkpoint tape.

CHECKPOINT DUMP TAPE

With a REQUEST or LABEL control card, the user may specify an unlabeled or labeled tape with checkpoint disposition on which the checkpoint dumps are to be written. REQUEST or LABEL should be the first control card of the job. If no such tape is supplied, CHECKPOINT will define an unlabeled tape with the name CCCCCCC as a local file the first time CHECKPOINT is requested, including operator initiated checkpoints. In any event, only one checkpoint dump tape should be defined for the job.

CHECKPOINT/RESTART defines the following files for its use:

CCCCCCC
CCCCCCI
CCCCCCM

The user should refrain from using these file names.

Address begins in column 1; leading zeros may be dropped in the address. The data word begins in column 7; spacing in the data word is not important but the word must contain 20 digits.

`LOC.`	Reads all correction cards in the next INPUT file record and modifies central storage accordingly.
`LOC,1000.`	Clears central storage from the reference address to the specified address; correction cards are then read from the INPUT file.
`LOC(2022,3465)`	Clears central storage from the first specified address to the second; correction cards are then read from the INPUT file. This program may be called to clear storage by providing an empty record in the INPUT file.

The f flags can have only the following values:

f = 0	The mass storage file is copied from beginning of information to its position at checkpoint time; and only that portion will be available at restart. The file is positioned at the latter point.
f = 1	The mass storage file is copied from its position at checkpoint time to end of information; and only that portion will be available at restart time. The file is positioned at the former point.
f = 2 or 3	The last operation on the file determines how the mass storage file is copied. Generally, these values of f are used when sp is non-zero.

If the value of the sp field is non-zero in the macro call, only the lfn's supplied by the user in the param list, plus system files will be processed. Processing is determined by the f flag settings.

When the manner of copying a mass storage file is to be determined from the last operation on the file, CHECKPOINT derives f values from the last code/status as follows:

f = 0 if code/status ends in 4, 5, 6, or 7.

f = 0 if code/status ends in 0, 1, 2, or 3 and end-of-information bit is set.

f = 1 if code/status ends in 0, 1, 2, or 3 and end-of-information bit is not set.

Generally, these values cause the entire mass storage file to be copied for: write operations, read operations resulting in end-of-information status, and rewind operations (excluding some OPEN functions).

EXAMPLES:

FOR COMPASS USERS:

```
              CHECKPT   PARAM
              .
              .
PARAM         DATA      0
```

All mass storage files would be processed (sp=0, n=0) at CHECKPOINT.

All operator or CKP control card requests are processed in the same manner as the above example.

FOR FORTRAN (RUN) USERS:

```
DATA (variable=0)
.
.                                                 or
.                                                 variable=0
CALL CHEKPTR (variable)                           CALL CHEKPTR (variable)
```

FOR FORTRAN EXTENDED USERS:

```
DATA (variable=0)                                  or
.                                                  variable=0
.                                                  CALL CHEKPTX (variable)
.
CALL CHEKPTX (variable)
```

For the above examples, checkpoint processing is performed for all mass storage files as described in the first example where sp=0, n=0. Selected files may be processed in the manner shown in the following example:

```
DIMENSION KPARAM(4)
KPARAM(1)=30000B
KPARAM(2)=5LTAPE1.OR.10000B
KPARAM(3)=6LTAPE23.OR.10000B
KPARAM(4)=5LTAPE3
.
.
.
CALL CHEKPTR(KPARAM,1)
or
CALL CHEKPTX(KPARAM,1)
```

OVERLAY PROGRAMS

The user should rewind the overlay files prior to requesting CHECKPOINT if they are on mass storage.

FORTRAN overlay programs should declare the mass storage overlay files to be TAPEn files and use REWINDn before CALL CHEKPTR (variable) or CALL CHEKPTX (variable).

EXAMPLE FOR FORTRAN (RUN):

```
OVERLAY(TAPE9,0,0)
PROGRAM MAIN(...,TAPE9)
DATA FILE/5LTAPE9/
.
.
CALL OVERLAY(FILE,1,0)
.
.
END
OVERLAY(TAPE9,1,0)
PROGRAM OVER1
DATA PARAM/0B/
.
.
REWIND9
CALL CHEKPTR(PARAM)
.
.
END
```

RESTART REQUEST

The RESTART control card directs a job to be restarted from its checkpoint tape:

RESTART,name,n,Sxxx.

name	Name of checkpoint file as defined at checkpoint time. This parameter is required and it must be the first parameter.
n	Number (decimal) of checkpoint to be restarted. If this parameter is omitted, a default value of 1 is assigned by RESTART. If it is greater than the number of the last checkpoint taken, the restart attempt will be terminated.
Sxxx	xxx is the number (decimal) of words in the smallest physical record on any S tape attached to the job. If this parameter is omitted, standard SCOPE physical record size (512) is assumed.

WARNING: If any S tapes with small physical records are attached to the job and the S parameter is not set to reflect this, the tapes may not be positioned correctly.

After locating the proper dump on the checkpoint tape, the restart program requests all tape files defined at checkpoint time, and repositions these files. The restart program also re-establishes all mass storage files from the copies appearing on the checkpoint tape, restores the central processor program, and restarts the user's job.

The restart job should not contain a REQUEST control card or a LABEL card requesting the file name given on the RESTART card unless the CHECKPOINT dump tape is a labeled tape. If no checkpoint tape is requested, RESTART will issue a request for the checkpoint file named on the RESTART control card.

If any permanent files are attached to the control point when CHECKPOINT is called, only the permanent file names (not the files) are copied to the tape. The permanent file names, cycle numbers, and corresponding logical file names are listed in the job dayfile.

When restarting, the user should attach all these permanent files to the control point, and reposition them before calling RESTART. RESTART will check that this has been done; if not, RESTART will terminate with a diagnostic message.

Any ECS user area attached to the control point will be copied in its entirety to the checkpoint tape. At restart time, it will be recopied to ECS from the checkpoint file. On the job card for the restart job, the user must request at least as much ECS as was attached to the original. If reconfiguration results in insufficient ECS available to the user, restart is not possible.

UNRESTARTABLE CHECKPOINT DUMPS

A checkpoint dump may not be restarted in the following cases:

A tape file necessary for restarting the program was overwritten after the checkpoint dump was taken.

A machine error propagated bad results but did not cause abnormal termination until after another checkpoint dump.

LIBRARY PREPARATION AND MAINTENANCE 10

UPDATE

UPDATE is a system program used to create, manipulate, and maintain library files. It is used most frequently by installations to maintain a program library containing source decks for the SCOPE operating system. UPDATE is flexible enough, however, to be used in the maintenance of user's source and data files written in either sequential or random library file format.

UPDATE determines what specific operations it is to perform from two sources. One source is the parameters on the UPDATE control card. The other source is the UPDATE input stream which contains directives and text.

UPDATE TERMINOLOGY

A program library is a file created and maintained by the UPDATE program. It is in the form of card images together with sufficient information to allow each card image to be uniquely referenced by UPDATE. UPDATE provides a means by which the user can manipulate these card images on the program library; it also produces output files in a format suitable for use as compiler/assembler input. The logical file name most often used to reference a program library is OLDPL, and the name OLDPL is commonly considered to be synonymous with old program library.

UPDATE determines the specific operations it is to perform from two sources. One source is the parameters on the system control card that calls UPDATE. The other source is the input stream consisting of directives and text. In a directive card image, column 1 contains the special character specified as the master control character for this UPDATE run; * is the default character and is used most frequently. The control character is followed by a string of characters terminated by a blank or comma. This character string must be defined in UPDATE as a valid directive. Parameters may follow the terminator.

As each card image is read from an UPDATE input stream, it is assigned a unique card identifier consisting of two parts. The first part is the ident name. Each card image is a member of only one UPDATE ident from which it obtains the ident portion of its identifier. An ident is normally established by the introduction of a *IDENT directive; however under certain conditions, it may be established by a *DECK or *COMDECK directive. The second part of the identifier is a sequence number within the ident. The specific card image is, therefore, identified and referenced by the identifier idnam.seqnum where idnam is the ident under which the card was introduced and seqnum is the sequence number within that ident. Once this identifier is established, it is retained along with the card image on the OLDPL; and UPDATE directives may reference the card image by its identifier. UPDATE directives which apply to an entire ident apply to the ident name and to each card image belonging to that ident.

Some UPDATE directives exist only in the input stream. Other directives are assigned identifiers and are placed on the OLDPL. Since these directives possess identifiers, they may be referenced thereafter in the same manner as text card images. Card images exist on the OLDPL in one of two states, active and inactive. For text card images, the state is important only for COMPILE file output. Only currently active card images are written to the COMPILE file. For UPDATE directives on the library, the state has more meaning. Normally, these directives are processed when the COMPILE file is generated; they are not written to it, but specify certain operations which determine the contents of the file. Inactive directives are ignored.

Each card image on the OLDPL also belongs to a deck. A deck is defined as an active *DECK or *COMDECK directive along with all card images which follow on the OLDPL (active or inactive) up to the next active *DECK or *COMDECK directive. The deck to which a card image belongs is determined by the card's position and by the status (active or inactive) of the *DECK or *COMDECK directives. Since the *DECK and *COMDECK directives can be deactivated (by *DELETE, *YANK and *SELYANK), card images which belong to one deck at the beginning of an UPDATE run may belong to a different deck at the end of the run. When a *DECK directive is deactivated, all card images in the deactivated deck become members of the preceding deck on the OLDPL and thereafter will be affected by directives which affect the previous deck as a unit, such as *PURDECK and *SEQUENCE.

PROGRAM LIBRARY

In SCOPE 3.3, three system programs are available for the preparation and maintenance of library files: EDITSYM, EDITLIB and UPDATE. EDITSYM and UPDATE are concerned with symbolic library files. EDITLIB is concerned with files containing object programs and overlays, such as those in the SCOPE system library file. This section of the SCOPE Reference Manual deals with UPDATE. EDITSYM is described in Appendix E and EDITLIB and is described in the EDITLIB System Programmer's Bulletin.

The program library, which UPDATE creates and modifies, contains card images in the compressed format used by UPDATE and the history information relevant to the card images.

Card images on the program library are grouped arbitrarily into decks. A deck is defined by a *DECK or *COMDECK directive and includes all card images up to but not including the next *DECK or *COMDECK directive or the end of the library. UPDATE decks should not be confused with compiler PROGRAM statements or assembler IDENT cards. An UPDATE deck can contain any number of PROGRAM statements, assembler IDENT cards, or sets of data.

PROGRAM LIBRARY FILE FORMATS

The UPDATE program can create and maintain symbolic library files written in two distinctly different formats: random and sequential.

RANDOM FORMAT

In a random format program library, each source deck is written as a separate variable length logical record. The file contains additional separate logical records to contain the directory, the decklist, and the *YANK and *SELYANK directives.

The first two words of the random program library index contain the length and disk address of the deck list record and the directory record. The third index word contains a display code key word, and the fourth and fifth index words contain the label of the program library, if present. The second word of each deck list entry contains its disk address.

Source cards on the program library are written as compressed symbolic card images. Each compressed image is prefaced by at least one word of control information, containing the correction history bytes (CHB) for the card. The first word gives the card sequence number, the card activity status, the number of words used by the compressed card image and the master CHB. The master CHB contains the deck or correction set under which the card was introduced. Additional CHBs may contain correction sets that have altered the status of the card.

Using the program library in random format can result in a substantial saving of time as detailed below. When Q mode is specified, it is not necessary for UPDATE search the library for the desired deck; the deck may be referenced directly. When the random library is cataloged as a permanent file with multi-read access, several users may use the library at one time.

This example demonstrates the creation of a random library as a permanent file:

```
(job card)
LABEL(OLDPL,R,L=SCOPESYSTEMLIB)
UPDATE(A)
CATALOG(NEWPL,RANDOMPL, ID=ME,CN=PW,EX=PW,MD=PW)
RETURN(OLDPL)
6/7/8/9
```

Any number of users may attach and read or write the random library created above (as in the next example) without requiring tape drives or waiting for the library tape to become available:

```
(job card)
ATTACH(OLDPL,RANDOMPL,ID=ME)
UPDATE(Q)
COMPASS(I=COMPILE,S=SCPTEXT)
7/8/9
*IDENT FIXCIO
(input stream)
*COMPILE CIO
7/8/9
```

SEQUENTIAL FORMAT

UPDATE may create new library files in sequential format. On magnetic tape, sequential format library files are written in SCOPE internal tape format. The entire sequential format library file is written as one binary record. The first word in the file is a display code key word; the second is a counter word containing the number of deck names in the deck list and a count of the card image identifiers in the directory; the last word in the file is a checksum.

The deck list starts in word three of the library record; it is a table containing one entry for each program deck in the file. Each entry is a 60-bit word containing a display-coded deck name, left adjusted with zero fill. The deck names are listed in the same sequential order as they appear in the library file.

The directory follows the deck list and contains the library identifier for each program deck that has ever appeared in the file. Each entry contains the deck identifier name in display code, left adjusted with zero fill. Program identifiers differ from deck names in that the deck name is taken from the *DECK or *COMDECK directive which precedes the source deck in the input stream. Program identifiers are taken from the *IDENT directive which precedes correction sets in the input stream. Identifiers are listed in the directory in the order in which they were introduced into the library. Identifiers for purged decks are not printed on the listable output, however they are retained and flagged in the directory. These purged deck identifiers are removed when the update E option is selected. The number of identifiers in a library file is limited by the amount of available memory in the UPDATE job field length, since each identifier in the directory requires one word of table space.

The remainder of the sequential format library file contains the compressed symbolic card images and their correction history bytes, as described for random format library files.

OPERATING REQUIREMENTS

A field length of at least 40000 octal words should be specified for most UPDATE runs. After each run, a message in the output will indicate the amount of core used.

The following two tables are a quick reference to the UPDATE parameters and directives.

Parameter	Function
A	Copy sequential OLDPL to random NEWPL
B	Copy random OLDPL to sequential NEWPL
C	Compile file in deck list order
D	80 columns of data
E	Edit mode
F	Full UPDATE mode
G	File containing PULLMOD output
I	Input file
K	Compile file in *COMPILE card order
L	Type of listing desired
M	Library to be merged with P
N	New program library
O	Output file
P	Old program library
Q	Quick UPDATE mode
R	No REWIND option
S	Source file
T	Source file without common decks
U	Continue with errors
W	NEWPL on disk to be sequential
X	Compile file in special format for COMPASS 2
Z	Input stream is in squeezed format
8	80-column compile file
*	Control character
/	Comment character

TABLE 10-2. UPDATE DIRECTIVES

Abbreviation	Directive Format	
none	*ABBREV	
*AF	*ADDFILE	filename,deckname
*B	*BEFORE	ident.seqnum
*C	*COMPILE	dname1,dname2,...dnamen
	*COMPILE	dname1.dname2
*CA	*CALL	comdeck
*CD	*COMDECK	deckname
	*COMDECK	deckname,NOPROP
*CH	*CHANGE	ident1,ident2
*CW	*CWEOR	level
*CY	*COPY	dname,ident1.seqnum,ident2.seqnum,filename
	*COPY	dname,ident1.seqnum,ident2.seqnum
*D	*DELETE	ident1.seqnum,ident2.seqnum
	*DELETE	ident.seqnum
*DC	*DECLARE	dname
*DF	*DEFINE	name,name1,...namen
*DK	*DECK	deckname
none	*DO	idnam1,idnam2,...idnamn
*DT	*DONT	idnam1,idnam2,...idnamn
*EI	*ENDIF	
*ET	*ENDTEXT	
none	*IF	param,name,num
*I	*INSERT	ident.seqnum
*ID	*IDENT	idname
*L	*LIST	
*LT	*LIMIT	num
*M	*MOVE	deckname1,deckname2
*NA	*NOABBREV	
*NL	*NOLIST	
*P	*PURGE	ident1,ident2,...identn
*PD	*PURDECK	dname1,dname2,...dnamen
	*PURDECK	dname1.dname2
*PM	*PULLMOD	ident1,ident2,...identn
*R	*RESTORE	ident1.seqnum
	*RESTORE	ident1.seqnum,ident2.seqnum
*RD	*READ	filename
*RW	*REWIND	fname
*S	*SEQUENCE	deckname1,deckname2,...decknamen
	*SEQUENCE	deckname1.deckname2
*SK	*SKIP	filename,number
*SP	*SELPURGE	deckname1.ident1,deckname2.ident2,...
*SY	*SELYANK	deckname1.ident1,deckname2.ident2,...
*T	*TEXT	
*W	*WEOR	level
*Y	*YANK	ident1,ident2,...identn
	*YANK	ident1.ident2
*YD	*YANKDECK	dname1,dname2,...dnamen

FILES PROCESSED BY UPDATE

The user may select other than the default values for a file by equating the letter designator to the desired file name on the UPDATE control card. For example P=OLDFILE changes the old program library file name from the default value, OLDPL, to the substitute OLDFILE. Any file over which the user has control may be assigned to magnetic tape. System program library tapes are all labeled.

Table 10-3. UPDATE FILES

File	Function	Type	Position after UPDATE Call
Input	Provides control information	Coded	Remains at end of record terminating UPDATE control cards. Other cards may follow. If UPDATE aborts, location of input file is unpredictable.
Output	Produces listings	Coded	Current position of OUTPUT file unless O is equated to a file other than OUTPUT. File is not rewound.
Compile	Produces card images for assembly and/or compilation	Coded	Rewound before and after UPDATE operation.
Old Program Library	Contains old program library	Binary	Rewound before and after UPDATE operation.
New Program Library	Contains new program library	Binary	Rewound before and after UPDATE operation.
Source	Contains copy of all active cards on program library except YANK, SELYANK and YANKDECK cards. The cards contain no sequencing information	Coded	Rewound before and after UPDATE operation.
Merge	Secondary library to be merged into new program library	Binary	Rewound before and after UPDATE operation.
Source	File containing PULLMOD output	Coded	Rewound before and after UPDATE operation.

By default, any output resulting from *PULLMOD is appended to the SOURCE file. The user may write this information to a different file by equating G on UPDATE control card to desired file name. Any rewind option which applies to the S (SOURCE) file also applies to the G file.

Files mentioned in ADDFILE, READ, REWIND, and SKIP operations are left as defined by the latest directive encountered on the input stream.

The files OLDPL, NEWPL, MERGE, COMPILE, and SOURCE normally are rewound before and after the UPDATE operation. The R parameter on the UPDATE control card may be used to specify which files are to be rewound. Any action specified for OLDPL also applies to MERGE.

UPDATE CONTROL CARD

The following SCOPE control card calls the UPDATE program:

UPDATE(parameter list)

The control card parameters specify the mode of the update and the files to be used. The parameter list is begun with a comma or left parenthesis and terminated by a period or a right parenthesis; parameters in the list may appear in any order. Each parameter must begin with one of the letters given in the table below. If the letter is followed by an equals sign, a character string which defines a vah.e to be substituted for the parameter default value must follow the equals sign. File name parameters are limited to seven characters. Parameters are separated by commas; when a parameter is not specified, a default value given in the parameter value table is assumed.

Table 10-4. PARAMETER VALUES

Letter	Function	If not specified	Value to right of equals sign	Value if equals sign is omitted
P	Old program library file is processed by UPDATE.	File name OLDPL assumed	Old library file name	File name OLDPL assumed
N	New program library fie is produced.	No new program library	New library file name	File name NEWPL assumed
I	Input stream contains directives and text processed by UPDATE.	File name INPUT assumed	Input file name	(Illegal)
O	Listable output file produced.	File name OUTPUT assumed	Output file name	(Illegal)
G	File containing PULLMOD output is produced.	PULLMOD output appended to source file	File to contain PULLMOD output	Output written to file name SOURCE
K	Same as C, except decks are written in the order specified on *COMPILE directives.	File name COMPILE assumed. Default is C	Compile file name	Output written to file name COMPILE

Table 10-4 (continued)

Letter	Function	If not specified	Value to right of equals sign	Value if equals sign is omitted
C	File produced contains source language decks in the order they appear on input and/or old program library, for assembler/compiler input.	File name COMPILE assumed	Compile file name; C=0 disables output to compile file	Output to file name COMPILE
T	Same as S, except common decks and *COMDECK directive card images are not written to the source file.	Source file not written	Source file name	File name SOURCE is assumed
S	Source file produced contains 80-column card images of all active cards, as well as their *DECK or *COMDECK directives.	Source file not written	Source file name	File name SOURCE is assumed
M	Secondary library file is merged with an old program library file to form a new program library file.	No merging takes place	Name of secondary file	File name MERGE is assumed

Table 10-5. UPDATE ACTION PARAMETERS

Letter	Specified Action	Default Action
F or Q	Selects Full or Quick mode for correction runs.	Normal (selective mode) is assumed (neither Q nor F).
A	Selects mode used to copy sequential OLDPL to random NEWPL. Suppresses all other action parameters except * and /.	None.
B	Selects mode used to copy random OLDPL to sequential NEWPL; suppresses all other action parameters except * and /.	None.
D	COMPILE file will contain 80 columns of data.	COMPILE file will contain 72 columns of data.
C=PUNCH	Selects both D and 8 options, as well as selecting PUNCH as compile file name.	None.
E	The OLDPL will be edited.	No editing will be done.

Table 10-5 (continued)

Letter	Specified Action	Default Action
L=code	Selects type of listable output to be produced (see Listable Output).	L=A1234 on correction run; L-A12 on creation run.
R=list	Files specified in list will be rewound before and after UPDATE run. Only file parameter letters P, C, S, N may appear in list.	Files identified by P, C, S and N are rewound before and after UPDATE run.
R	No rewinds will be issued on files identified by P, C, S, and N.	Files identified by P, C, S and N are rewound before and after UPDATE run.
U	UPDATE continues regardless of normally fatal errors.	None.
W	New program library file will be in sequential format.	New library file will be in random format (if possible).
X	Compile file will contain 2-word header and compressed card image for each card written to the compile file. This parameter is for COMPASS 2.0 interface.	Compile file will contain uncompressed card images.
Z	UPDATE input file is assumed to be in compressed symbolic format. This option does not affect *READ files.	Input file is assumed to be in normal card image format.
8	Compile file output will contain 80-column card images.	Compile file output will contain 90-column images.
*=char	UPDATE master control character must precede a directive name. Any character having a display code value of 01 to 54 (octal) inclusive may be used, except for left and right parentheses (51 and 52 octal).	Master control character will be *.
/=char	Comment control character in column 2 listed with a correction set. Any character with display code value 01-54 may be used, except parentheses (51 and 52).	Comment control character will be /.

UPDATE CORRECTION MODES

The Q and F special options determine the type of processing for UPDATE correction runs. A correction run is defined as any run in which the first card (other than file manipulation, *ABBREV and *NOABBREV) on the input file is not a *DECK or *COMDECK card. The modes in which UPDATE may operate on a correction run are:

Full mode (F selected)

Quick mode (Q selected)

Normal (selective) mode (neither F or Q selected)

Q overrides F if both options are selected.

NORMAL (SELECTIVE) MODE

Selective mode is the default correction mode when neither the F nor Q option is selected. All decks and comdecks on the library are processed by UPDATE. The NEWPL, if specified, will contain all decks and comdecks in the order of their occurrence on the OLDPL (after any corrections have been made). The source file, if specified, will contain all active cards which would be written to the NEWPL if it were created. The compile file will contain all decks modified during this UPDATE and all decks specified on *COMPILE cards. When a deck calls a modified comdeck, the calling deck is also considered to have been modified, unless the *COMDECK directive specifies NOPROP.

FULL MODE

When F is specified, source and compile files, if specified, contain all decks on program library file, unless indicated otherwise. NEWPL is the same as in normal mode. *COMPILE cards are ignored.

QUICK MODE

The Q option supersedes the F option, if both options are specified. In Q mode, only decks specified on *COMPILE cards are processed. Modifications (except *ADDFILE) which reference cards in decks not specified on *COMPILE cards are not processed; and UPDATE will abort after printing the unprocessed corrections. The compile file will contain decks specified on *COMPILE cards. Contents of source and NEWPL files are dependent on the type (random or sequential) of OLDPL and on the deck names specified on *COMPILE cards.

SEQUENTIAL FORMAT OLDPL FILE:

NEWPL will contain all decks specified on *COMPILE cards as well as comdecks called by them. Comdecks encountered on the OLDPL before the last explicitly mentioned deck are also placed on the NEWPL. Source file will contain all active cards that would be written to NEWPL if it were selected.

Warning: Under a Q mode UPDATE using a random OLDPL, a single correction IDENT containing corrections to both a DECK and a COMDECK may cause trouble if the COMDECK logically precedes the DECK on the OLDPL. Such an UPDATE will run, and no errors will be detected. However, if the same run is repeated with the N parameter specified on the UPDATE card, the sequence numbers assigned to the text cards in the correction set referred to in the second run will not be the same as they were in the first run. This situation cannot be prevented without sacrificing the speed for which Q mode was designed. The correct sequence numbers are those assigned when N is specified.

RANDOM FORMAT OLDPL FILE:

NEWPL and source file will look the same as above, except only the comdecks called by decks specified on *COMPILE cards will be included on NEWPL.

COMPILE FILE

The compile file parameter on the UPDATE control card determines the name of the logical file, as follows:

Parameter	**Meaning**
Omitted	Output on file name COMPILE
C or K	Output on file name COMPILE
C=lfn;K=lfn	Output on logical file name given for lfn
C=0	Compile file not to be produced

During a file creation run or during a full update (F mode), all decks in the program library are written to the compile file. During a normal (selective) UPDATE run, all decks modified during the run, plus all decks specified on *COMPILE directives, are written to the compile file. During a quick update (Q mode), only decks specified on *COMPILE are written to the compile file. By default, decks are written to the compile file in the order encountered on the OLDPL. Decks being updated are written to the compile file after corrections have been made. In full update mode, the order in which decks are written to the compile file cannot be changed.

In normal and quick modes, the order can be altered by using the K file identifier on the UPDATE control card. When K is used, decks are written to the compile file in the order of specification on *COMPILE. In normal mode, after all decks listed in *COMPILE are written to the compile file, they are followed by any decks modified by the run but not mentioned on *COMPILE. These modified decks will be written as they are encountered on OLDPL. If both K and C file parameters appear on the UPDATE control card, the K option will take precedence.

SOURCE FILE

A copy of all active card images and their *DECK or *COMDECK directives are written as 80-column card images. The source file parameter on the UPDATE control card determines the logical file on which they are written.

Parameter	**Meaning**
Omitted	Source file not written
S or T	Output to file name SOURCE
S=lfn; T=lfn	Output to file name given as lfn

When the T parameter is used, no common card deck images will be written to SOURCE. If both S and T are used, the T parameter will take precedence. All decks are written on SOURCE as they are encountered on the OLDPL. The content of SOURCE is determined by the UPDATE operating mode, by the deck names on *COMPILE directives, and by the format (random or sequential) of the OLDPL.

In full mode and in normal (selective) mode, SOURCE will contain all cards to recreate and resequence the library, including all *DECK, *COMDECK, *WEOR, *CWEOR, *CALL, *TEXT, and *ENDTEXT directives in addition to all active card images.

In quick mode, SOURCE contents are determined by the format of the library file. With a sequential OLDPL, all decks named on *COMPILE directives are copied, as well as all comdecks called by those decks. Common decks encountered on the OLDPL before the last named deck are also copied to SOURCE. With a random format OLDPL, only decks named on *COMPILE directives and the common decks called by those decks are written to the SOURCE file.

CARD IDENTIFICATION

Card identification assigned by UPDATE is permanent and can be changed only through the *SEQUENCE directive. Card identifiers are in the explicit format:

ident.seqnum

where ident is the identifier that introduced the card deck; seqnum is the sequence number within the identifier.

UPDATE identifiers are established on a creation run by *DECK and *COMDECK cards and on a correction run by *IDENT cards or (in *ADDFILE mode) by *DECK or *COMDECK cards.

On *BEFORE, *INSERT, *DELETE, *RESTORE, and *COPY cards, normally one or more fields contain the complete identifier for a specific card image on the program library, the ident.seqnum. This identifier may be abbreviated as .seqnum or seqnum. The presence or absence of the leading period is critical. UPDATE maintains two default ident names to be used in expanding the abbreviated form to the complete form. These two default names are initially set to YANK$$$ and are updated in the following manner. One of the two default names will be updated whenever a new ident name is encountered in the complete identifier. The default name may be referenced by seqnum. If the ident name is also a deck name known to UPDATE, the default of .seqnum will be updated.

To summarize:

Short Form	**Expands to**
.seqnum	idnam1.seqnum
seqnum	idnam2.seqnum

where idnam1 is the last explicitly named identifier which is also a deck name known to UPDATE, and idnam2 is the last explicitly named identifier whether or not that identifier was a known deck name. By default, the card identification will be:

YANK$$$.seqnum

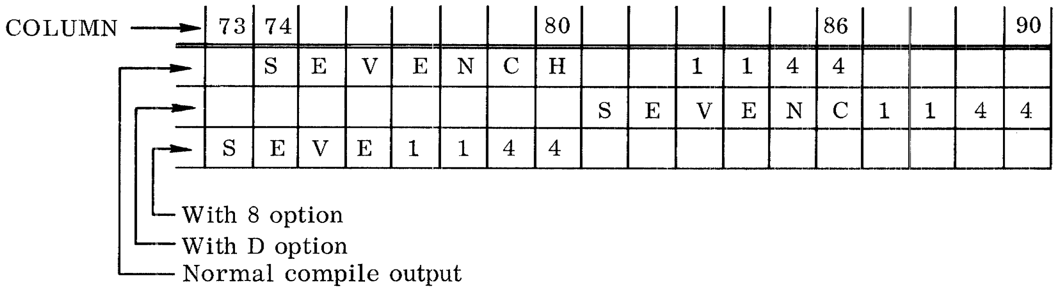
Deck names and card identifiers appearing on directives other than *BEFORE, *INSERT, *RESTORE, *DELETE, or *COPY, do not change the default values. The default feature may not be used on any other directive.

Card identifiers and deck names may consist of not more than nine characters. Any character with an octal display code value of 1 through 54 may be used. Card identification appears on UPDATE output in the form ident.seqnum as in previous versions of UPDATE; however, the compile file output may be modified.

COMPILE FILE SEQUENCE FIELDS

UPDATE will attempt to include sequence information in the compile file output. This information is compressed into the available space at the end of the card image. If the entire card identification will not fit, UPDATE will truncate the ident field until it fits, leaving the sequence number intact. When both the D and 8 options are elected (or C=PUNCH), no sequencing information will be appended to the output.

In this example, the ident is SEVENCH, the sequence number is 1144:



COLUMN	73	74						80						86				90
Normal compile output		S	E	V	E	N	C	H			1	1	4	4				
With D option									S	E	V	E	N	C	1	1	4	4
With 8 option	S	E	V	E	1	1	4	4										

UPDATE DIRECTIVES

UPDATE directives are cards in the input stream which contain a master control character in column 1, a valid UPDATE directive beginning in column 2, and a blank or comma immediately following the directive. The default master control character, *, will be used in subsequent examples, however, the master control character may be redefined through the *=parameter on the UPDATE control card. The master control character is stored in the key word that prefixes the library file created; therefore some consistency in use of the redefining parameter should be used, especially with regard to directives that UPDATE places into the library file. Directives CALL, DECK, COMDECK, WEOR, CWEOR, TEXT, SELYANK, YANKDECK, ENDTEXT and YANK, when placed on the library, carry with them the master control character with which they were introduced. UPDATE can recognize them on subsequent runs only when the same master control character is given. If file modification is attempted with a master control character other than that specified on OLDPL, a non-fatal error will occur; and processing will continue after the user changes the control character to the one specified on the OLDPL.

UPDATE directives are written in the form:

**control word*

or

**control word   parameter list*

where * represents the master control character (* by default), and the control word follows without any intervening spaces. The control word is terminated by a comma or one or more blanks; any number of additional blanks may appear between the control word and the parameter list. Parameters are separated from each other by commas; embedded blanks are not permitted. Control words may be written in an abbreviated form, as shown in table 10- 2. All numeric values are to contain decimal digits, except where noted.

FILE MANIPULATION

When text and directives are to be read from files other than the input stream (I file parameter), file manipulation directives are used to direct UPDATE. The directives may appear anywhere in the input stream; they may not appear on any file other than the main input file and may not reference files specified for internal use by UPDATE, such as those identified by the file parameter letters.

**REWIND fname*

This directive causes the file indicated by fname to be rewound or repositioned to the beginning of its information.

**SKIP fname,n*

*SKIP causes the UPDATE input file fname to be spaced forward over n number of records. If n is omitted or specified as zero, the input file is spaced forward over one logical record.

**READ fname*

UPDATE is directed to read from file fname until an end-of-record mark is encountered, at which point UPDATE resumes reading from the main input stream containing the *READ directive. Any directives, except *SKIP, *REWIND, *READ and *ADDFILE, may appear on fname which serves as an alternate input stream to UPDATE.

**WEOR n*

n = decimal number corresponding to a SCOPE end-of-record level number.

The *WEOR directive is used to organize the compile file for ease of input to compilers, assemblers, etc. It writes an end-of-record on the COMPILE file. Level 0 end-of-record is assumed if n is not specified. The *WEOR directive, itself, is not written on the COMPILE file. *WEOR may be inserted anywhere in a deck on the program library; it becomes effective only when that deck is written to the COMPILE file.

**CWEOR n*

The *CWEOR directive acts the same as *WEOR except if the compile file has not been written or the current record is empty, the end of record is not written.

LIBRARY FILE CREATION

UPDATE operates in either of two basic modes: creation mode or correction mode. UPDATE considers any run a creation run in which the first card image (other than file manipulation directives, */comments, *DECLARE, *ABBREV or *NOABBREV) in the input stream is a *DECK or *COMDECK directive. Although an old program library file may be assigned to the job, UPDATE will ignore its existence.

In a creation run, each *DECK and *COMDECK in the input stream defines a deck to be incorporated into the library file being created. The directives also cause entries to be made in the file directory and deck list. UPDATE will identify and sequentially number each card image written to the library file with identifiers in the form ident.seqnum.

The following directives are used in a creation run:

**DECK dname*

This directive introduces a new deck for the library file being created; dname must not duplicate any dname previously introduced. A deck is terminated by the next occurrence of another *DECK or *COMDECK or an end- of-record mark on the input stream. File manipulation directives may appear in the body of the text deck, but they are not included in the card identification and numbering scheme. Directives introduced from the alternate input stream indicated by *READ are included in the card identification and numbering scheme. Output directives other than *DECK and *COMDECK may also be introduced and numbered. *END directives are ignored.

**COMDECK dname*

**COMDECK dname,NOPROP*

*COMDECK introduces a common deck in the input stream. The directive is subject to the same rules as *DECK as well as to the same identification and numbering scheme. In normal mode (Q and F not specified), UPDATE writes all modified decks to the compile file. Decks which call modified common decks are also considered to have been modified unless the common deck has the NOPROP option specified on its *COMDECK card.

LIBRARY FILE CREATION EXAMPLES

Example 1:

To create a program library called PL consisting of four decks, the first two written in COMPASS, the next two in FORTRAN), and to generate a compile file that would permit these decks to be processed by the assembler and compiler, the following deck structure is required:

```
(job card)
REQUEST,PL.
UPDATE(N=PL)
7/8/9
*DECK COMPGROUP
First COMPASS source deck
*DECK COMPGRP1
Second COMPASS source deck
*WEOR
*DECK FORTGROUP
First FORTRAN source deck
*DECK FORTGRP1
Second FORTRAN source deck
6/7/8/8
```

The compile file produced by this run would contain two logical records as a result of the *WEOR directive. Grouping of more than one source program written in a given language is permitted behind one *DECK directive. Such would be the case when modification of one program requires reassembly of all programs in the group.

Example 2:

```
(job card)                  Contents of REMTAPE
REQUEST(NEWPL)
REQUEST(REMTAPE)
UPDATE.                                                   *DECK A
7/8/9                                                     (Text of A)
*READ REMTAPE                                             *DECK B
*DECK LOCAL                                               (Text of B)
(Text of LOCAL)                                           *DECK C
6/7/8/9                                                   (Text of C)
                                                          6/7/8/9
```

Resulting library file contains decks A, B, C and LOCAL.

LIBRARY CORRECTION DIRECTIVES

The library correction process is the most common use of UPDATE. UPDATE accepts a correction set composed of UPDATE directives and data card images. The instructions provide UPDATE with instructions concerning the modification of the program library text stream. The old library is not disturbed. The corrections are permanent only in the new program library being created.

The following directives are used for correcting and updating a program library:

**IDENT idnam,B=num,K=identn,U=identn*

This directive introduces a correction set in the input stream. All corrective operations, except *PURGE, *SELPURGE, *PURDECK, and *ADDFILE, must occur after *IDENT. All card images in the correction set are identified by idnam as specified on this card. The identifier may consist of no more than 9 characters; any character with a display code octal value of less than 55 (space) is legal. The identifier must not be the same as any identifier currently in effect. This identifier remains in effect until the next *IDENT, *ADDFILE, *PURGE, *SELPURGE, or *PURDECK directive is encountered.

The correction set identifier idnam specified on this *IDENT card and all correction cards in this correction set may be subject to the following parameters.

B=num	Bias to be added to sequence numbers assigned to cards introduced by *IDENT.
K=identn	Identifier which must be currently known for UPDATE to process the correction set. If identn is not known to UPDATE, input will be skipped until UPDATE reads the next *IDENT directive.
U=identn	This parameter acts in the same manner as the K parameter except it specifies an identifier which must not be currently known by UPDATE for this correction set to be processed.

Any number of parameters may be specified on an IDENT card. They must be separated from idnam and each other by commas or space. No provision is made for continuation cards. If more than one B parameter is present, the last will be used. If more than one K or U parameter is present, all conditions must be true or the current correction set will be skipped.

Example:

```
*IDENT ZAP,B=100,K=ACE,U=NON,U=ARF
```

The bias of 100 will be added to all ZAP correction set card sequence numbers. If the idnam ACE is unknown or if either NON or ARF is known, the correction set ZAP will be skipped.

**PURGE identifiers*

*PURGE causes the complete and irreversible purging of a correction set and any modifications introduced by it. The directive may appear at any place in the input stream. *PURGE terminates any preceding correction set, therefore *INSERT and *DELETE may not follow a *PURGE until an *IDENT appears. The special deck *YANK$$$, which contains the *YANK cards, cannot be purged. A *PURGE card may be written in three forms.

**PURGE idnam1,idnam2,...*

The explicitly specified identifiers will be purged.

**PURGE idnam1.idnam2*

All correction sets in the directory from idnam to idnam2, inclusive, will be purged.

PURGE idnam,

The identified correction set idnam and all correction sets introduced subsequent to idnam will be purged.

Since all modifications are recorded in the library, this form is order-dependent and may be used to restore a program library to an earlier level. Purged items cannot be restored, however.

Permanent corrections such as those made by *PURGE, *SELPURGE, *PURDECK, or *SEQUENCE may never be reversed, therefore the last form may be used to restore the library to an earlier level only if no permanent correction has intervened.

LIBRARY FILE PURGING EXAMPLES

Example 1:

The correction sets BAD, WORSE, and WORST are no longer needed. The following job creates a new library (NEWPL) with those corrections removed.

```
(job card)
REQUEST(OLDPL)
REQUEST(NEWPL)
UPDATE(N,C=0,L=12)
7/8/9
*PURGE BAD,WORSE,WORST
6/7/8/9
```

Example 2:

A program library has been periodically modified over a number of months; at some point in time, it becomes desirable to step back permanently to a previous level. The following deck sequence illustrates this use of UPDATE:

```
(job card)
REQUEST,LIBAUG.
REQUEST,LIBMAY.
UPDATE(N=LIBMAY,P=LIBAUG,C=0)
7/8/9
*PURGE JUNMOD1,*
6/7/8/9
```

LIBAUG is the most recent (August) version of the program library. A new program library modified only through May is to be created. All modifications made after May begin at JUNMOD1 in the directory.

**SELPURGE dname.idnam*

The *SELPURGE card causes all cards in dname which belong to correction set idnam to be purged. It also nullifies the effects of idnam in deck dname. The idnam is not purged, therefore remains known to UPDATE. No deck other than dname is altered in any way.

**PURDECK dname,dname1,dname2,...*

*PURDECK causes all cards within the decks named to be purged, regardless of correction set identifiers. No identifiers are purged, therefore deck names purged as a result of *PURDECK may be reused as deck names.

**PURDECK dname2.dname4*

This form of *PURDECK causes all decks from dname2 to dname4 inclusive to be purged. Contents of the range dname2 to dname4 are determined by the order specified in the library file deck list.

Example:

Deck BAD on program library LIB is no longer of use and is to be removed permanently from the program library. The following deck sequence illustrates such permanent removal:

```
(job card)
REQUEST,LIB.                          Most recent program library
REQUEST,LESSBAD.                      New program library, with BAD
                                      purged, to be created by this
                                      run.
UPDATE(P=LIB,N=LESSBAD,C=0)
7/8/9
*PURDECK BAD                          Purges all cards within deck
                                      BAD.
6/7/8/9
```

The deck BAD is removed from the library. If BAD is also a correction set identifier name, that idnam would not be purged. *PURDECK operates so that any cards which have the identifier BAD but which are physically located outside of the deck BAD will not be purged. A *PURGE BAD directive should be added to the above run if the correction set identifier name is to be purged.

**DELETE identifiers*

Cards may be deactivated with the *DELETE card. A card image is marked as inactive; it remains in the library and retains its sequencing. It may be referenced in the same way as an active card. It is not, however, copied to the compile or source file.

**DELETE i1.n1*

**DELETE i1.n1,i2.n2*

i1,i2 = identifier name

n1,n2 = decimal sequence number

The first form of the directive specifies one card; the second, an inclusive range of cards. This range may include cards already deleted which are deleted again. In the second form, the i1.n1 card must occur before the i2.n2 card. In both forms the cards specifically referenced must exist. Text cards may be inserted after the last card deleted.

**RESTORE i1.n1*

**RESTORE i1.n1,i2.n2*

Specified cards are reactivated with the *RESTORE directive. The first form specifies one card; the second, an inclusive range of cards. Text cards may be inserted after the last card restored.

**INSERT i1.n1*

Cards may be inserted in a deck with the *INSERT directive. i1.n1 specifies the card after which the insertion is to be made.

**BEFORE i1.n1*

Cards may be inserted with the *BEFORE directive. Insertions will be made before the card specified by i1.n1.

**YANK a1,a2,a3,....an*

All effects of a correction set may be removed temporarily with the *YANK card. The correction set, identified by a1...an, must exist at the time *YANK is encountered. *YANK restores the library to the form it had before a correction set occurred. If a *YANK directive was included in the correction set, its effect is nullified. Insertions may not follow the *YANK directive.

Only *YANK, *YANKDECK, *SELYANK and *DEFINE directives should appear in the YANK$$$ deck. These directives are inserted automatically in the YANK$$$ deck when they are found in the input stream. UPDATE purges all other directives from YANK$$$ deck when E mode is selected.

Example:

To reverse the effect of a correction set temporarily, but not permanently, remove it from the program library LIB:

1. This change may be made temporarily for testing purposes:

```
(job card)
REQUEST,LIB.
UPDATE(P=LIB)
COMPASS(I=COMPILE)
7/8/9
*IDENT NEGATE
*YANK GOTTOGO
6/7/8/9
```

2. To put the preceding change onto a new program library as a random format permanent file:

```
(job card)
REQUEST,LIB.
REQUEST,NEWLIB,*PF.
UPDATE(P=LIB,N=NEWLIB)
CATALOG(NEWLIB,LIBCHG,RP=10,ID=BJMC)
COMPASS(I=COMPILE)
7/8/9
*IDENT NEGATE
*YANK GOTTOGO
6/7/8/9
```

The *YANK directive in the above example will become the first card on the new library NEWLIB. The identifier for this card will be NEGATE.1. The effects of *YANK may be nullified in future runs (and consequently the effects of GOTTOGO restored) by specifying:

```
*IDENT RESTOR
*DELETE NEGATE.1
```

or

```
*IDENT RESTOR
*YANK NEGATE
```

or

```
*PURGE NEGATE
```

If the correction set NEGATE contained other corrections as well as *YANK, the *YANK could be permanently removed by specifying:

```
*SELPURGE YANK$$$.NEGATE
```

It could be removed temporarily by

```
*SELYANK YANK$$$.NEGATE
```

**DO identifier*

**DONT identifier*

The *DO card may be used to negate the effect of a *YANK or *SELYANK. The *DONT directive restores the *YANK or *SELYANK. *DO and *DONT cards may be inserted at any place in the library in the same way any text card is inserted.

The *DONT directive may also be used to initiate a yank for a correction set at some place other than the beginning of the OLDPL. When a correction set has not been affected by *YANK, the appearance of a *DONT referring to that correction set will yank the correction cards until a *DO appears to terminate the *YANK function.

Example:

The correction set ACE, on the OLDPL, contains a directive *YANK DART. Within deck ZOTS on the OLDPL are correction cards introduced by correction set DART. To nullify the *YANK and thus reactivate cards belonging to DART which are between cards ZOTS.19 and ZOTS.244, and to leave all other cards belonging to DART yanked, the input stream should include:

```
*IDENT REST
*INSERT ZOTS.19
*DO DART
*INSERT ZOTS.244
*DONT DART
```

**SELYANK d1.i1,d2.i2,...dn.in*

dn = deck name
in = ident (correction set) name

The *SELYANK directive has the same effect as *YANK except the effect is limited to the deck specified on *SELYANK.

**YANKDECK dname1,dname2,...dnamen*

The *YANKDECK directive provides the facility to deactivate all cards within a specified deck. The effect on each individual card is the same as that caused by a YANK or SELYANK directive.

The significant difference between the YANK, SELYANK, and YANKDECK is in the grouping of cards which each card affects. YANK applies to all cards in a given correction set. SELYANK applies to all cards in a given correction set which are physically positioned within a given deck. YANKDECK applies to all cards within a given deck regardless of the correction set to which they belong.

As a means of comparing the effects of *YANK, *SELYANK and *YANKDECK, consider the following:

```
*YANK OLDMOD
```

This directive causes all effects of the correction set OLDMOD on the entire library to be nullified. Cards introduced by OLDMOD are deactivated; cards deactivated by OLDMOD are reactivated.

```
*SELYANK OLDDECK.OLDMOD
```

This directive accomplishes the same effect as the *YANK directive above except its effect is limited to cards within the deck OLDDECK.

```
*YANKDECK OLDDECK
```

This directive affects all cards in OLDDECK, without regard to which correction set they belong.

**/ comment*

Comments to be listed with a correction set may be included with a comment directive. It has a master control character * in column 1, a comment control character / in column 2, and a comma or blank in column 3. This card is ignored by UPDATE except that it is copied onto the listable output file. A comment may appear at any place in the input stream.

**ADDFILE fname,ident.seqnum*

**ADDFILE fname,dname*

fname = name of file from which information is to be read

ident.seqnum = card identifier after which decks are to be added

dname = deck name after which decks are to be added

When *ADDFILE is encountered, UPDATE reads creation control cards and text data from the named file, fname, and inserts this information after the deck (dname) or the card (ident.seqnum) on the new program library. The first card of fname must be *DECK or *COMDECK. UPDATE reads from this file until an end-of-record, which returns UPDATE to the main file. If the file fname on *ADDFILE is the main input file, cards are added until an end-of-record or the next UPDATE directive, whichever occurs first. UPDATE directives which are placed on the library (*DECK, *COMDECK, *CALL, *WEOR, *CWEOR, *TEXT, *IF, *ENDIF, *ENDTEXT) and file manipulation directives (*REWIND, *READ) do not terminate file adding from the INPUT file. The file specified on *ADDFILE is not rewound. File positioning may be done through the use of UPDATE file manipulation directives.

One or both of the parameters fname and dname or ident.seqnum may be omitted from *ADDFILE. If the parameter dname is absent, the *ADDFILE function will take place at the end of the library. If both parameters are missing, the *ADDFILE function will occur at the end of the library and will be taken from the main input file (by default, INPUT, or as specified by the I parameter). If only one parameter is present, it is assumed to be a file name.

**CALL dname*

dname = name of common deck

This directive is used to insert the text of a common deck, dname, onto the compile file. Common data, such as system symbol definitions, may be declared in the common deck and called for use in subsequent decks without repeating the data cards. The *CALL directive does not appear on the compile file; however, the contents of the common deck, excluding the *COMDECK directive, is copied at the point at which it is called.

SELECTIVE COMPILE OUTPUT

The selective compile output feature is handled by the directives described below. This feature is used in determining information to be written on the compile file. To control this process at the deck level, *COMPILE directives are introduced.

**COMPILE a,b,c,...,d*

**COMPILE a.d*

a, b, c, d = deck names

The *COMPILE directive specifies decks to be placed on the compile file for subsequent compilation or assembly. In the first form, specific decks are named. In the second form, a range of decks is specified. Decks written on the compile file include the two specified as well as all decks named between them in the list of deck names produced by UPDATE. All decks affected by these statements are written on the compile file regardless of the existence of any modifications within them. *COMPILE directives may appear anywhere in the input stream.

If the full assembly (F) feature is selected on the UPDATE call card, the *COMPILE cards are ignored.

SPECIAL DIRECTIVES

**COPY dname,idnam1.seqnum1,idnam2.seqnum2*

The *COPY directive allows the user to duplicate cards which already exist in the old library and to insert such duplication at any place in a new library or to copy it to an external file. Attempting to copy decks being introduced during the same UPDATE will produce an informative diagnostic.

All active cards are copied from idnam1.seqnum1 to idnam2.seqnum2, inclusive. All cards must be in the deck dname which must currently exist on the OLDPL. This form of *COPY must appear in a position within the input stream where text cards are normally allowed; therefore an *ADDFILE, *INSERT, *DELETE, *BEFORE, or *RESTORE directive must be in effect. The following input stream is valid since an *INSERT is in effect and the cards BDECK.4 through BDECK.8 will be copied and inserted after the text cards. The copied cards will be identified as part of correction set X.

```
*IDENT X
*INSERT BLAP.11
(text cards)
*COPY BDECK,BDECK.4,BDECK.8
```

The following text stream is not valid since an *INSERT is not in effect and UPDATE does not know where to put the card copies:

```
*IDENT X
*COPY BDECK,BDECK.4,BDECK.8
```

The third parameter, idnam2.seqnum2, may be omitted; in which case, only one card will be copied.

**COPY dname,idnam1.seqnum1,idnam2.seqnum2,fname*

The second form of *COPY is not restricted as described above. It may appear at any point in the input stream.

All cards to be copied must be within deck name dname. Cards are copied to the file fname; disposition of these copied cards is up to the user. The file fname will be a coded file and contain 80-column card images. It will contain one record for each *COPY directive. No sequencing information is appended.

**MOVE dname1,dname2*

The *MOVE directive enables the user to reorder decks while producing a new program library. The deck dname1 is moved from its present position on the old library and placed after dname2 on the new library. A *MOVE referencing a deck introduced in the same UPDATE run will produce an informative diagnostic.

**SEQUENCE dname*

**SEQUENCE dnam1.dname2*

**SEQUENCE dn1,dn2,dn3*

The *SEQUENCE directive allows selected routines to be resequenced on the new program library. All active cards in deck dname are resequenced under the identifier dname. All existing correction history bytes and all inactive cards are purged. In the second form of *SEQUENCE, all decks from dname1 to dname2 inclusive are resequenced. One or more decks may be specified on one *SEQUENCE directive if the deck names are separated by commas, as in the third form above.

Caution should be exercised in the use of the *SEQUENCE card. It is possible for cards to belong to a correction set which has the same name as the deck being resequenced. To prevent duplication of identifiers for cards outside the deck, UPDATE will purge any such cards.

**TEXT*

**ENDTEXT*

These directives allow the maintenance of data on the program library which UPDATE might otherwise confuse with control information. When UPDATE encounters *TEXT, that card and all cards following it (except other *TEXT, *ENDTEXT directives) up to and including the next *ENDTEXT directive are considered as data; and they are written to the program library. These cards are not checked or altered in any way. They are given an identifier and may be modified and corrected as any other UPDATE text. The first *TEXT directive in the input stream must be preceded by *IDENT on a correction run or by *DECK or *COMDECK on a creation run. The *TEXT and *ENDTEXT directives are maintained on the program library as text cards. These cards are not, however, written to the compile file.

**CHANGE id1,id2*

The *CHANGE directive affects the contents of the program library in only one respect. The name of the correction set identifier id1 in the file directory is changed to id2. Caution must be exercised because this change invalidates any *YANK or *SELYANK directives contained on the library which reference the changed correction set. *CHANGE directives are honored when they are encountered, so any subsequent references must be to the new correction set, not the old. *CHANGE directives may not be used to change deck names.

**DECLARE dname*

The *DECLARE directive provides a means of checking on the validity of UPDATE corrections. Once a deck name is specified in a *DECLARE directive, certain restrictions go into effect which are applied to all subsequent UPDATE corrections. These restrictions ensure that any following corrections apply only to the deck specified in the *DECLARE directive. UPDATE thus ensures that no other deck may be inadvertently altered. New decks inserted via the *ADDFILE directive need not be named in a *DECLARE directive. The restrictions are:

*PURGE and *YANK directives are not allowed.

All *INSERT, *DELETE, *RESTORE, and *BEFORE operations are marked as applying to the deck specified on *DECLARE. If one of these directives specifies a card not within the declared deck, a non-fatal error occurs, and the operation is not performed.

All UPDATE operations other than *ADDFILE will be restricted to the deck dname until another *DECLARE directive appears in the input stream. The DECLARE mode of operation may be terminated by the insertion of a *DECLARE directive with no deck name.

**ABBREV*

**NOABBREV*

All UPDATE control words may be written in abbreviated form. The abbreviations are expanded when they are read and appear as full words in all listings, on all UPDATE output, and in the program library. Since checking and expansion consumes time, a switch is provided to turn off abbreviation checking when no abbreviations are used. Checking is turned off by *NOABBREV and turned on by *ABBREV. Abbreviations recognized by UPDATE are listed in table 10-2. UPDATE will search for abbreviations unless and until a *NOABBREV directive is encountered.

**LIMIT n*

The line limit of the output file may be changed from its default value of 6000 lines with a *LIMIT directive. n is a decimal number.

**IF parameters*

The appearance of a group of text cards on the compile file can be governed by certain conditions as specified by parameters on the *IF directive:

Parameter	**Conditions**
*IF -DECK,name,n	*DECK name is not known
*IF DECK,name,n	*DECK name is known
*IF -IDENT,name,n	*IDENT name is not known
*IF IDENT,name,n	*IDENT name is known
*IF -DEF,name,n	Name does not appear in *DEFINE list
*IF DEF,name,n	Name does appear in *DEFINE list

If the condition specified on the *IF directive is not true, then the next n active cards will not be written to the COMPILE file. Inactive cards are not counted in determining n.

**ENDIF*

This directive may be used in place of the third *IF parameter to determine the end of the range of cards affected by the *IF directive. *IF and *ENDIF directives may be inserted at any point in the text stream. *IF directives affect only compile file output.

**DEFINE param1,param2,...paramn*

*DEFINE directives are used only to define parameters to be tested by *IF. Parameters defined do not relate to correction set or data names. *DEFINE cards are placed in the *YANK$$$ deck and may appear in the input stream any place a *YANK directive may appear.

**PULLMOD ident1,ident2,....identn*

This card is used to regenerate correction sets. If corrections were made to the library after the specified sets, the regenerated sets may not accurately reflect the original corrections. For instance, *PULLMOD cannot determine if cards have been purged. The user must determine whether or not irrecoverable changes occurred later than the specified correction sets.

EDIT AND MERGE OPTIONS

The editing (E) option causes all purged entries in the directory to be removed from the library. Idents not referenced are purged on the first pass. All remaining idents (including newly purged) are given new ordinals on the second pass. A second edit run removes all idents purged in the first edit run. All idents again receive new ordinals, reflecting their actual order on the program library.

With the merge (M) option, two files may be merged into one new program library. If M is not equated to a file name, the name MERGE is assumed. The P file (OLDPL) is the master file. To make them unique, duplicate deck and ident names on M are modified by appending an A or changing the last character.

LISTABLE OUTPUT FROM UPDATE

Listable output from an UPDATE run is governed by control card options L and O and the *LIST, *NOLIST, and *LIMIT cards. O option determines the file on which listable output is placed by UPDATE; default list file is OUTPUT. L option determines the type of output produced; L may be equated to a string of characters as listed below; default is L=A1234 for correction runs; L=A12 for creation runs.

Option Character	Output
A	List of known DECK or IDENT names, COMDECKs processed (subset of 1), DECKS written to COMPILE file, or known DEFINITIONS (*DEFINE card)
0	Suppresses all UPDATE listings
1	Error messages
2	Active control cards
3	All cards which changed status during this UPDATE
4	Cards encountered in input stream
5	All active *DECK, *COMDECK, *CWEOR, EOR, *CALL cards
6	Count of all cards and count of all active cards by deck name and by correction set name
7	All currently active cards
8	All currently inactive cards
9	Correction history information on all cards listed as a result of options 5, 7, 8
F	All options except 0

Option 4 produces a copy of the input stream on a correction run. A page eject occurs at each *IDENT. Valid control cards recognized by UPDATE are marked by five asterisks to the left. If some type of interpretation was performed by UPDATE (directive abbreviation expanded or default identifier inserted), asterisks are replaced by five slashes. *ERROR* appears to the left and right of erroneous cards. The input file name appears to the right of cards read as the result of a *READ directive.

Cards inserted by *ADDFILE are not listed if list option 4 is selected by default; but they are listed if option 4 is explicitly selected. Option 4 may be turned on by a *LIST card and off by a *NOLIST card. Option 3 produces a commentary on effective changes to cards processed. This commentary consists of card image, name of deck containing card identifier, and a key as shown below:

Key	Action
I	Card introduced during run
A	Inactive card on OLDPL became active
D	Active card on OLDPL became inactive
P	Card was purged during run
SEQ	Card was resequenced during run

When currently active cards are purged, the word ACTIVE occurs in addition to P.

Examples of list option 3 output:

Deck Name	Card Image	Identifier	Type of Change
UPDATE	BX6 X1	UPDATE.316	D
UPDATE	LX2 4	OLDID.2	P
UPDATE	SA1 B4	CORREC.1	I

The *LIMIT directive may be used to specify a limit to the number of lines produced as UPDATE listable ouput. When the defined value (or if not specified, the default value of 6000 lines) is reached, options 3 and 4 are turned off. Errors and control cards are still listed, however, if options 1 and 2 were selected. Options 5 to 9 are not affected.

List options 5-9 are provided for auditing an OLDPL. Output is written to a temporary file and appended to the output (O) file at the end of the UPDATE. When the F option is selected, options 5-9 apply to all decks on the OLDPL. If F is not selected, options 5-9 apply only to decks listed on *COMPILE cards. Cards listed option 7 are under marked to the right of the card by the letter A (active); cards listed under option 8 are marked by the letter I (inactive).

OVERLAPPING CORRECTIONS

UPDATE can detect four overlapping correction situations:

Type 1 Two or more modifications are made to one card by a single correction set.

Type 2 A modification attempts to activate an already active card.

Type 3 A modification attempts to deactivate an already inactive card.

Type 4 A card is inserted after a card which was inactive on the OLDPL.

When any of these types is detected, UPDATE will print the offending line with the words TP.n OVLP appended on the far right. The listing of overlap lines is controlled by list option 3. If any overlap condition is encountered during a run, this dayfile message is printed: (number) OVERLAPPING CORRECTIONS.

Detection of an overlap does not necessarily indicate a user error. Overlap messages are advisory, and point to conditions in which the probability of error is greater than normal.

Type TP.2 and TP.3 are detected by comparing existing correction history bytes with those to be added. Complex operations involving YANK and PURGE may generate these overlap messages even though no overlap occurs.

Modifications for each correction set are performed by UPDATE in the order in which sets are introduced. The order is irrelevant if no correction is dependent on another. If a dependent relationship exists, however, order is of paramount importance.

1. To create program library called NEWPL containing two program decks and two common decks:

```
(job card)
REQUEST,NEWPL.
UPDATE(N)
7/8/9
*COMDECK D1
.
.
.
*COMDECK D2
.
.
.
*DECK XA
.
.
.
*DECK XB
.
.
.
*CALL D2
6/7/8/9
```

   The COMPILE file that is produced by default contains decks XA and XB in that order. Deck XB has been expanded to contain common deck D2.

2. To modify a program library and produce an assembly listing:

```
(job card)
REQUEST,FN.                Old program library file FN was created in a prior run.
UPDATE (P=FN)
COMPASS (I=COMPILE)
7/8/9
*IDENT CS1
*INSERT XA.1
(Insertions)
*DELETE XA.20,XA.23
6/7/8/9
```

   The COMPILE file which is read by COMPASS will contain deck XA since that deck was modified by UPDATE.

3. To generate a new program library with corrections:

```
(job card)
REQUEST,FN1.
REQUEST,FN2.
UPDATE(P=FN1,N=FN2)
7/8/9
*IDENT XRAY
*DELETE D2.3,D2.5
*INSERT D2.8
(Insertions)
6/7/8/9
```

4. Example of Q option use. UPDATE places the deck dname in file FN1 on the file COMPILE. COMPASS reads deck dname from COMPILE file and assembles it.

```
(job card)
REQUEST,FN1.
UPDATE(Q,P=FN1)
COMPASS(I=COMPILE)
7/8/9
*IDENT YOKE
(Correction Set)
*COMPILE dname           Name of deck to be placed on COMPILE file for COMPASS assem-
                         bly.
6/7/8/9
```

5. To construct a new program library from the old program library and add a new routine, a new common deck, and a new SYSTEXT deck:

```
(job card)
REQUEST,OLDPL.
REQUEST,NEWPL.
UPDATE(N,C=0)
7/8/9
*ADDFILE INPUT,YANK$$$
*COMDECK D1A
.
.   (New common deck to be inserted as first deck on library
    file)
.
*ADDFILE INPUT
*DECK DDD
DDD                       Name card required by EDITLIB for system text
.
.   (SYSTEXT deck to be inserted as last deck on library file)
.
*ADDFILE INPUT,XB
*DECK XC
.
.   (New routine XC to be inserted following last card of routine
    XB)
.
6/7/8/9
```

No compile file is produced. The library file, NEWPL, will contain decks D1A, DDD and XC in addition to all decks which were on OLDPL.

6. Simplified example for generating a program library:

Contents of File A1	**Identifier given to card (does not appear on file A1):**
`  *COMDECK CSET`	CSET.1
`    COMMON A,B,C`	CSET.2
`  *DECK SET1`	SET1.1
`    PROGRAM ZIP`	SET1.2
`C   A DO- NOTHING JOB`	SET1.3
`    END`	SET1.4
`  *DECK SET2`	SET2.1
`    SUBROUTINE JIM`	SET2.2
`    A = B - SIN(C)`	SET2.3
`    END`	SET2.4
`  6/7/8/9`	

For the UPDATE task:

```
(job card)
REQUEST,A1.
REQUEST,NEWPL.
UPDATE(I=A1,N)
6/7/8/9
```

The contents of COMPILE file will contain two source decks whose cards are identified by their deck name and sequence number:

```
     PROGRAM ZIP              SET1.2
C    A DO-NOTHING JOB         SET1.3
     END                      SET1.4
     SUBROUTINE JIM           SET2.2
     A = B - SIN(C)           SET2.3
     END                      SET2.4
```

7. To alter the NEWPL file produced by example 6:

```
(job card)
REQUEST,OLDPL.            (OLDPL is file produced by example 6)
UPDATE.
7/8/9
*IDENT ADD1
*DELETE SET1.3
*CALL CSET
*INSERT SET1.2
  B=1.0
  C=3.14159
  CALL JIM
*COPY SET1,SET1.4
*COPY SET2,SET2.2
*CALL CSET
*COPY SET2,SET 2.3,SET2.4
6/7/8/9
```

Deck SET1 will appear on the COMPILE file as:

```
PROGRAM ZIP                   SET1.2
COMMON A,B,C                  CSET.2
B = 1.0                       ADD1.2
C = 3.1419                    ADD1.3
CALL JIM                      ADD1.4
END                           ADD1.5
SUBROUTINE JIM                ADD1.6
COMMON A,B,C                  CSET.2
A = B - SIN(C)                ADD1.7
END                           ADD1.8
```

STANDARD SCOPE CHARACTER SETS A

The character set selected when the system is installed should be compatible with the printers.

With an installation parameter, the installation keypunch format standard can be selected as 026 or 029; the installation parameter can also allow a user to override the standard; a user may select a keypunch mode for his input deck by punching 26 or 29 in columns 79 and 80 of his JOB card or any 7/8/9 end-of-record card. The mode remains set for the remainder of the job or until it is reset by a different mode selection on another 7/8/9 card.

CDC 63-CHARACTER SET

Display Code	Character	Hollerith (026)	Hollerith (029)	External BCD	Display Code	Character	Hollerith (026)	Hollerith (029)	External BCD
00	(none)†			16	40	5	5	5	05
01	A	12-1	12-1	61	41	6	6	6	06
02	B	12-2	12-2	62	42	7	7	7	07
03	C	12-3	12-3	63	43	8	8	8	10
04	D	12-4	12-4	64	44	9	9	9	11
05	E	12-5	12-5	65	45	+	12	12-8-6	60
06	F	12-6	12-6	66	46	–	11	11	40
07	G	12-7	12-7	67	47	*	11-8-4	11-8-4	54
10	H	12-8	12-8	70	50	/	0-1	0-1	21
11	I	12-9	12-9	71	51	(	0-8-4	12-8-5	34
12	J	11-1	11-1	41	52	)	12-8-4	11-8-5	74
13	K	11-2	11-2	42	53	$	11-8-3	11-8-3	53
14	L	11-3	11-3	43	54	=	8-3	8-6	13
15	M	11-4	11-4	44	55	blank	no punch	no punch	20
16	N	11-5	11-5	45	56	, (comma)	0-8-3	0-8-3	33
17	O	11-6	11-6	46	57	. (period)	12-8-3	12-8-3	73
20	P	11-7	11-7	47	60	≡	0-8-6	8-3	36
21	Q	11-8	11-8	50	61	[	8-7	8-5	17
22	R	11-9	11-9	51	62	]	0-8-2	12-8-7	32
23	S	0-2	0-2	22	63	:(colon)†	8-2	8-2	00*
24	T	0-3	0-3	23	64	≠	8-4	8-7	14
25	U	0-4	0-4	24	65	→	0-8-5	0-8-5	35
26	V	0-5	0-5	25	66	∨	11-0 or 11-8-2	11-0 or 11-8-2	52
27	W	0-6	0-6	26	67	∧	0-8-7	12	37
30	X	0-7	0-7	27	70	↑	11-8-5	8-4	55
31	Y	0-8	0-8	30	71	↓	11-8-6	0-8-7	56
32	Z	0-9	0-9	31	72	<	12-0 or 12-8-2	12-0 or 12-8-2	72
33	0	0	0	12	73	>	11-8-7	0-8-6	57
34	1	1	1	01	74	≤	8-5	12-8-4	15
35	2	2	2	02	75	≥	12-8-5	0-8-2	75
36	3	3	3	03	76	¬	12-8-6	11-8-7	76
37	4	4	4	04	77	;(semicolon)	12-8-7	11-8-6	77

†When the 63-Character Set is used, the punch code 8-2 is associated with display code 63, the colon. Display code 00_8 is not included in the 63-Character Set and is not associated with any card punch. The 8-6 card punch (026 keypunch) and the 0-8-4 card punch (029 keypunch) in the 63-Character Set are treated as blank on input.

*Since 00 cannot be represented on magnetic tape, it is converted to BCD 12. On input, it will be translated to display code 33 (number zero).

CDC 64-CHARACTER SET

Display Code	Character	Hollerith (026)	Hollerith (029)	External BCD	Display Code	Character	Hollerith (026)	Hollerith (029)	External BCD
00	: †	8-2	8-2	00*	40	5	5	5	05
01	A	12-1	12-1	61	41	6	6	6	06
02	B	12-2	12-2	62	42	7	7	7	07
03	C	12-3	12-3	63	43	8	8	8	10
04	D	12-4	12-4	64	44	9	9	9	11
05	E	12-5	12-5	65	45	+	12	12-8-6	60
06	F	12-6	12-6	66	46	–	11	11	40
07	G	12-7	12-7	67	47	*	11-8-4	11-8-4	54
10	H	12-8	12-8	70	50	/	0-1	0-1	21
11	I	12-9	12-9	71	51	(	0-8-4	12-8-5	34
12	J	11-1	11-1	41	52	)	12-8-4	11-8-5	74
13	K	11-2	11-2	42	53	$	11-8-3	11-8-3	53
14	L	11-3	11-3	43	54	=	8-3	8-6	13
15	M	11-4	11-4	44	55	blank	no punch	no punch	20
16	N	11-5	11-5	45	56	(comma)	0-8-3	0-8-3	33
17	O	11-6	11-6	46	57	(period)	12-8-3	12-8-3	73
20	P	11-7	11-7	47	60	≡	0-8-6	8-3	36
21	Q	11-8	11-8	50	61	[	8-7	8-5	17
22	R	11-9	11-9	51	62	]	0-8-2	12-8-7	32
23	S	0-2	0-2	22	63	%	8-6	0-8-4	16
24	T	0-3	0-3	23	64	≠	8-4	8-7	14
25	U	0-4	0-4	24	65	→	0-8-5	0-8-5	35
26	V	0-5	0-5	25	66	∨	11-0 or 11-8-2	11-0 or 11-8-2	52
27	W	0-6	0-6	26					
30	X	0-7	0-7	27	67	∧	0-8-7	12	37
31	Y	0-8	0-8	30	70	↑	11-8-5	8-4	55
32	Z	0-9	0-9	31	71	↓	11-8-6	0-8-7	56
33	0	0	0	12	72	<	12-0 or 12-8-2	12-0 or 12-8-2	72
34	1	1	1	01					
35	2	2	2	02					
36	3	3	3	03	73	>	11-8-7	0-8-6	57
37	4	4	4	04	74	≤	8-5	12-8-4	15
					75	≥	12-8-5	0-8-2	75
					76	¬	12-8-6	11-8-7	76
					77	;(semicolon)	12-8-7	11-8-6	77

†This character is lost on even parity magnetic tape.

*Since 00 cannot be represented on magnetic tape, it is converted to BCD 12. On input, it will be translated to display code 33 (number zero).

ASCII 64-CHARACTER SUBSET*

Display Code	Character	Hollerith (026)	Hollerith (029)	ASCII Code	Display Code	Character	Hollerith (026)	Hollerith (029)	ASCII Code
00	: †	8-2	8-2	072	40	5	5	5	065
01	A	12-1	12-1	101	41	6	6	6	066
02	B	12-2	12-2	102	42	7	7	7	067
03	C	12-3	12-3	103	43	8	8	8	070
04	D	12-4	12-4	104	44	9	9	9	071
05	E	12-5	12-5	105	45	+	12	12-8-6	053
06	F	12-6	12-6	106	46	–	11	11	055
07	G	12-7	12-7.	107	47	*	11-8-4	11-8-4	052
10	H	12-8	12-8	110	50	/	0-1	0-1	057
11	I	12-9	12-9	111	51	(	0-8-4	12-8-5	050
12	J	11-1	11-1	112	52	)	12-8-4	11-8-5	051
13	K	11-2	11-2	113	53	$	11-8-3	11-8-3	044
14	L	11-3	11-3	114	54	=	8-3	8-6	075
15	M	11-4	11-4	115	55	blank	no punch	no punch	040
16	N	11-5	11-5	116	56	, (comma)	0-8-3	0-8-3	054
17	O	11-6	11-6	117	57	. (period)	12-8-3	12-8-3	056
20	P	11-7	11-7	120	60	≠	0-8-6	8-3	043
21	Q	11-8	11-8	121	61	' (apostrophe)	8-7	8-5	047
22	R	11-9	11-9	122	62	!	0-8-2	12-8-7	041
23	S	0-2	0-2	123	63	%	8-6	0-8-4	045
24	T	0-3	0-3	124	64	" (quote)	8-4	8-7	042
25	U	0-4	0-4	125	65	_ (underline)	0-8-5	0-8-5	137
26	V	0-5	0-5	126	66	]	11-0 or 11-8-2	11-0 or 11-8-2	135
27	W	0-6	0-6	127					
30	X	0-7	0-7	130	67	&	0-8-7	12	046
31	Y	0-8	0-8	131	70	@	11-8-5	8-4	100
32	Z	0-9	0-9	132	71	?	11-8-6	0-8-7	077
33	0	0	0	060	72	[	12-0 or 12-8-2	12-0 or 12-8-2	133
34	1	1	1	061					
35	2	2	2	062	73	>	11-8-7	0-8-6	076
36	3	3	3	063	74	<	8-5	12-8-4	074
37	4	4	4	064	75	\	12-8-5	0-8-2	134
					76	^ (circumflex)	12-8-6	11-8-7	136
					77	; (semicolon)	12-8-7	11-8-6	073

†This character is lost on even parity magnetic tape.

*BCD representation is used when data is recorded on even parity magnetic tape. In this case, the octal BCD/display code correspondence is the same as for the CDC 64-character set.

CONTROL CARD SUMMARY B

All requests that can be issued to SCOPE on control cards are noted in the following summary, together with parameters issued as part of each request. In this summary, constants are capitalized and variables are in lower case; the variables are defined below the illustrated format. In the formats shown, commas are used as separators and periods are used as terminators in all cases.

Following the description of each request, the function executed by that request and the document or reference manual (RM) that contains a full discussion of that function are noted.

Card Format		Function	Reference Section
JOB IDENTIFICATION AND CONTROL CARDS			
job,Tt,CMa,ECb,Pp,Dym,MTn, NTk,TPk,CPcp.		Identifies job and specifies system requirements.	SCOPE 3.3 RM, 2
job	Job name		
t	CP time limit		
a	Central memory storage		
b	ECS field length		
p	Priority level		
y	Letters identifying dependent job string		
m	Number of jobs on which this depends		
n	Number of tape units required, MT or TP (7 track) NT (9 track)		
cp	Central processor type A or B		
RFL,nfl.		Redefines job field length	SCOPE 3.3 RM, 6
nfl	New field length		
TRANSF,lfn1,lfn2,...lfnn.		Decreases dependency count of named jobs	SCOPE 3.3 RM, 2
lfn	Name of next job for execution		

EQUIPMENT AND FILE ASSIGNMENT CARDS

Card Format		Function	Reference Section
REQUEST,lfn,dt,eq.		Requests assigment of device on which file resides or will be written	SCOPE 3.3 RM, 2
lfn	Logical file referenced		
dt	Device or device type; dt field may be expanded		
eq	Specific device (EST ordinal)		
RPACK,pname,N.		Assigns private disk pack on which new (N) output file is written	SCOPE 3.3 RM, 2
pname	Private pack family name		
RPACK,pname,E,vrno.		Assigns private disk pack which contains (E) input file	SCOPE 3.3 RM, 2
pname	Private pack family name		
vrno	Visual ID number		
REQUEST,lfn,PK,pname		Requests file be assigned to private pack	
pname	Pack family name		
REMOVE,lfn,pname.		Removes files from private pack	SCOPE 3.3 RM, 2
lfn	Name of file		
pname	Family pack name		

Card Format		Function	Reference Section

DISPOSE(lfn,x)
DISPOSE(lfn,x=ky)
		Releases files to output device	SCOPE 3.3 RM, 2

DISPOSE(lfn,*x=ky)		Dispose file at EOJ; ignores ky	SCOPE 3.3 RM, 2
lfn	File to be released		
x	Disposition code		
ky	k=C (central site) y must be 2-character device code (alphanumeric) defined by installation; k=I (INTERCOM) or E (EXPORT/IMPORT), y designates site or user ID		
RETURN,lfn1,lfn2,...,lfnn.		Releases files and associated devices from job	SCOPE 3.3 RM, 2
lfn	Logical file names		
UNLOAD,lfn1,lfn2,...,lfnn.		Same as above	
COMMON,lfn.		Declares a file common or attaches common file to job	SCOPE 3.3 RM, 2
lfn	Name of common file		
RELEASE,lfn.		Releases file from common status	SCOPE 3.3 RM, 2
lfn	Logical file name		

Card Format		Function	Reference Section
LOADER CONTROL CARDS			
LOAD,lfn.		Loads program on lfn into central memory	SCOPE 3.3 RM, 6
lfn	Logical file name		
LOADER,name.		Selects CP or PP loader	SCOPE 3.3 RM, 6
name	CPLOADR or PPLOADR		
MAP,p.		Selects map, no map, or partial map option	SCOPE 3.3 RM, 6
p	ON, OFF, or PART		
NOGO.		Completes loading and linking for program execution, but suspends execution	SCOPE 3.3 RM 6
OVERLAY,lfn,lev1,lev2,Cnnnnnn.		*Establishes overlay	SCOPE 3.3 RM, 6
lfn	File on which overlay is written		
lev1	Primary overlay number (octal)		
lev2	Secondary overlay number (octal)		
nnnnnn	Number of words from start of blank common where overlay loading starts		

Card Format		Function	Reference Section
REDUCE.		Reduces field length of job after loading	SCOPE 3.3 RM, 6
SECTION,sname,pn1,pn2,...pnn.		*Defines section within segment	SCOPE 3.3 RM, 6
sname	Name of section		
pn	Name of subprogram		
SEGZERO,sn,pn1,pn2,...pnn.		*Establishes main segment for loading	SCOPE 3.3 RM, 6
sn	Segment name		
pn	Subprograms or sections comprising main or zerolevel segment		
SEGMENT,sn,pn1,pn2,...pnn.		*Establishes segment for loading	SCOPE 3.3 RM, 6
sn	Segment name		
pn	Subprograms or sections comprising segment		

* OVERLAY, SECTION, SEGMENT and SEGZERO cards are loader directives placed within the program itself. They do not belong with the control cards in the first record of a job.

Card Format		Function	Reference Section
PROGRAM EXECUTION CARDS			
lfn,p1,p2,...pn.		Loads and executes program	SCOPE 3.3 RM, 6.2
lfn	Name of file containing program		
p	Parameters passed to program		
EXECUTE,lfn,p1,p2,...pn.		Completes loading and linking of elements for execution, then executes this program	SCOPE 3.3 RM, 6
lfn	Name of file containing program		
p	Parameters passed to program		
COMPILATION/ASSEMBLY CARDS			
RUN,cm,fl,bl,lfn1,lfn2,lfn3, lc,as,cs		Compiles FORTRAN program	FORTRAN 2.3 RM, 60174900, App. F
cm	List and execute option		
fl	Field length of object program		
bl	I/O buffer lengths		
lfn1	Compiler input file name		
lfn2	File to receive compiler output		
lfn3	Binary file name		
lc	Printed output line limit for object program		
as	ANSI/IO list format interaction option		
cs	Cross-reference listing option		

Card Format		Function	Reference Section
FTN,I=lfn1,B=lfn2,E=lfn3,L=lfn4.		Compiles FORTRAN Extended program	FORTRAN Ext. 2.0 RM, 60176600, App. H
lfn1,	Compiler input (I) file name		
lfn2	File to receive binary (B) output		
lfn3	Input file for UPDATE routine (E)		
lfn4	Listing option (L)		
COBOL,lfn1,lfn2,l,lfn3,ot,ov,lib cs,ts.		Compiles COBOL program	COBOL 3.0 RM 60253000, sec. 6
lfn1	File containing source input		
lfn2	File to receive binary output		
l	List option		
lfn3	File containing COBOL source library		
ot	Suppresses certain output from compilations used as subprograms with another compilation		
ov	Separates overlay segments from main programs		
lib	COBOL is to be added to system library by EDITLIB		
cs	Requests ANSI collating sequence		
ts	Requests tape sort		

Card Format		Function	Reference Section
ALGOL,lfn1,lfn2,ob,lfn3,l,p,a.		Compiles ALGOL program	ALGOL 2 RM 60214900, sec . 5
lfn1	Source input file		
lfn2	Source input list option		
ob	Object program format		
lfn3	Supplementary input		
l	Assembly language list option		
p	Punch option		
a	Suppresses array bounds checking		
COMPASS,L=lfn1,I=lfn2,B=lfn3,S=lfn4.		Compiles COMPASS programs	COMPASS RM 60190900, sec. 8
lfn1	File to receive program listing		
lfn2	File containing source program to be assembled		
lfn3	File to receive binary output		
lfn4	Systems text overlay name		
SIMS,I=lfn1,L=lfn2,C=lfn3.		Compiles SIMSCRIPT program	SIMSCRIPT 2.0 RM 60178300, sec. 10
lfn1	Input file containing source program		
lfn2	File to receive listing of source program		
lfn3	File to receive COMPASS coding		

Card Format	Function	Reference Section
SIMULA,I=lfn1,L=lfn2,X=lfn3, P=lfn4,A=lfn5,B=lfn6. lfn1 File containing source program to be compiled; standard file is INPUT lfn2 File to receive source program listing; standard file is OUTPUT lfn3 File containing binary program in load-and-go form; standard file is LGO lfn4 File containing binary program in format for punched output; standard file is PUNCHB lfn5 File to receive object program in assembly language format for printing; standard file is OUTPUT lfn6 File to receive object program in assembly language format for punched cards; standard file is PUNCH	Compiles SIMULA program	SIMULA RM 60234800, sec. 7
SORTMRG.	Calls SORT/MERGE	SORT/MERGE 3 RM 60252600, sec. 6
PERT66.	Calls PERT/TIME	PERT/TIME RM 60133600, sec. 5

Card Format	Function	Reference Section
APT	Calls APT system	APT RM 60174500, sec. 11
BASIC,I=lfn1,L=lfn2, k=lfn3,B=lfn4. lfn1 Input file lfn2 Output file for printing at terminal lfn3 File to receive error listing and execution output lfn4 Object program file	Compiles and executes BASIC program	BASIC 2 RM 60306200, sec. 7
LIMIT,n. n Decimal number of 4096-word CM (60-bit) word groups	Limits amount of mass storage assigned to job	SCOPE 3.3 RM, 2

PROGRAMMING OPTION CARDS

Card Format	Function	Reference Section
SWITCH,n. n Sense switch number	Sets switch to on or off	SCOPE 3.3 RM, 2
MODE,n. n Type of halt	Defines halt conditions	SCOPE 3.3 RM, 2
EXIT	Establishes exit path in event of selected errors	SCOPE 3.3 RM, 2
EXIT(S)	Establishes exit path after control card or assembly errors	SCOPE 3.3 RM, 2

Card Format		Function	Reference Section
COMMENT,n...n.		Inserts comments	
n	Comment characters		
CKP.		Establishes checkpoint	SCOPE 3.3 RM, 9
RESTART,lfn,n,Sxxx.		Restarts job at check-point	SCOPE 3.3 RM, 9
lfn	Name of checkpoint file		
n	Decimal number of checkpoint where job is to be restarted		
Sxxx	Decimal number of words (xxx) in smallest physical record (used only with S tape)		

PERMANENT FILE CARDS

Card Format		Function	Reference Section
ATTACH,lfn,pfn,ID=name,PW=list, CY=cy,PP=priv,SD=sd,MR=mr.		Assigns permanent file to job	SCOPE 3.3 RM, 5
lfn	Logical file name		
pfn	Permanent file name		
name	Creator identifier		
list	Passwords		
cy	Cycle number		
priv	Privacy procedure		
sd	Subdirectory		
mr	Multi-read access		

Card Format		Function	Reference Section
CATALOG,lfn,pfn,ID=name,PW=list, RP=ret,CY=cy,TK=tk,TD=rd,EX=ex, MD=md,CN=cn,PP=priv,SD=sd,RN=rn.		Makes mass storage file permanent	SCOPE 3.3, 5
lfn	Logical file name		
pfn	Permanent file name		
name	Creator identifier		
list	Password list		
ret	Retention period		
cy	Cycle number		
tk	Turnkey password		
rd	Read password		
ex	Extend password		
md	Modify password		
cn	Control password		
priv	Privacy procedure		
sd	Subdirectory		
rn	Automatic permanent file name generation		
EXTEND,lfn.		Makes permanent extension to permanent file	SCOPE 3.3 RM, 5
lfn	Logical file name		

Card Format	Function	Reference Section
PURGE,lfn,pfn,ID=name,PW=list, CY=cy,PP=priv,SD=sd.	Drops permanent file from system	SCOPE 3.3 RM, 5

lfn	Logical file name
pfn	Permanent file name
name	Creator identifier
list	Password list
cy	Cycle number
priv	Privacy procedure
sd	Subdirectory

Card Format	Function	Reference Section
RENAME,lfn,pfn,ID=name,RP=ret, CY=cy,TK=tk,RD=rd,EX=ex,MD=md,CN=cn.	Renames control information for permanent file	SCOPE 3.3 RM, 5

lfn	Logical file name
pfn	Permanent file name
name	Creator identifier
ret	Retention period
cy	Cycle number
tk	Turnkey password
rd	Read password
ex	Extend password
md	Modify password
cn	Control password

Card Format		Function	Reference Section
FILE/RECORD MANIPULATION CARDS			
BKSP,lfn,n.		Backspaces n logical files	SCOPE 3.3 RM, 8
lfn	Logical file name		
n	Number of files to be backspaced		
REWIND,lfn1,lfn2,...lfnn.		Rewinds files named	SCOPE 3.3 RM, 8
lfn	Logical file name		
SKIPB,lfn,n,lev,m.		Skips backward by n logical records	SCOPE 3.3 RM, 8
lfn	Logical file name		
n	Number of records to skip		
lev	Level number of records to skip		
m	B for binary files, C for coded files		
SKIPF,lfn,n,lev, m.		Skips file forward by n logical records	SCOPE 3.3 RM, 8
lfn	Logical file name		
n	Number of records at or greater than lev to skip		
lev	Level number of records to be skipped		
m	B for binary records; C for coded records		

COPYING, COMBINING, AND COMPARING CARDS

Card Format		Function	Reference Section
COMBINE,lfn1,lfn2,n.		Combines input file records into one logical record of level 0 on output file	SCOPE 3.3 RM, 8
lfn1	Input file		
lfn2	Output file		
n	Number of records		
COMPARE,lfn1,lfn2,n,l,e,r.		Compares pairs of files or records	SCOPE 3.3 RM, 8
lfn	Logical file name		
n	Number of records to compare		
l	Level number of records		
e	Number of non-comparable words to be written		
r	Number of records to be processed during comparison		
COPY,lfn1,lfn2.		Copies all files in a volume lfn 1 to lfn 2	SCOPE 3.3 RM, 8
lfn	Logical file name		
COPYBF,lfn1,lfn2,n.		Copies n binary files from lfn1 to lfn2	SCOPE 3.3 RM, 8
lfn	Logical file name		
n	Number of files		
COPYBR,lfn1,lfn2,n.		Copies n binary records from lfn1 to lfn2	SCOPE 3.3 RM, 8
lfn	Logical file name		
n	Number of records		

Card Format	Function	Reference Section
COPYCF,lfn1,lfn2,n. lfn Logical file name n Number of files	Copies n Hollerith or External BCD files from lfn1 to lfn2	SCOPE 3.3 RM, 8
COPYCR,lfn1,lfn2,n. lfn Logical file name n Number of records	Copies n Hollerith or External BCD records from lfn1 to lfn2	SCOPE 3.3 RM, 8
COPYBCD,lfn1,lfn2,n. lfn Logical file name n Number of records	Copies packed output files to tape for subsequent off-line listing	SCOPE 3.3 RM, 8
COPYL,lfn1,lfn2,lfn3,program. lfn1 Old set of records lfn2 New set of records lfn3 Updated set	Replaces routines in a file	SCOPE 3.3 RM, 8
COPYN,p,out,in1,in2,...in10. p Record format out Output file in Input file	Input from up to 10 binary files copied to output file (Input directive record required)	SCOPE 3.3 RM, 8
COPYSBF,lfn1,lfn2. lfn1 Input file lfn2 Output file	Copies lfn1 to lfn2 formatting binary file for single space printing	SCOPE 3.3 RM, 9

Card Format	Function	Reference Section
COPYXS,xlfn,scplfn,n.	Converts binary X tapes to S tape format	SCOPE 3.3 RM, 8

xlfn	Input X tape must be requested in S format
scplfn	Output tape must be requested in S format
n	Number of files

LABELING CARD

Card Format	Function	Reference Section
LABEL,lfn,$\left\{\begin{matrix}R\\W\end{matrix}\right\}$,Y,L=fnl,V=reel,E=ed, T=ret,C=create,F=df,M=mfn,P=pos, D=den,N=char set, X=dc, VSN=vsn.	Writes or checks labels on tape file	SCOPE 3.3 RM, 8

lfn	Logical file name
R	Read file
W	Write file
fnl	File label name
reel	Reel number
ed	Edition number
ret	Retention period
create	Creation date
df	Data format
mfn	Multifile name
pos	Number of referenced file in a multifile volume
den	Density of data on tape
char set	Character set for coded data conversion on 9-track tape
dc	Disposition code
vsn	Volume serial number

Card Format	Function	Reference Section
FILE EDITING CARDS		
EDITSYM,I=lfn1,C=lfn2,L=lfn3, OPL=lfn4,NPL=lfn5. lfn1 Input file lfn2 Compile output file lfn3 List file lfn4 Old program library lfn5 New program library	Calls EDITSYM to produce program library file for FORTRAN, COMPASS, or other processor	SCOPE 3.3 RM, App. E
UPDATE,ident=lfn1,...,identn=lfnn, list. ident Type of file lfn Logical file name list Optional parameters specify update modes, comments, rewind, format, etc.	Calls UPDATE to perform program library maintenance (directive record required)	SCOPE 3.3 RM,10

Card Format		Function	Reference Section
PROGRAM DEBUGGING			
DEBUG,p.		Used with SNAP or TRACE	SCOPE 3.3 RM, 7
p	Specifies dump produced and debugging routines to be loaded with overlay		
DMP,x,y.		Dumps specified area of central memory	SCOPE 3.3 RM, 7
x	Beginning address in dump		
y	Last address in dump		
DMPECS,x,y,f,lfn.		Dumps specified area of ECS	SCOPE 3.3 RM, 7
x	Beginning dump address		
y	Last dump address		
f	Print format		
lfn	File to receive dump		

Card Format		Function	Reference Section
SNAP,ID=id,IA=ia,FWA=fwa,LWA=lwa, INT=int, F=f,FN=n,UR=ur.		Periodic snapshot dumps of registers and CM areas areas when ia executes	SCOPE 3.3 RM, 7
id	SNAP identifier		
ia	Address of trap		
fwa	Address of first word dumped		
lwa	Address of last word dumped		
int	Interval between words dumped		
f	Format of dump		
n	Frequency of dump		
ur	User's entry point		
TRACE,ID=id,IA=ia,LA=la,FN=n, TR=tr,TM=tm,TL=tl,RD=rd,OL=ol, OR=or.		Dumps registers and CM area when register tr or location tl is referenced in range ia-la	SCOPE 3.3 RM, 7
id	Trace identifier		
ia	Initial address of range		
la	Last address of range		
n	Frequency		
tr	Register trigger		
tm	Masking trigger		
tl	Location trigger		
rd	Register dump		
ol	Storage location		
or	References		

TAPE AND DISK LABELS C

MAGNETIC TAPE LABELS

ANSI LABELS

ANSI (American National Standards Institute previously known as USASI) labels for 7-track 1/2 inch magnetic tape are recorded in BCD mode at a density defined by the installation parameter IP.LDEN. For 9-track 1/2 inch magnetic tape, ANSI labels are recorded in odd parity at the density specified for the file. Both 7- and 9-track tape labels are 80 characters.

The SCOPE standard U labels, as described, are designed to conform to the proposed USA Standard for Magnetic Tape Labels and File Structure for Information Interchange submitted by the X.3.2/457 Committee on November 28, 1966, revision of X.3.2/513, July 18, 1968.

In this appendix, n means any numeric digit, 0-9; a means any of the 6-bit characters of the set in Appendix A.

An optional field may contain the information described. If an optional field does not contain the designated information, it should contain blanks if alphanumeric, zeroes if numeric. Fields not described as optional are considered mandatory and must be written as specified.

Four U labels are generated and processed by SCOPE: VOL1, HDR1, EOV1, EOF1. File Header Labels (HDR2 through HDR9) and User File Header Labels (UHLa) are allowed on input tapes and will be skipped by SCOPE. The following labels are not allowed: UVL1 through UVL9, EOV2 through EOV9, UTLa, EOF2 through EOF9.

The data block count includes neither label records not tape marks that are part of a label group. Tape marks written by an explicit WRITEF request are included in the count.

VOLUME HEADER LABEL (VOL1)

Field	Name	Length	Position	Description
1	Label identifier	3	1-3	Must be VOL
2	Label number	1	4	Must be 1
3	Visual Reel number	6	5-10	Six n characters
4	Security	1	11	Not processed by SCOPE
5	Volume density	1	12	Density of file information on tape blank or 00 = 556 bpi 1 = 200 bpi 2 = 800 bpi
6	Reserved for operating system	19	13-31	Must be blank
7	Reserved for future standardization	49	32-80	Must be blank

FILE HEADER LABEL (HDR1)

Field	Name	Length	Position	Description
1	Label identifier	3	1-3	Must be HDR
2	Label number	1	4	Must be 1
3	File label name	17	5-21	Any a characters to identify this file
4	Multi-file identifier	6	22-27	Any a characters to identify the set of files that includes this one. This ID must be the same for all files of a multi-file set (mfn)
5	Reel number	4	28-31	4 n characters incremented by one immediately after trailer label is written on the volume
6	Multi-file position-number	4	32-35	4 n characters denoting position number of file within a set. SCOPE checks only the low order 3 digits.
7	Reserved for future standardization	4	36-39	Must be blank
8	Edition number	2	40-41	2 n characters distinguishing sucessive iterations of same file
9	Reserved for future standardization	1	42	Must be blank
10	Creation date	5	43-47	Date file was created; YYDDD, 2 n characters for year and 3 n characters for Julian date (001 to 366)
11	Reserved for future standardization	1	48	Must be space

Field	Name	Length	Position	Description
12	Expiration date	5	49-53	Same format as field 10. File is regarded as expired when current date is equal to or later than the date in this field. When this condition is satisfied, the remainder of this volume may be overwritten. To be effective on multifile volumes, the expiration date of a file must be less than or equal to the expiration date of all previous files on the volume.
13	Security	1	54	Same as field 4 of the Volume Header Label; not checked by SCOPE.
14	Block count	6	55-60	Must be zeros
15	Reserved for future use	20	61-80	Must be spaces

FILE TRAILER LABEL (EOF1)

1	Label identifier	3	1-3	Must be EOF
2	Label number	1	4	Must be 1
3-13	Same as corresponding fields in file header (optional)	50	5-54	Same as corresponding fields in file header
14	Block count	6	55-60	Six n characters, number of data blocks written since last file header label
15	Reserved for future use	20	61-80	Must be blank

VOLUME TRAILER LABEL (EOV1)

Field	Name	Length	Position	Description
1	Label identifier	3	1-3	Must be EOV
2	Label number	1	4	Must be 1
3-13	Same as corresponding fields in file header (optional)	50	5-54	Same as corresponding field in file header
14	Block count	6	55-60	6 n characters, number of data blocks written since preceding volume label
15	Reserved for future use	20	61-80	Must be blanks

The volume trailer label format is identical to file trailer label format except for the third character.

The structure of a U-labeled tape file created by SCOPE is shown below. Required labels are indicated by a 4-character identifier; tape marks are indicated by an asterisk.

Single Reel File

VOL1 HDR1*...Data Blocks...*EOF1**

Multireel File

VOL1 HDR1*...First Volume Data...*EOV1**

VOL1 HDR1*...Last Volume Data...*EOF1**

Multifile Reel

VOL1 HDR1*...File A...*EOF1*HDR1*...*EOF1**

Multireel Multifile

VOL1 HDR1*...File A...*EOF1*HDR1*...File B...*EOV1**

VOL1 HDR1*...Continuation of File B...*EOV1**

VOL1 HDR1*...Last of File B...*EOF1*HDR1*...File C...*EOF1**

Special case end-of-tape conditions may occur when the reflective spot is encountered during the writing of a label group on a multifile set tape. Depending on exactly where the reflective spot appears, one of the following configurations will be generated.

```
1.              ...FILE A...EOV1**

     VOL HDR1**EOF1*HDR1*...File B...
            (A)     (A)   (B)

2.              ...File A...*EOF1*HDR1**EOV1**
                             (A)   (B)    (B)
     VOL HDR1*...File B...
           (B)

3.              ...File A...*EOF1*EOV1**
                             (A)
     VOL HDR1*...File B...
            (B)
```

The EOV1 belongs to neither file and serves only to indicate that the set is continued on another reel. The third configuration is allowed by Appendix A2.6 of the ANSI specification.

3000 SERIES COMPUTER LABELS

Three Y labels are generated and processed by SCOPE: file header, EOF, and EOT.

FILE HEADER

Field	Name	Length	Position	Description
1	Density	1	1	Density at which label and data are recorded. 2 = 200 bpi 5 = 556 bpi 8 = 800 bpi 7-track 8 = 1600 bpi 9-track
2	Label ID	2	2-3	( )
3	Logical Unit Number	2	4-5	Logical unit number; not checked by SCOPE.
4	Retention Cycle	3	6-8	3 n characters specifying number of days tape is to be protected from accidental destruction
5	File Name	14	9-22	Any a characters to identify this file
6	Reel Number	2	23-24	2 n characters to denote the particular reel in a series comprising the file

Field	Name	Length	Position	Description
7	Date	6	25-30	Date file was created; MMDDYY
8	Edition	2	31-32	2 n characters distinguishing successive iterations of the same file
9	User Information	48	33-80	Any a characters; SCOPE does not check this field

FILE TRAILER (EOF)

Field	Name	Length	Position	Description
1	Label ID	3	1-3	EOF
2	Block Count	5	4-8	5 n characters, number of data blocks written since file header label
3	Unused	72	9-80	Unused

VOLUME TRAILER (EOT)

Field	Name	Length	Position	Description
1	Label ID	3	1-3	EOT
2	Block Count	5	4-8	5 n characters, number of data blocks written since file header label
3	Unused	72	9-80	Unused

DISK PACK LABEL FORMAT

The first two PRU's of a disk pack (64 60-bit words each) contain its basic label, formatted as follows. The words of each PRU are numbered 0-63; the bytes of each word are numbered 0-4, from left to right; the bits of a whole word or of a byte are numbered 0-59 or 0-11, from right to left.

PRU 0	word 0	bits 36-59	DEV1
		bits 30-35	Binary zero
		bits 0-29	Julian date when label was written, yyddd in display code
	word 1		Visual ID number of pack, 1 to 6 characters, right justified with 4 to 9 display code zeros as padding.

word 2		Binary zero
word 3		Binary zero for a blank-labeled pack. For a private pack:
	bits 18-59	The logical name of the pack family, 1 to 7 characters left justified with binary zero padding.
	bits 12-17	Random bits
	bits 6-11	Family number of the pack, in binary, between 0 and 4. The packs in a family are numbered as they are added to the family in response to RPACK(pname,N) calls. Maximum number of packs in a family is 5.
	bits 0-5	Binary zero unless the family number is 0, in which case this field contains the number of files in the family, in binary, between 0 and 77 octal.
words 4-6		Binary zero
word 7	bytes 0-3	Binary zero
	byte 4	Checksum of the other 319 bytes in PRU, formed as if by addition in a 12-bit accumulator with end-around carry.
words 8-9		RBR header for the pack. The only significant fields are in word 9:
	bits 54-59	Half the length of the RBR body, in CM words, as a binary number.
	byte 4	Total number of physically available record blocks in pack. (These two fields are not copied into the RBR header in CMR when the pack is attached to the system, but they serve as guides in formatting the label.)
	byte 3	Number of logically available record blocks in the pack, which is the count of zero bits in the body of the RBR. This number is copied into the corresponding position of the RBR header in CMR when the pack is attached to the system. For a blank-labeled pack, or a private pack that is not number 0 in its family, this number is one less than the number in byte 4, indicating that only one record block is reserved for the label. For pack number 0 in a private pack family, it is n less than the number in byte 4, where n is the greatest number of record blocks that the label could occupy.

The RBR body begins at word 10 of PRU 0, and occupies the minimum number of words to furnish one bit for every physically available record block, according to the count in byte 4 of word 9. If this count is not exactly divisible by 60, the surplus bits at the end of the last word are set to 1, indicating non-existent record blocks. At the beginning of word 10, n bits are set to 1 to indicate record blocks reserved for the label and unavailable for file writing. n is the difference between the numbers in bytes 4 and 3 of word 9.

This RBR body is copied into the proper RBR area in CMR when the pack is attached to the system.

If the RBR body is longer than 54 CM words, it extends into the beginning of PRU 1. Otherwise, any words in PRU 0 after its conclusion contain binary zero.

PRU 1

As explained above, if the RBR body is longer than 54 words, its 55th and following words are found in PRU 1. Apart from this, word 0-57 of PRU 1 contain binary zero.

For a blank-labeled pack, the remainder of PRU 1 contains binary zero, except that byte 4 of word 63 contains a checksum of all the other bytes in the PRU, formed by adding as if in a 12-bit accumulator with end-around carry.

In a private pack, words 58-62 contain a list of visual ID numbers of packs in its family. Pack number 0 (family serial number, not visual ID NUMBER) is noted in word 58, pack number 1 (if any), in word 59, and so on. The list is terminated by the first word, between words 60 and 63 inclusive, whose byte 0 contains binary zero. Everything after that zero byte, and before byte 4 of word 63, contains random bits. Byte 4 of word 63 contains the checksum of the PRU, as described in the preceding paragraph.

Each of the words containing the visual ID number of a pack in the family is formatted:

bits 54-59	Serial number of the pack in its family, a binary number between 0 and 4.
bits 48-53	Random bits
bits 36-47	Number of files in the family if bits 54-59 contain 0; otherwise random bits.
bits 0-35	Visual ID number of pack in the family, 1 to 6 alphanumeric characters right justified with 5 to 0 display code zeros as filler.

PRU 2 and following

For any pack that is not member number 0 of a private pack family, the rest of record block 1 is ignored, and the space available for files begins at PRU 0 of record block 2. For member 0 of a private pack family, the FNT entry model and the RBT chain for all files in the family are stored beginning with PRU 2 of record block 1, and on through as many record blocks as necessary. The total number of record blocks reserved for this purpose is indicated, as described above, in words 9 and 10 of PRU 0, record block 1.

The information for each file begins with a two-word FNT entry model as follows:

First word:	bits 18-59	File name, left justified with binary zero filler
	bits 12-17	Random bits
	bits 0-11	Binary zero
Second word:	bits 48-59	0700 octal
	bits 36-47	Binary zero if file is empty and has no RBT chain; otherwise non-zero but the exact value is not significant.
	bits 0-35	Not significant.

If the file is not empty, its RBT chain, consisting of word pairs, follows. These word pairs have the same form they had in the system RBT just before the pack family was detached from the system, and that they will have in the system RBT the next time it is attached to the system, except:

> Byte 0 of the first word of the last pair contains binary zero, to indicate the end of the chain for the file. Byte 0 of the first word of every other pair contains binary 1 for the first pair, 2 for the second, and so on.
>
> Bits 3-11 of byte 1 of the first word of each pair contains, not the RBR ordinal that was shown in the RBT when the pack was attached to the system, but the serial number, within the family, of the pack on which all the record blocks indicated in this word pair are to be found.

No special indication appears after the information for the last file of the pack family. The fact that it is the last file is determined from the file count in bits 0-5 of word 3, PRU 0, record block 1 of the same label.

DIAGNOSTIC MESSAGES D

Messages that the SCOPE 3.3 Operating System prints are listed below.

Messages are listed in alphabetical order. Items in which the first characters will change according to the parameters of the job in progress are arranged according to the second word of the message. Items beginning with numbers follow the alphabetical lists, and those beginning with asterisks appear last. The routine that produces each message is listed on the right margin.

Abbreviations that commonly appear in this section:

CH	Channel	FNT/FST	File name/status table
CM	Central memory	MLRS	Maximum logical record size
CMR	Central memory resident	MT	7-track magnetic tape
CP	Central processor unit, or card punch	NT	9-track magnetic tape
		PF	Permanent file
ECS	Extended core storage	PFD	Permanent file directory
EOF	End-of-file	PP	Peripheral processor unit
EOI	End-of-information	PRU	Physical record unit
EQ	Equipment	RBR	Record block reservation table
EST	Equipment status table	RBT	Record block table
FET	File environment table	RBTC	Record block table catalog
FL	Field length	UBC	Unused bit count

Messages beginning with MT or NT are concerned with tape reliability. If the word RVD appears, the condition indicated has been recovered successfully. If ERR appears, the condition was not successfully recovered. In these instances the operator can type RECHECK, and the system will attempt recovery once more. If the operator types GO, the system continues from the current tape position as if no error had occurred; but subsequent data may be bad.

Procedures that the system follows in attempting recovery depend on the installation choice of tape reliability options. They may involve creating intentional blocks of noise record length (less than 8 characters), different techniques in positioning to the last good record, and use of a controlled backspace hardware function.

A DOUBLE EOF WAS FOUND BEFORE A /   COPYN

A NUMERIC EXTENDS BEYOND AN END OF FILE   COPYN

A OPTION INVALID WITH RANDOM OLDPL OR SEQUENTIAL NEWPL   UPDATE

A PARAMETER BEGINS BEYOND AN EOF-EOF   COPYN

A PARAMETER IS GREATER THAN 7 CHARACTERS   COPYN

ABOVE IS ILL-FORMED AND IGNORED   EDITLIB

Preceded by control card in question. EDITLIB does not abort, but proceeds to the next control card. Possible reasons for rejection are:

Contains more than 30 elements (words and/or numbers).

First element is not an EDITLIB directive.

Any other element exceeds seven characters.

An element begins with two or more digits and contains a letter.

An element which should be a name is a number.

An element which should be a residence code is not CM, or DS.

On a SKIPB card, the file name is not followed by a number.

On a SKIPF card, the file name is followed by an asterisk or dash; it should be a name or number.

ADDRESS xxxxxxx IS UNDEFINED   TRACE

Output specification address xxxxxxx is unsatisfied, output for that specification is ignored.

AN ID(P1) IS REQUIRED ON ALL TEXT CARDS   COPYN

ARG ERROR   LOC

Last word address of area to be cleared by LOC (Load Octal Correction) control card equals or exceeds field length, or last word address is less than first word address; job terminated.

AUTO-RECALL ERROR   1EJ

Job terminated because completion bit was already set when job went into auto-recall.

AUTO-TAPE-ASSIGNMENT UNSUCCESSFUL   2SU,2TO

Fatal error, code 22. Automatic assignment of a labeled input tape was incorrectly requested. Either the OPEN function was not issued or it was issued without declaring label fields or automatic recall.

B OPTION INVALID WITH SEQUENTIAL OLDPL   UPDATE

BACKUP GO WHEN CORRECTED (RESTART)   RESTART

Checkpoint number specified exceeds that in first leader record on checkpoint tape. Dayfile message requires operator action; mount earlier checkpoint tape or rewind.

BAD COMPARE   COMPARE

Displayed on console, system and job dayfile if discrepancy occurs during COMPARE. If fatal, operator may drop job.

BAD FILE NAME ON REQUEST   5DA

File name on REQUEST or REMOVE control card has bad format. Job terminated.

BAD LABEL   1DA

Unrecognizable label on disk pack read in response to RPACK control card or a DEVADD type-in, or label with wrong name read in response to RPACK control card. Job terminates from DEVADD type-in error; otherwise system waits for operator to assign another disk pack drive, assign same drive with different pack, or drop job.

BAD NAME CHECK xxxxxxx   EDITLIB

Program name xxxxxxx from input file is not same as first program name on ADD, ADDBCD, ADDCOS, or ADDTEXT directive. If source is running system library, xxxxxxx program not found in directory.

BAD PACK NAME IN REQUEST   5DA,1DA

Name on RPACK or REQUEST control card has bad format; or REMOVE card file name on private pack differs from parameter. Job terminates.

BAD TEXT   LDR,2LA,LOADERE

Illegal TEXT table entry; relocation code is illegal or table length incorrect. Fatal error.

BCD RECORD FOR ADD IS ILLEGAL   EDITLIB

Either ADDBCD or ADDTEXT should be used with BCD records.

BINARY CARD READ AS FUNCTION CARD | EDITLIB

First record on input file must contain all, and only, directive cards for EDITLIB. Card is assumed binary if character other than letter, digit, blank, or - . , ( ) / * $ appears.

BINARY RECORD MISSING FROM INPUT | COPYN

Logical records requested from system INPUT file must begin with the next logical record on INPUT file.

BKSP HIT EOF | EDITLIB

Execution of SKIPB directive encountered end of file.

BLANK COMMON EXCEEDS AVAILABLE CORE, TRUNCATED | LOADER

During blank common allocation, no length can exceed FL or overlap 20-word LOADER residence. No reference is truncated, only the allocation for core map.

BLANK PFN | PFC,PFA

At least first two characters of permanent file name are blank.

BLANK TAPE READ | 1RT,1RS,1NR

Return dependent on EP bit setting. No error code returned. No data was received after a 1 second delay.

BUFFER ARGUMENT ERROR | CIO,1RT, 1RS,1NR

Fatal error, error code 22. Return dependent on EP bit setting. At least one pointer, FIRST, IN, OUT, or LIMIT has an illegal value.

BUFFER PARAMETER ERROR | 1SX

Error code 11. Return dependent on EP bit setting. Circular buffer pointers in FET fail to satisfy:
$0 \leq$ FIRST $<$ LIMIT $\leq$ FL, FIRST $\leq$ IN $<$ LIMIT, FIRST $\leq$ OUT $<$ LIMIT.

C OPTION DOES NOT START WITH C | LDR,2LA,LOADERE

C must be first character of OVERLAY control card option which allows user to designate how many words above blank common overlay should be loaded.

C OPTION NOT LAST PARAMETER | LDR,2LA,LOADERE

Termination character does not follow C option on OVERLAY card.

C OPTION USED ON OVERLAY (0,0)   LDR,2LA

This option cannot be used on level 0,0 overlay card.

CALL IGNORED, COMMON DECK NOT IN DICTIONARY   EDITSYM

*CALL,dn does not name a common deck.

CALLING JOB IS NOT DEPENDENT   JDP

Job calling TRANSR, TRANSC, or TRANSF does not have dependency identifier.

CALLING ERROR (CKP)   CKP

Bit 0 of parameter is not zero, RECALL bit is not set in call to CKP, or parameter list is outside field length. Fatal dayfile message, job terminated.

CANNOT CATALOG - LATEST INDEX NOT WRITTEN   PFC

Message is followed by logical file name.

CANNOT COMMON P.F.   1AJ, CTS

CANNOT COMMON NON-ALLOC. DEVICE   1AJ, CTS

CANNOT COMPLETE LOAD, JOB ABORTED   LOADER

Intended to correct fatal error conditions in control card mode when LOADER requests LDR to load library routines. Since such a request appears to LDR as a user request, 2LE, a fatal error message does not abort, but returns to LOADER. An error message from 2LE appears before this message.

CANNOT COMPLETE THIS OVERLAY, BAD INPUT   LOADER

During overlay generation, if LDR encounters difficulty in loading text or tables, it produces a message such as USER ERROR, BAD TEXT TABLE. When LOADER attempts to complete the overlay, the fatal error bit (set by LDR) is detected, this message is output, and job is terminated. Core map will contain last good overlay generated.

CANNOT DISPOSE NON ALLOC FILE   DSP

DISPOSE function illegal for non-allocatable file. DISPOSE will be ignored.

CANNOT DISPOSE NON-LOCAL FILE   DSP

DISPOSE function illegal for non-local file. DISPOSE will be ignored.

CANNOT DISPOSE PERMANENT FILE   DSP

DISPOSE function illegal for permanent file. DISPOSE will be ignored.

CANNOT DISPOSE THE INPUT FILE — DSP

DISPOSE function illegal for INPUT file. DISPOSE will be ignored.

CANNOT EXTEND LATEST INDEX NOT WRITTEN — PFE

Message is followed by logical file name.

CANNOT LOAD TEXT INTO xxxxxx — LDR

Overlay that first declared labeled common at xxxxxx has already been written out.

CANT ((ADDTEXT)) FROM ((SYSTEM)) — EDITLIB

ADDTEXT reads and formats records into an overlay for system file. This is not possible if SYSTEM is specified as the input file since these records are already in overlay format. A simple ADD can probably get relevant record from SYSTEM.

CANT ADD WITHOUT READYING — EDITLIB

ADD, ADDBCD, ADDCOS, or ADDTEXT directive cards cannot be executed by EDITLIB unless a file has been named by a preceding READY card with no intervening COMPLETE card.

CANT FIND DIRECTORY RECORDS ON INPUT FILE — EDITLIB

EDITLIB cannot locate records that comprise directory on system file (should be in two records immediately following DSD); presumably this is not a system file.

CANT FIND DSD — EDITLIB

EDITLIB did not find DSD record near beginning of file during execution of LIST card.

CANT GET PACK ASSIGNED — 1BT

Disk pack named in a BLANK type-in is elegible to be blank labeled, but monitor refuses to assign pack to control point. Report this message to a system analyst.

CANT LIST BETWEEN READY AND COMPLETE — EDITLIB

EDITLIB cannot list programs in a file while it or another file is being constructed (READY card executed more recently than COMPLETE card).

CANT MOVE WHILE READY PENDING — EDITLIB

After READY card is read, location of a program in running system cannot be changed by MOVE card before file named in READY is written out by COMPLETE card.

CANT TRANSFER WITHOUT READYING — EDITLIB

EDITLIB can execute TRANSFER card only after file is named by preceding READY card with no intervening COMPLETE card.

CARD CALL LOADER FATAL ERROR — LOADER

Execution attempt in response to control card call for loader cannot be carried out. File is not in format required for loading.

CARD ERROR, CANNOT FIND FILE NAME — LDR,2LA,LOADERE

A control card call for loader does not contain a file name, or the file name cannot be found.

CARD ERROR, FIELD LENGTH TOO SMALL — LDR,2LA,LOADERE

Control card call for loader specified inadequate field length.

CARDS MISSING FROM OBJECT DECK — LDR,2LA,LOADERE

Word count for loader table not satisfied or it exceeded 20(octal) in text table.

CATALOG ATTEMPT ON NON-LOCAL FILE — PFC

Message is followed by logical file name.

CHECKPOINT xxxx ON lfn — CKP

CIO CODE NOT DEFINED ON DEVICE — CIO

Function code in FET is not applicable to device containing file. Return dependent on EP bit setting fatal error code 22.

CKP REQUESTED — CKP

CKP has received control from a user call. Informative dayfile message.

COMMON DECK EDITING MUST PRECEDE TEXT EDITING — EDITSYM

*COMDECK control card occurred after text deck correction or a *DECK control card. EDITSYM run terminated.

COMMON SECTION TOO LARGE — EDITSYM

Space available for common decks has been exceeded.

COMPARISON ABANDONED BECAUSE OF E-O-R LEVEL DIFFERENCE AFTER RECORD n FILE x LEVEL p FILE y LEVEL q — COMPARE

Message appears on OUTPUT file and run ends when both records of nth pair do not terminate with same level end-of-record; level numbers are octal.

((COMPLETE)) FINDS REC.MSG. IN FILE SSSSSSS — EDITLIB

Fault in disk file used by EDITLIB; SSSSSSS is a local file which holds directory under construction.

CONFLICT IN RECORD n — COMPARE

nth pair of records are not identical, word-for-word. When one record is longer, a separate message appears. Depending on parameters, this message may be followed by a listing of some or all words which differ. For example: 0020,00000000000000000000/00000000000000000001. The 17th word(octal 20) of record n of first file named on COMPARE control card was 0 and the corresponding word in the second file named is 1. First word of record would be 00000. Words are printed as 20 octal digits each. Comparison continues after message.

CONFLICTING RECORD COUNT EXHAUSTED — COMPARE

Run ends if number specified in control card is such that comparison would be abandoned if a higher number of record pairs were in conflict. For instance, if CONFLICT IN RECORD n is written for five pairs of records and control card has no sixth parameter, run terminates after CONFLICT IN RECORD n has been written 30000 times.

CONTROL CARD ERROR — COMPARE

Console display, system and job dayfile. COMPARE control card contains fewer than 2 parameters or parameter required to be a number, implied by fewer than 2, contains a non-numerical character. Job terminated.

CONTROL CARD ERROR, NO CKPFILE (RESTART) — RESTART

RESTART card has two numbers instead of a number and file name. Job terminated.

CONTROL CARD ERROR, NUMBER ERR (RESTART) — RESTART

Checkpoint number must be unsigned decimal integer greater than zero. Job terminated.

CONTROL CARD ERROR, S PARAMETER (RESTART) — RESTART

S parameter on RESTART card must begin with the character S. Job terminated.

CONTROL CARD ERROR PARAM CNT (RESTART) RESTART

More than two parameters on RESTART card. Job terminated.

CONTROL CARD REWIND (INPUT) IS ILLEGAL COPYN

System INPUT cannot be rewound.

CONTROL POINT ERROR 1LT

Control point error detected following storage relocations.

COPY REQUESTED BUT NO OLD PROGRAM LIBRARY EDITSYM

*COPY read from correction input but no old program library requested on EDITSYM call card. Run terminated.

COPYL DID NOT FIND xxxxxxx. COPYL

COPYL DONE COPYL

COPYXS - CONTROL CARD ERROR. FILE COUNT NON-NUMERIC COPYXS

Illegal character encountered in a field that should contain a number. Job terminated.

COPYXS - INPUT AND OUTPUT TAPES NOT BOTH S FORMAT COPYXS

REQUEST card for both input and output tape must specify S format. Job terminated.

COPYXS - INSUFFICIENT FIELD LENGTH COPYXS

Minimum field length required is 12000 octal. Job terminated.

CP xxxxxx.xxx SEC 1EJ

Central processor running time for job.

CP 00000, 000 SEC 1AJ

CRnn CKSUM ERROR RC.xxxx, CD.yyyy 2RC

System and job dayfiles. User local file assigned to reader with nn EST ordinal contains binary card with bad checksum and no checksum suppress punch in row 4 of column 1. Job terminated. Record number is decimal xxxx, counting first record as 0000. Card number is decimal yyyy, counting first card of record as 0001.

CRnn FORMAT ERROR RC.xxxx, CD.yyyy 2RC

System and job dayfiles. User local file assigned to card reader with nn EST ordinal contains 7/9 card, presumably binary, with unrecognizable format, and not included in cards for 80-column binary reading. Job terminated. Record number is decimal xxxx, counting first record in the file as 0000; card number is decimal yyyy, counting first card record as 0001.

CRnn HOLL.CHECK RC.xxxx, CD.yyyy 2RC

System and job dayfiles. User local file assigned to card reader with nn EST ordinal contains a card without 7/9 in column 1, presumably Hollerith, with invalid punch combination in at least one column. It was read as a blank column. Record number is decimal xxxx, counting first record in file as 0000; card number is decimal yyyy, counting first card of record as 0001.

CRnn MODE CHANGE RC.xxxx, CD.yyyy 2RC

System and job dayfiles. User local file assigned to card reader with nn EST ordinal had mixed cards (binary, Hollerith, 80-column) in a single record. Record number is decimal xxxx, counting first record as 0000; first change of type occurred at card number yyyy decimal, counting first card as 0001.

CTS CALL ON PROTECTED FILE CTS

User tried to make COMMON a system file. Job terminated.

CYCLE HAS BEEN DUMPED PFA

Cycle of permanent file on preceding line was dumped by DUMPF routine but not released.

CYCLE INCOMPLETE PFA

Cycle indicated on preceding line is incomplete because another control point is cataloging it, or a job terminated while cataloging. For the latter, initial deadstart required to reload all files.

CYCLE NOT IN SYSTEM PFE,PFP,PFA

Requested cycle of permanent file specified in preceding line not presently cataloged.

CYCLE NUMBER REDEFINED PFR

CYCLE SUCCESSFULLY PURGED PFP

Requested cycle of permanent file specified in preceding line has been purged; other cycles still remain.

DATA EXCEEDS MLRS	1WS,1NW
Fatal error, code 22.	
DATA BLOCK TOO LONG	1RT,1RS,1NR
Fatal error, code 22.	
DAYFILE DUMPED	1DF
Response to n.DAYFILE,eq. type-in.	
DEBUG CARD OUT OF ORDER	1AJ
Issued if DEBUG card appears after TRACE or SNAP cards during load. Job terminated.	
DECK DOES NOT END WITH *END	EDITSYM
Text deck ended, but no *END card appeared. Run terminated.	
DECK NOT ON OLD PROGRAM LIBRARY	EDITSYM
Deck specified by *EDIT or *COPY card is not on old program library. Run terminated.	
DECK STRUCTURE CHANGED	UPDATE
DECK/COMDECK card was inserted, deleted, or purged.	
((DEL)) EXHAUSTS PNT BEFORE SATISFACTION	EDITLIB
DELETE (P1-P2) card used, but P2 parameter does not appear after P1 in program name table.	
DIGITS IS NOT OCTAL	LDR,2LA,LDQ
All parameters on OVERLAY card should be octal.	
DIRECTORY ALMOST FULL	LPF
Warning that permanent file directory is 80 percent full.	
DIRECTORY EMPTY	EDITLIB
A program named on an EDITLIB control card cannot be found in the directory model being worked on (presumably not the running system directory) because the model is empty.	
DIRECTORY UNDER CONSTRUCTION GETS TOO BIG - TOO MUCH CM RESIDENCE	EDITLIB
Directory to replace SCOPE system directory in CM exceeds EDITLIB buffer, 20000(octal) words. Too many programs are CM resident. Try EDITLIB again with LENGTH(n) as first control card; n is 3-9, for 30000(octal), 110000(octal) words.	

DISP CODE GT 2 CHARS	DISPOSE

The disposition code (x) on the DISPOSE card must be 1 or 2 characters. DISPOSE card is ignored.

DISP CODE NEEDED IF ID GIVEN	DSP

Disposition code must be present if an identifier is specified in a DISPOSE call; function is ignored.

DISP ID GT 2 CHARS	DISPOSE

Identifier on a DISPOSE card must be 1 or 2 characters. DISPOSE card ignored.

DISPOSE ARGS NOT IN FL	DSP

DISPOSE function will be ignored.

DISPOSE IGNORED, NO FILE -- xxxxxx	DSP

File to be disposed could not be found.

DMP - ARG OUTSIDE FL	DMP

The beginning, or x parameter, is not within job field length. No dump will result.

DMP - CANT FIND SYMBOLIC NAME	DMP

DEBUG card is either absent or ignored by system; or symbolic parameter on DMP control card is incorrect. No dump will result.

DMP - DEBUG FILE EMPTY - CANT LABEL DUMP	DMP

Job included a DEBUG card, but no DEBUG file was created, probably because of errors. Resulting dump will not be labeled.

DMP - DEBUG FILE NOT YET CREATED LABEL/CHANGE DUMP NOT POSSIBLE	DMP

No DEBUG file at this point so it cannot be dumped.

DMP - FWA EXCEEDS LWA	DMP

Beginning parameter exceeds ending parameter. No dump will result.

DMP - LWA EXCEEDS FL - FL SUBSTITUTED	DMP

Ending parameter (last-word-parameter) is greater than job FL. Field length is substituted and dump is made.

DMP - NO ABSOLUTE DUMPS UNLESS IP.DEBUG=1	DMP

For privacy reasons, installation parameter IP.DEBUG has been turned off (=0). No absolute dumps will be made, but relative dumps (within job field length) will be made as requested.

DMP - ONLY 2 ARGS PERMITTED W/O DEBUG CARD	DMP

DMP control card specifies 3 or more parameters and DEBUG card is absent or ignored by system. Dump will be made; but only first 2 parameters will be effective.

DMP - PARAMETER WD ADDRESS OUTSIDE FL	DMP

CP program called DMP and placed a CM address in the DMP PP input register. CM word addressed contains x and y parameters of area to be dumped; but the parameter word addressed is not within job field length. No dump will result.

DMP - SYMBOLIC ARG REQUIRES DEBUG CARD	DMP

Symbolic (non-numeric) parameter was specified on DMP control card but DEBUG control card is absent or ignored by the system. No DEBUG file has been created, and the symbolic parameter cannot be related to CM addresses. No dump will result.

DSD FOUND BEFORE THE END RECORD SPECIFIED	EDITLIB

EDITLIB TRANSFER card specifies first and last records of a group to be transferred from source file to new system file; but DSD (which must be the last record transferred to a new system file) occurred before the last-named record was reached.

DUP COMMON FILES OF xxxxxxx	1AJ

Local and common file in FNT named (xxxxxxx) same as on COMMON card. Job terminated with control card error.

DUPLICATE CYCLE	PFC

Cycle of permanent file specified on preceding line already exists.

DUPLICATE CYCLE NUMBERS REQUESTED	PFR
DUPLICATE ENTRY POINT xxxxxxx	LOADER
DUPLICATE FILE NAME (RESTART)	RESTART

Requested tape is already assigned to this control point. Job terminates.

DUPLICATE PACK NAME 1DA

Pack name on RPACK control card same as a pack name already at that control point. Job terminated.

E/I JOB CANNOT DISPOSE TO INTERCOM TERMINAL DSP

EXPORT/IMPORT user tried to DISPOSE a file to an INTERCOM terminal. DISPOSE function will be ignored.

E/I NOT ON THIS SYSTEM DSP

User tried to DISPOSE a file to an EXPORT/IMPORT terminal where EXPORT/IMPORT was not implemented in system. DISPOSE function will be ignored.

ECS PARITY ERROR AT xxxx 1EJ

ECS parity error during a system storage move terminated job. Job may be active at a control point or in job initialization phase requesting storage at a control point. xxxx is absolute ECS address/1000 (octal).

EDIT CONTROL CARD SET MUST BE FOLLOWED BY A *EDIT CARD EDITSYM

*INSERT, *DELETE, *ADD, *CANCEL, or *RESTORE control card sets follow *DECK, *COMDECK, *COPY, *WEOR instead of an *EDIT card. Run terminated.

EDITLIB-CTS FAULT IN INITIALIZING COMMON FILES EDITLIB

Program fault in EDITLIB, or CTS that is not expected.

EDITLIB PROG.FAULT IN SUBRT.MAKE EDITLIB

EDITLIB is trying to ADD a record which begins like a binary CP program, but is badly formatted. Such a binary program must be organized in tables, and EDITLIB checks organization to get entry point names. If record appears to end in middle of a table, this message is issued.

EDITLIB PROGRAM FAULT IN SUBRT.SQB EDITLIB

EDITLIB found a malformation in a directory it is manipulating; probably indicates a fault in EDITLIB itself. If EDITLIB is altering the running system, it may indicate that some other program defaced system directory.

EDITSYM ERRORS EDITSYM

Dayfile message.

EMPTY PROGRAM NAME TABLE EDITLIB

Processing of COMPLETE directive found an empty table.

END OF FILE IMPROPERLY READ ON INPUT FILE	EDITLIB

TRANSFER card directed records to be copied from input file to system file. End-of-file was read before all records were found.

END-OF-INFO ON FILE x AS RECORD y	COMPARE

Program compared y-1 record pairs from two files but end-of-information occurred on x file before specified number of pairs was reached. Written on job output file, and run terminated.

EOF/EOI ENCOUNTERED	COPYCR,COPYCF, COPYBR,COPYBF

End-of-file encountered before record count on control card was exhausted, or end-of-information encountered before file count was exhausted.

EOR APPEARS BEFORE *END IN *DECK ADDITION	EDITSYM

Run terminated.

EOR REQUESTED BUT COMPILE FILE NOT REQUESTED	EDITSYM

*WEOR read, but compile file not requested on EDITSYM call card. Run terminated.

ERROR CONDITION NOT CLEARED LAST REQUEST	CIO

Return dependent on EP bit setting. Error code 22. FET shows that an error occurred on the previous function. The program should have taken appropriate action and cleared the error status.

ERROR IN ABS OVERLAY FILE FORMAT	LDR,2LA,LOADERQ

LOADERQ is in absolute overlay mode, but identification code of subroutine being processed does not equal 50 octal.

ERROR IN PARAD (1RC)	1RC

Address of parameter list is outside field length. Fatal dayfile message; job terminated.

ERROR LOADING LIBRARY OVERLAY xxxxxxx	LDV

An unrecoverable parity occurred while loading an overlay in the system library from ECS.

ERROR LOADING xxx FROM ECS LOC yyyyyyy	MTR

An uncorrectable parity error occurred while loading a PP program or overlay xxx, from ECS starting at absolute address yyyyyyy (octal).

ERROR MODE = x, ADDRESS = xxxxxx — 1EJ

Program terminated because of address or operand error. If x = 0, program attempted to jump to location 0. See error mode numbers under MODE control card in JOB PROCESSING section.

ERROR ON TRACE OR SNAP CARD — DEBUG

Parameter errors on TRACE/SNAP card or no parameters. In latter case, job terminated.

ERROR ON TRACE xxxxxxx CARD — TRACE

Error on TRACE card with xxxxxxx ID.

FDB ADDRESS INVALID — PFC,PFA, PFP,PFE

FET OUTSIDE FL — CIO

All or part of file environment table outside user field length. Fatal error code 22.

FET TOO SHORT — 1RS,1WS, 1NR,1NW

Fatal error code 22.

FIELD GREATER THAN 80 CHARACTERS — LDR,2LA

Loader directive improperly implemented. LOADERQ does not recognize a binary table and assumes a loader directive even if it is not. Fatal error. Information sent to dayfile.

FIELD IS NON NUMERIC ILLEGAL TEXT CARD — COPYN

FIELD LENGTH NOT SUFFICIENT FOR OVERLAY GENERATION — LOADER

Field length must accommodate loader plus loader tables and core map. OVERLAY execution requires less.

FIELD LENGTH TOO SMALL — LOADERE

User field length cannot accommodate both user program and tables produced by CP Loader. Fatal error.

FIELD LENGTH TOO SMALL FOR ENTRY POINT TABLE — LOADERE

CP Loader calls LDQ to read entry point table and external reference table into field length when library programs are to be loaded. If EPT, but not ERT, is placed in field length, library loading is completed without using the ERT. A subsequent warning message ERT HAS NOT BEEN USED appears on MAP. If EPT cannot be read into field length, job terminates.

FILE ALREADY AT THIS CPT — PFA

FILE DEVICE NOT ALLOCATABLE — IORANDM

Console, system and job dayfiles. IORANDM was called to read or write beginning of random access record; but file is assigned to non-allocatable equipment. Job terminated.

FILE EXTENDED — PFE

Permanent file indicated on preceding line has been extended.

FILE HAS BEEN CATALOGUED AS
CYCLE XX,pfn IN SD XXX — PFC

Successful permanent file catalog, giving cycle, permanent file name, and subdirectory.

FILE HAS BEEN RENAMED AS xxxx — PFR

Information only. An existing file on an allocatable device, opened with the FET r bit set has not index. FET r bit is cleared and control is returned to user program.

FILE MAY NOT RESIDE ON DEVICE ASSIGNED — 3DO,6WM

Fatal error, code 22.

FILE NAME ERROR ON EDITSYM CALL CARD — EDITSYM

Parameter error on EDITSYM call card. The first character must be alphabetic. Job terminated.

FILE NAME NOT SPECIFIED — DISPOSE

File name must be specified on a DISPOSE card. DISPOSE card will be ignored.

FILE NAME GT 7 CHARS — DISPOSE

File name on DISPOSE card must not exceed 7 characters. DISPOSE card will be ignored.

FILE NAME ON UPDATE CARD GR 7 CHAR, UPDATE ABORTED — UPDATE

Dayfile message.

FILE NAME OR SL ILLEGAL — OVERLOG

Bad load sequence parameter list entry. lfn = zero or SL non-zero.

FILE NEVER OPENED — PFC

FILE NOT FOUND

System cannot find file specified in ENPR or EVICT type-in.

FILE NOT IN SYSTEM PFA

Permanent file specified on the preceding line not known to system.

FILE NOT IN USERS FL CTS

Address of request word is outside user's field length. Job terminated.

FILE NOT ON A PF DEVICE PFC

FILE NOT ON MASS STORAGE DEVICE pfn PFC

FILE x RECORD y HAS PHYS. REC. LONGER THAN 1024 WORDS COMPARE

Appears on job output file. Run ends when record y on file x contains too long PRU. Detected by a PP program (not by COMPARE). Limit accepted by PP program is probably 512 words; COMPARE buffer limit is 1024 words.

FILE RECORDS NOT NAMED IORANDM

Console, system and job dayfiles. IORANDM called to read or write beginning of random access record, identified by record name. If record was first written by number it must be addressed by number. Job terminated.

FILE SUCCESSFULLY PURGED PFP

Last cycle of a file has been purged.

FILE VACUOUS LDR,2LA

Input file initially positioned at EOF mark. Job terminated.

FL TOO SMALL FOR LOADER LOD

Approximately 2000 (octal) CM words are required for PP or CP loader and a very small relocatable program.

FNT FULL 1EJ

Output file cannot be created as FNT is full. Loop will continue until empty entry is found.

FNT FULL - DROP OR RECHECK RESTART

No FNT space is available. Operator may drop job, or type n.RECHECK.

FOLLOWING PERMANENT FILE(S) FOUND (CKP) CKP

Listed permanent files were attached to control point when CKP was called.

FORMATS INCOMPATIBLE***COPY UNCERTAIN***	COPYCR,COPYCF, COPYBR,COPYBF
Tape formats selected for input/output files can cause a possible loss of data significance. Job continues.	
FOURTH PARAMETER SHOULD BE "B" OR "C"	SKIPF,SKIPB
Control card did not indicate binary or coded file.	
FWA-LWA ERROR	IO
Console, system and job dayfile. READIN or WRITOUT macro is being executed; workspace (sixth word of FET) or FET is less than 6 words. Job terminated.	
GARBAGE IN OLDPL HEADER, UPDATE ABORTED	UPDATE
HIGH SPEED READ ON A NON-ALLOCATABLE DEVICE	LDQ
LDQ is trying to use a bad FST; job is terminated.	
ID NAME NOT IN INPUT FILES SEARCHED	COPYN
Either P1 or P2 cannot be located by the COPYN routine.	
IDENTIFICATION ERROR	PFA,PFC
ID parameter must be specified on PF CATALOG and ATTACH cards.	
ILLEGAL ADDRESS REQUEST TO APR	APR
Address in APR 5 or APR 10 call is out of range for the control point. Job terminates.	
ILLEGAL CHAR IN FILE NAME	DSP
File name for DISPOSE function must be alphanumeric. DISPOSE function will be ignored.	
ILLEGAL COPYL PARAMETER	COPYL
ILLEGAL DIAG. SEQ. PARAMETER	APR
Diagnostic sequencer received unacceptable parameter for console entry or program call card.	
ILLEGAL EOF	EDITSYM
EOF appears before EOR in common section. Run terminated.	
ILLEGAL EOR OR EOF	EDITSYM
EOR or EOF appears before *END in copying a deck. Run terminated.	

ILLEGAL FILE NAME	CIO

File name has embedded blanks, does not begin with letter, or exceeds 7 characters. Fatal error code 22.

ILLEGAL I/O REQUEST	6WM

First of a 4-line message issued as a result of a call from I/O driver; followed by:

```
FILE NAME xxxxxxx
FET ADDRESS yyyyyy
```

Fourth line describes error condition.

ILLEGAL LEVEL NUMBER	SKIPF,SKIPB
ILLEGAL PARAMETER LIST	JDP,TRANSR

Parameters in TRANSR, TRANSF and TRANSC control cards must be a legal job name left-justified, zero filled, and less than 5 characters.

ILLEGAL RECATALOG ATTEMPT pfn	PFC

File specified is already permanent.

ILLEGAL REQ FUNCTION	REQ

The REQ routine was called by a central program without auto-recall, with a non-zero status, or with an EST ordinal not in range of EST table.

ILLEGAL REQUEST FUNCTION (RESTART)	RESTART

Error occurred when RESTART attempted to request tape. Job terminated.

ILLEGAL SEQUENCE NUMBER ON EDITSYM CONTROL CARD	EDITSYM

Sequence number contains alphabetic or special character.

ILLEGAL TO TRANSFER AFTER DSD	EDITLIB

EDITLIB is constructing new system file, on which DSD has been written. TRANSFER card calls for one or more further records not listed in directory to be written on new file; DSD must be the last such record.

ILL-FORMED MULTIFILE SET	1MF

Fatal error, code 22. The multifile set tape was not created in the standard ANSI format.

IMPROPER MASTER CHARACTER CHANGED TO x	UPDATE
IMPROPER UPDATE PARAMETER, UPDATE ABORTED	UPDATE

Dayfile message. Unrecognizable parameter on UPDATE card.

INCORRECT OPERATOR ASSIGNMENT	REQ

Assigned equipment does not match type requested; operator should re-assign correct type.

INDEX ADDRESS NOT IN FL	1OP,1CL

Fatal error, code 22. All or part of specified index buffer is not within control point field length.

INDEX BUFFER TOO SMALL	1OP

Return dependent on EP bit setting. Error code 23. Specified index buffer is too small to contain the index logical record. First part of record has been read.

INDEX MAY BE INVALID - FILE WRITTEN SINCE LAST CLOSE	1OP

Information only. The last logical record in the file has been read into index buffer.

INPUT FILE ENDED BEFORE ((ADD)) CARD SATISFIED	EDITLIB

EDITLIB is trying to execute ADD, ADDBCD, ADDCOS, or ADDTEXT; input file ended before last or only program named in control card was found.

INPUT FILE ENDED PREMATURELY	EDITSYM

EOR appears before *END in a *COMDECK addition. Run terminated.

INPUT REC. FOR ((ADDBCD)) HAS IMPROPER NAME CARD	EDITLIB

EDITLIB is trying to add coded record to system file. Input record does not begin with Hollerith card containing record name starting in column 1, or the name is not acceptable.

INPUT REC. MISPREFIXED FOR ((TRANSFER))	EDITLIB

EDITLIB is trying to copy records from input file to new system file. TRANSFER card specified program name and input record had no prefix or a prefix contained different name, or control card did not specify a program name but input record did have prefix.

INSUFFICIENT FIELD LENGTH 1AJ

Program call card initiated load of absolute program or a 0,0 overlay, but field length was insufficient. Job terminated.

INSUFFICIENT FIELD LENGTH, UPDATE ABORTED UPDATE

Table manager ran out of room for internal tables.

INTERCOM NOT ON THIS SYSTEM DSP

User tried to dispose a file to an INTERCOM terminal, but INTERCOM not implemented on system. DISPOSE function is ignored.

INTERCOM JOB CANNOT DISPOSE TO E/I TERMINAL DSP

INTERCOM user tried to dispose to an EXPORT/IMPORT terminal. DISPOSE will be ignored.

INVALID CARD FORMAT LDQ

Premature termination character in loader directive. Fatal error.

INVALID CHARACTER LDR,2LA,LOADERE

Loader directive card legal characters are letters, numbers, parentheses, blank, comma, and period. Fatal error.

INVALID CONTROL CARD LOD,LDQ

Program call card for a PP program did not meet all requirements. Program name must begin with letter, cannot exceed three characters; no more than two parameters allowed.

INVALID CYCLE PFC,PFA

Cycle number requested for permanent file specified in previous line is invalid.

INVALID FET ADDRESS Nxxxx 2CA

INVALID LOADER DIRECTIVE LDR,2LA,LOADERE

On loader directive card first 7 characters are not SEGZERO, SECTION, SEGMENT, OVERLAY. Fatal error.

INVALID OVERLAY - LEVEL OR FWA 1AJ

Program call card initiated overlay load not of level 0,0 or first word address was not 100 (octal). Job terminated.

INVALID PRIVACY PROCEDURE PFC,PFA

INVALID REQUEST TO CPC	CPC,EDITLIB

Bits 54-59 of request word are 0, so it cannot contain PP program name in bits 42-59. Bits 42-59 can contain only number below 00010 (octal) as part of file action macro. Console, system and job dayfiles. Job terminated.

INVALID STACK ENTRY	1SX

Called by error code 22. Request stack entry contains undefined order code or out-of-range RBT address; or it points to FNT/FST that contains out-of-range RBT address. Dayfile message.

JOB HUNG IN AUTO-RECALL	1EJ

This job terminated because completion bit was not set and control point had no activity.

JOB PRE-ABORTED	1EJ

Appears at beginning of job output file if input card caused CKSUM ERROR RC.xxxx, CD.yyyy or FORMAT ERROR RC.xxxx, CD.yyyy. Job terminated when brought to control point.

JOB REPRIEVED	RPV

User job terminated, but was restarted in the recovery routine.

LDQ ERROR - FILE IS ACTIVE	LDQ

File status in FNT was BUSY. Fatal error.

LDQ ERROR - FNT ADDRESS = 0	LDQ

LDQ has been given an FNT address of zero. Job is terminated.

LDQ ERROR - NOT A CM PROGRAM	LDQ

Request to load other than a CM program; fatal error.

LDQ ERROR - PNT IS OUT OF RANGE	LDQ

LOADERQ gave LDQ an out of range PNT ordinal. Job terminated.

LDQ ERROR - RBT WORD PAIR IS NON-ZERO	LDQ

LDQ has a bad RBT word pair; job is terminated.

LDV CANT LOAD FROM ECS - IP.ECLIB = 0 LDV

Request to load system library overlay from ECS cannot be satisfied because necessary code has been omitted from the system. Re-assemble system program LDV with IP.ECLIB non-zero.

LEVEL NUMBER GREATER THAN 17B or 15D SKIPF,SKIPB

Third parameter on a skip control card exceeded 17 (octal) or 15 (decimal).

LEVELS NOT PERMITTED IN STANDARD CALL LOADER

If S and V bits are not on in user call, load is interpreted as normal. If L1 and L2 are nonzero, it may have been overlay or segment call but appropriate bit is missing. Processing continues as for normal load.

LFN ALREADY IN USE PFA

Logical file name on next message line is already in use at this control point.

LFN GREATER THAN 7 CHARACTERS SKIPF,SKIPB

LFN parameter of skip control card is too long.

LOADER BY THIS NAME NOT IN SYSTEM 1AJ

User request or control card selected loader but format or name was illegal.

LOADER CANNOT BE LOADED FROM ECS LOD,LOQ

PP portion of loader is not capable of loading CM portion from ECS. CM portion must be either CM or disk resident.

LOADER CONTROL CARD OUT OF SEQUENCE LOD,LDQ

NOGO or EXECUTE card before LOAD card.

LOADER NOT FOUND IN LIBRARY LOD,LDQ

CM Loader not in library directory.

LOADER TABLES GARBAGE, OR OVERLAY SEQ ERROR LOADER,LOADERE

LOADER tables are threaded in a list below LOADER in CM. If tables are destroyed by user program (in normal mode only) and job terminates; or if secondary overlay is attempted without first generating corresponding primary overlay (of zero level), THREAD routine cannot execute properly. The tables are also destroyed if a table pointer overflows.

LOADER, xxxx ERROR FLAG SET.	LDR,2LA	Precedes all error messages issued by LDR or 2LA. (2LE overlay is called to issue message). xxxx is FATAL or NON-FATAL.
LOC ARG ERROR	LOC	Incorrect parameter, such as last word address greater than field length.
LOD CANNOT FIND SYSTEM IN FNT	LOD	If program declares SYSTEM as common file, control point assignment of SYSTEM changes. LOD uses FNT for SYSTEM when CM Loader is read from disk. To determine a match with correct FNT entry, entry association with control point zero is verified.
L.SEQ LESS THAN 2CM WORDS	APR	Diagnostic sequencer lacks sufficient table space in CM.
MAXIMUM TAPE UNITS EXCEEDED	REQ	The control point already has the full allotment of tape units assigned. Job terminated.
MEM ARG ERROR STATUS ALREADY COMPLETE	MEM	Request made to MEM when completion bit was set in status word.
MESSAGE ARG ERROR	MSG	Address given to MSG is out of user's field length. Job terminated.
MESSAGE FORMAT ERROR	MSG	Character in message is 60(octal) or over. Job terminated.
MESSAGE LIMIT EXCEEDED	MSG	Number of messages issued by job exceeds installation maximum. Job terminated.
MLRS TOO LARGE	1WS,1NW	Fatal error, code 22.
MMTC MEMORY PARITY ERROR	8T1,8T2	Fatal error, code 4.
MODE CHANGE RC.xxxx, CD.yyyy	2RC	Appears at beginning of output file for job if input record contained cards of mixed format (binary, Hollerith,

MT uu NO WRITE ENABLE

A tape without a write ring cannot be written. If the tape is to be written, the operator should clear and unload the unit, insert a write ring in the reel and remount, and ready the unit.

MT uu NOISE WARNING n 1N2,1N3

Read recovery has entered phase 1-4. Informative message.

MT uu NOT READY

Job is trying to access unit, but finds it not ready. The operator should mount the proper tape, if necessary, and ready unit.

MT uu RD ERR DEV CAP EXC 1RV

PRU read exceeded maximum allowed.

MT uu RD ERR LOST DATA 1RV

Data was lost and could not be recovered.

MT uu RD ERR NOISE IN IRG 1RV

On forward read, noise existed in the interrecord gap, but it appeared larger on reverse read. With a tape in SCOPE format or an MMTC controller, data is probably valid.

MT uu RD ERR READ OPP MODE 1RV

Change mode of read request. Data received after this error may not be dependable.

MT uu RD ERR REC FRAGMENT 1RV

All SCOPE tape PRU's must conform to data format conventions.

MT uu RD ERR TAPE PAR ERR 1RV

Data record cannot be guaranteed unless this tape parity error is recovered.

MT uu RD ERR XMSN PAR ERR 1RV

Hardware error.

MT uu RD ERROR HARDWARE LD 1R3

During parity error processing, lost data or transmission errors occurred. Recovery routines cannot determine whether current record is noise.

MT uu REJECT

Hardware is rejecting a function code. Call a customer engineer.

MT uu RESERVED

The tape unit is reserved on another channel and should not be. Hardware or software malfunction occurred during tape input/output processing.

MT uu SKIPPED MFR xxxx 1N3

A multiframe block was skipped without verification. xxxx is PRU count.

MT uu SKIPPED SFR xxxx 1N3

A single frame block was skipped without verification. xxxx is PRU count.

MT uu WRT ERR BAD CBKSP 1P1

Informative message. Last good record was not found on the first attempt after a controlled backspace hardware function.

MT uu WRT ERR BAD ERASE 1P2

During tape erasing, parity error was detected; erasure was incomplete. If unerased data is greater than noise record length, the tape will be read correctly.

MT uu WRT ERR CANT FND LGR 1P1

Last good data record was not found.

MT uu WRT ERR CANT FND SNR 1P4

Expected valid system noise record was not found.

MT uu WRT ERR ERASE LIMIT 1P2

Installation defined number of erases occurred without successfully writing data record. An additional erase occurred for each GO command entered by the operator.

MT uu WRT ERR FL POS UNCRT 1P1

After a reverse read of a bad PRU, the tape appears to be positioned in the midst of the previous record.

MT uu WRT ERR FMK BFR LGR 1P1

During search for the last good record, a file mark was encountered.

MT uu WRT ERR FMK BFR SNR 1P4

File mark encountered before an expected system noise record.

MT uu WRT ERR LGR BFR SNR 1P4

Last good record occurred before an expected system noise record.

MT uu WRT ERR LOST DATA 1P1,1P2,1P3,1P4

Data lost while reading tape. Usually fatal error.

MT uu WRT ERR NO LGRL 1P1

TAPES table does not have a value for the last good record length.

MT uu WRT ERR NOISE IN IRG 1P3,1P4

Noise greater than allowable minimum between 2 PRUs.

MT uu WRT ERR POS UNCRT 1P1,1P3,1P4

Tape may be positioned in the midst of the last good record.

MT uu WRT ERR REV EQU LGR 1P1

A PRU read forward as noise length was length of the last good record when read in reverse.

MT uu WRT ERR PSBL FRAGMNT 1P1

Tape appears to contain fragment after reverse read of record written in error.

MT uu WRT RVD 1RV

Write parity error has been recovered.

MT uu WRT ERR SGL FRM NOZ 1P3

Installation recovery procedure does not allow system noise records. A single frame of noise was detected during tape read verification. Recovery of PRU cannot be guaranteed.

MT uu WRT ERR XMSN PAR ERR 1P1,1P2,1P3,1P4

Hardware error.

MT uu WRT ERR 25 FT ERASED 1P2

While searching for a good tape area, 25 feet of tape have been erased. Tape is not suitable for interchange purposes. If system noise records are allowed, this message is not issued.

MT uu XMSN PAR ERR MT uu XMSN PARITY ERROR	
Hardware malfunction.	
MULTI-FILE DISPOSITION ON UNLABELED FILE	REO
Multi-file disposition should occur only on labeled files.	
MULTIPLY DEFINED OUTPUT xxxxxx	LDR
LDR loaded this routine previously and cannot load it again.	
MUST POSITION MEMBER FILE	3DO
Fatal error, code 15. Function other than POSMF issued on file whose FET multi-file name field is non-zero.	
CHAR	COPYCR,COPYCF, COPYBR,COPYBF
Number of records/files to be copied is zero, a letter, or special character on the copy control card.	
NAME GREATER THAN 7 CHARACTERS	LDR,2LA,LOADERE
Name on a loader directive card is too large. Fatal error. Dayfile shows bad card.	
NEW FL TOO SMALL RFL, xxxxxx. (RESTART)	RESTART
Field length at restart must not be less than field length at checkpoint. xxxxxx is minimum field length for restart. Fatal dayfile message; job terminated.	
NO CORRECT PASSWORDS SUBMITTED	PFA
NO ENTRY MATCHES XFER	LOADER
If this warning results from EXECUTIVE or PROGRAM call card job is terminated; otherwise, nonfatal error bit is returned to user.	
NO E/I ID FOR CENTRAL SITE JOB	DSP
Central site user wishing to DISPOSE a file to an EXPORT/IMPORT terminal must specify an EXPORT/IMPORT identifier. DISPOSE function will be ignored.	
NO EXTEND PERMISSION	PFE

80-column). Record number is decimal xxxx, counting first (control card) record of file as 0000; card number at which first change occurred is decimal yyyy, counting first card as 0001.

MOVE ROUTINE FINDS CANT READ PROG. IN SYSTEM FILE — EDITLIB

Fault in disk file used by EDITLIB (not expected to occur).

MTxx ASSIGNED TO lfn — 2SU

Magnetic tape unit xx automatically assigned to file lfn.

MTxx ASSIGNED TO lfn
MOUNT SCRATCH/READY UNIT

Operator should mount scratch tape and ready unit xx.

MTxx ASSIGNED TO lfn
READY ABOVE UNIT

Operator should mount reel requested on unit xx.

MTxx IMPROPER TRAILER, EOV FORCED — 1RP

End of reel was reached on a labeled tap finding a trailer label. Likely tape was written with user processing bit on, should be read that way.

MT uu BAD DSCRD — 1P1

A block expected to be of noise length when reread was greater than noise length. Hardware malfunction.

MT uu BLANK TAPE READ

Informative message. Fatal error if detected by tape driver.

MT uu BLOCKS READ nnnnn — 4LB,4LC

Informative message reporting nnnnn number of blocks read when end-of-file or end-of-volume label is read.

MT uu BLOCKS WRITTEN nnnnn — 4LB,4LC

Informative message reporting nnnnn number of blocks written in the file. Displayed when system writes an end-of-file or end-of-volume trailer label.

MT uu LABEL INFO ERR IN FET — 4LB,4LC

Informative message. Programmer error.

MT uu LABEL PARITY ERROR — 4LB,4LC

Informative message. Tape is probably defective.

NO INDEX POINTER IN FET, OR 0 LGTH. INDEX	IORANDM
Console display, system dayfile and job dayfile. IORANDM called by READIN or WRITOUT macro to read or write beginning of random record; but FET is either too short to contain pointers to the record index or points to zero length index. Job terminated.	
NO INPUT FILE ON THE COPYN CONTROL CARD	COPYN
COPYN card must specify at least one input file.	
NO MOVE ALLOWED TO OR FROM ECS	EDITLIB
EDITLIB MOVE card calls for program move to or from ECS residence.	
NO MULTIFILE ENTRY	1MF
Fatal error code 22. POSMF function was issued for a member file of a non-existent set.	
NO OUTPUT FILE ON THE COPYN CONTROL CARD	COPYN
COPYN control card must specify an output file.	
NO PERM TO ADD CYCLE	PFC
NO PERM FOR PUBLIC ID	PFC
NO PERMISSION FOR PUBLIC FILE	PFC
NO PERMISSION GRANTED	PFA
NO PERMISSION TO PURGE	PFP
NO PERMISSION TO RENAME	PFR
NO PRIVACY PROCEDURE	PFA
Required privacy procedure not specified on ATTACH.	
NO PROGRAM LIBRARY INPUT TO BE EDITED	EDITSYM
Correction input encountered but no old program library specified on EDITSYM control card. Run terminated.	
NO PROGRAM FOUND FOR SEGMENT	LOADER
Not one program requested by user call could be found.	
NO READY	EDITLIB
A READY card did not precede the COMPLETE directive card encountered.	
NO ROOM FOR NEW CYCLE	PFC
Not possible to add new cycle to permanent file specified in previous line.	

NO ROOM IN INDEX FOR NEW NAME	IORANDM
Console, system and job dayfiles. WRITOUT macro called to write beginning of random access record named. Name is not in file index and no vacant slot is available. Job terminated.	
NO SITE DESIGNATOR WITH ID	DSP
Site indicator C, E, or I required.	
NO SPACE EST aa EQ bb AL cc P	3DO
No available mass storage for new or overflowing file; aa, EST ordinal; bb, equipment type; cc, allocation type; P, permanent file. Operator has option to drop job.	
NO TERMINATOR FOUND	LDR,2LA,LDQ
Sent to dayfile when loader directive card is not terminated by period or right parenthesis.	
NO TRANSFER ADDRESS	LOADERE, LOADER
At completion of load, no program provides transfer address for CP Loader. Job terminates. Error condition can be overridden by NOGO. card or EXECUTE,ent. card.	
NON-EXISTENT RBR REQUESTED	1SX
Called by error code 5. RBR pointer in RBT is greater than number of RBR's in system.	
NON-PERMANENT FILE CANNOT BE EXTENDED lfn	PFE
File with name lfn is not permanent file.	
NOT ALLOWED TO ADD ECS RES. PROG TO FILE SYSTEM	EDITLIB
ECS resident program cannot enter running system through EDITLIB.	
NOT ALLOWED TO DELETE ECS RES. PROGRAM FROM SYSTEM	EDITLIB
ECS resident program cannot be deleted from the running system.	
xx NOT IN PPLIB	PP RES
Resident called to load overlay not in library. Job terminated.	
NT uu RD ERR MMTC MEM PAR	1RV
Memory parity error occurred while trying to read the current data record.	
NT uu RD/WRT ERR .... (See corresponding MT message.)	

NO ROOM IN INDEX FOR NEW NAME IORANDM

Console, system and job dayfiles. WRITOUT macro called to write beginning of random access record named. Name is not in file index and no vacant slot is available. Job terminated.

NO SPACE EST aa EQ bb AL cc P 3DO

No available mass storage for new or overflowing file; aa, EST ordinal; bb, equipment type; cc, allocation type; P, permanent file. Operator has option to drop job.

NO TERMINATOR FOUND LDR,2LA,LDO

Sent to dayfile when loader directive card is not terminated by period or right parenthesis.

NO TRANSFER ADDRESS LOADERE, LOADER

At completion of load, no program provides transfer address for CP Loader. Job terminates. Error condition can be overridden by NOGO. card or EXECUTE,ent. card.

NON-EXISTENT RBR REQUESTED 1SX

Called by error code 5. RBR pointer in RBT is greater than number of RBR's in system.

NON-PERMANENT FILE CANNOT BE EXTENDED lfn PFE

File with name lfn is not permanent file.

NOT ALLOWED TO ADD ECS RES. PROG TO FILE SYSTEM EDITLIB

ECS resident program cannot enter running system through EDITLIB.

NOT ALLOWED TO DELETE ECS RES. PROGRAM FROM SYSTEM EDITLIB

ECS resident program cannot be deleted from the running system.

xx NOT IN PPLIB PP RES

Resident called to load overlay not in library. Job terminated.

NT uu RD ERR MMTC MEM PAR 1RV

Memory parity error occurred while trying to read the current data record.

NT uu RD/WRT ERR .... (See corresponding MT message.)

NUMBER OF RECORDS GREATER THAN 777777B — SKIPF,SKIPB

Second parameter of a skip control card exceeded maximum number of allowable records.

ONE OR MORE PASSWORDS REDEFINED — PFR

ONLY 1 OVERLAY DESIGNATOR USED — LDR,2LA,LOADERE

Overlay card does not specify both primary and secondary levels. Fatal error.

ONLY ONE PARAMETER — LDR,2LA,LOADERE

Overlay loader directive has only one level parameter. Fatal error.

OVERLAY CALL FROM RELOCATABLE — LOADERE, LOADER

Relocatable program called for overlay load. This mode is illegal; once overlay loading is initiated, control returns to called overlay. If file named in user call contains all absolute overlays and OVERLAY 0,0 has not been called, control will return to the user call. Fatal error.

PACKED CARD LONGER THAN 13 WORDS — EDITSYM

Input error; run terminated.

PARAM LIST TOO LONG (CKP) — CKP

The value of n in the user parameter exceeds 42 (decimal). Job terminated.

PARITY CHECK FILE x RECORD y — COMPARE

If x is first file named on COMPARE control card, y record cannot be read in either mode without a parity check. If x is second file named, record cannot be read without a parity check in mode corresponding to first file read. In either case, comparison of that record pair is abandoned and comparison continues with next record pair.

PARAMETER COUNT IS GREATER THAN 4 — SKIPF,SKIPB

Too many parameters on skip control card.

PARAMETERS 2 AND 3 ARE NOT NUMERIC FIELDS — SKIPF,SKIPB

PARITY ERROR LOADING FROM ECS yyyyyyy — 1AJ,LDV

Uncorrectable parity error occurred while loading absolute CPU program or overlay in system library from ECS starting at absolute address yyyyyyy (octal).

PARITY ERROR ON jobname Occurs while restoring OUTPUT queue. Type n.GO. to continue. Bad record will be placed in the queue.	XXXRESQ
PF ABORT	PFC,PFA,PFP PFE,PFR
PFD FULL, FILE NOT PERMANENT Informative message.	PFC
PFD 4/5 FULL Informative message.	PFC
PFM SYSTEM ERROR, CODE xx (CKP) Permanent file system error code xx occurred while trying to read permanent file name. Job terminated.	CKP
PFN ALREADY EXISTS	PFC
PFN FOUND IN SD xxx Permanent file in preceding message is in subdirectory xxx. Subdirectory parameter was omitted or is incorrect.	LDF,PFA, PFC,PFR
PHYSICAL/LOGICAL POSITIONS DISAGREE Information only, no error code. Tape file is not physically positioned at load point, although logical PRU count indicates that it should be.	2TO
PLS HAVE DIFFERENT CONTROL CHARACTERS, UPDATE ABORTED	UPDATE

PP CALL ERROR	1EJ
Job terminated because central processor requested PP program with name that did not begin with a letter.	
PP xxxxxx.xxx SEC	1EJ
Total peripheral processor running time for a job.	
PROGRAM xxxxxxx HAS ENTRY PT WITH DUPL. NAME	EDITLIB
Entry points in programs on system file cannot duplicate names in other programs on file. Exceptions are programs output by Chippewa operating system.	
PROGRAM NOT IN SYSTEM	EDITLIB
Program named on MOVE or DELETE control card not in system file. EDITLIB passes to next function card.	
PURGE ATTEMPT ON NON-PERMANENT FILE lfn	PFP
File named lfn is not permanent file.	
PURGE BIT SET - CANNOT CHECKPOINT (CKP)	CKP
Checkpoint cannot be taken on this job since a permanent file used by this job has been purged. Job will terminate.	
PURGED FILE CANNOT BE EXTENDED lfn	PFE
File named lfn has been purged.	
Ps xxxx KILLED MTR	MTR
s is PP number; xxxx is name of program communicating with monitor.	
P2 IS NOT IN THE FILE OR IS UNDEFINED	COPYN
P2 NUMERIC EXTENDS BEYOND DOUBLE EOF	COPYN
Numeric P2 on record identification card indicates a logical record beyond double end-of-file that terminates file.	
RANDOM CALL, NONRANDOM FILE	IORANDM
Displayed on console, system and job dayfiles. READIN or WRITOUT macro called IORANDM to beginning of random access record. FET shows file is not random. Job terminated.	

RBTC FULL, FILE NOT PERMANENT PFC

Informative message

READ OR SKIPF AFTER WRITE CIO,1MF

Return dependent on EP bit setting. Error code 30. Attempt was made to read or skip data on a file positioned after a newly written record.

READ PERMISSION NOT GRANTED CIO

Fatal error, code 22. Read function issued to a file for which read permission was not granted.

READY TWICE WITHOUT COMPLETE EDITLIB

EDITLIB can construct only one system file at a time; second file cannot be readied before a COMPLETE card appears on first file.

RECORD n IN FILE x p WORDS LONGER THAN SAME RECORD IN FILE y COMPARE

If nth record from file x is k + p words long, corresponding record of file y is only k words long. First k words of both records are compared, and message is put on output file. Comparison continues after skipping extra p words of file x.

RECORD NAME NOT IN FILE INDEX IORANDM

Console, system and job dayfiles. Random access record identified by name not in file index. Job terminated.

RECORD NUMBER TOO HIGH IORANDM

Console, system and job dayfiles. Random access record identified by record number beyond end of file. Job terminated.

RECOVER-BAD ARGUMENT LIST

Checksum option is selected, but address specified is less than FWA of post-processor routine.

RECOVERED PARITY ERROR

Successful recovery from a parity error on a device.

RECOVERED PARITY ERROR 1SX
PARITY ERROR POSITION LOST

Mass storage parity error was corrected, but a subsequent parity error destroyed identification of bad sector before it was sent to dayfile. Only RECOVERED PARITY ERROR appears in job dayfile.

RECOVR-TOO MANY RECOVERY REQUESTS

RECOVR-BAD CHECKSUM

Checksum of post-processor subroutine taken after job was restarted by system. Does not equal the one taken during initial call to RECOVR.

REJECT - xx STATUS aaaa bbbb — 1SX

Equipment xx is not available for execution of stack request. If equipment number is lost, REJECT - EQUIPMENT NO. LOST is displayed.

RELEASE ILLEGAL ON PERMANENT FILE — 4ES

Fatal error code 22 octal. Code return is dependent on error processing bit setting.

RENAME ATTEMPT ON NON-PERMANENT FILE — PFR

REPLICATION EXCEEDS AVAILABLE CORE — LOADER

Receiving address conflicts with loader or its tables.

REPRIEVE ABORTED CHECKSUM BAD — RPV

An attempt was made to reprieve job after an error, but recovery routine's checksum was in error.

REQ DEVICE TYPE ERROR — REQ

Requested device type is not recognized by system.

REQUEST FUNCTION - RECALL NOT SET — REQ

Displayed if recall bit of PP input register is not on when REQ is entered.

REQUEST FUNCTION ADDRESS ERROR — REQ

Address specified is not within user field length.

REQUEST LFN ERROR — REQ

File name is not valid.

REQUEST TAPE — 1LT,1BT,1DF

Operator must type n.ASSIGNxx; control point and xx is EST ordinal.

REQUESTED OVERLAY PROG. NOT FOUND — LDR,2LA

Overlay of given level not on file. Produced by LDR when absolute overlays are input; by 2LA when absolute overlays are being loaded. File is searched end-around up to EOF mark, rewound, then searched to where searching began.

REQUESTED SEGMENT INCOMPLETE	LOADER
Some programs explicitly requested cannot be located; message does not result from unsatisfied externals.	
RESCANNING SKIPPED PFD SECTORS	DUMPF
DUMPF is rescanning permanent file directory entries bypassed earlier because CM array was full; or entire directory has been scanned, leaving only skipped PFD entries to be dumped. No action required.	
RESTART COMPLETE	1RC
Indicates successful RESTART. Informative dayfile message.	
RESTART NUMBER TOO BIG (RESTART)	RESTART
Checkpoint number on RESTART card is greater than number of checkpoints on tape.	
REWRITE REQUIRES MODIFY PERMISSION	4ES
Fatal error code 22.	
RP TOO LONG, MAX USED	PFC
SD INVALID - IGNORED	PFA,PFC
SEG OR OVERLAY CARD PREV PROCESSED	LDR,2LA, LOADERE
SECTION card cannot be used in overlay mode; in segmentation mode, it must precede all segment cards. Dayfile message. Fatal error.	
SEGMENT CALL, BUT NO SEGZERO PRESENT	LOADER
When segment load, called by user, is from normal or overlay program, delinking point cannot be established, as certain pointers are established for all segment loading during initial loading of SEGZERO. Probably results from erroneous setting of s bit.	
SEGMENT CALL NOT IN SEGMENT MODE	LOADERE, LOADER
A user call for a segment load was encountered during a normal load. Probably the s bit in the call to the loader was set erroneously. Fatal error.	
SEGMENT OR SECTION CARD PROCESSED	LDR,2LA, LOADERE
Segments and sections are illegal in the overlay mode.	

SEGMENT REQUEST WHILE IN OVERLAY	LOADER

Loading of segments and overlays is mutually exclusive, although both may load normally. OVERLOD called for segment load; not possible as original pointers to segment tables will not have been preset by SEGZERO processor.

SEGZERO HAS NOT BEEN PROCESSED	LDR,2LA, LOADERE

SEGMENT card occurred before required initial SEGZERO processed; or SEGZERO card occurred after OVERLAY card processed. Cannot mix segments and overlay. Fatal error.

SEGZERO INCOMPLETE, JOB ABORTED	LOADERE, LOADER

Not all programs specified on SEGZERO card are on input file. Job terminated. If not all external references can be satisfied no error comment is made as this is normal for segmented job.

SEGZERO SEGMENT NAMES DIFFER	LDR,2LA, LOADERE

Appears on dayfile when the segment names differ for contiguous SEGZERO cards defining continuation parameters. Fatal error.

SITE INDICATOR AFTER = MUST BE C, I, OR E	DISPOSE

Site indicator (k) on DISPOSE card must be C (central site), I (INTERCOM terminal) or E (EXPORT/IMPORT terminal). DISPOSE card will be ignored.

SKIPF FUNCTION MET END OF FILE	EDITLIB

End-of-file occurred before a SKIPF satisfied.

SL LIST IS EMPTY, FATAL ERROR	LOADER

User call contained SL pointer with zero word address. (vacuous selective load). Fatal error bit returned in user reply.

SNAP CARD PARAMETER ERROR, ID = xxxxxxx	SNAP

Issued for each SNAP table containing an error. These SNAP cards will be ignored.

SNAP ERR, IA DUPLICATED, ID=xxxxxxx	SNAP

Card named has IA which was used earlier. SNAP will be ignored.

SORRY, YOU MAY NOT DEBUG THE DEBUG ROUTINES	TRACE

Trace range falls within debugging routines.

SOURCE AND DESTINATION BOTH SYSTEM EDITLIB

Most recent READY card was READY(SYSTEM) but succeeding ADD card named SYSTEM as source program to be added to current system library.

SOURCE AND OUTPUT FILES ARE THE SAME LOADERV LOADER

The output file specified on overlay card cannot be the same as source file for overlay generation.

SOURCE FILE WAS POSITIONED AT RECORD - EDITLIB

EDITLIB TRANSFER card names first of one or more records to be transferred from a source file to a new system file; but source file is not positioned at named record.

SOURCE IS ZERO IN REPL LOADER

No address given for source; it is set to RA+1 (usually contains zero) and replication is initiated.

SPECIFIED DEVICE NON-EXISTENT CIO,4ES

Fatal error code 22.

SPECIFIED EQUIPMENT UNDEFINED REQ

EST ordinal passed to REQ was 0. Fatal error.

SPECIFIED EST DEVICE DID NOT MATCH REQ

Requested device type does not match entry specified by EST ordinal.

SPECIFIED EST ORDINAL OUT OF RANGE REQ

EST ordinal passed to REQ exceeded EST table limits. Fatal error.

STACK DEPTH EXCEEDED UPDATE

Stack in which cards are saved becomes full while processing BEFORE or ADDFILE directives.

STOPPED AT IMPROPER RECORD EDITSYM

Record specified by *CATALOG control card is not common or text section record.

SUB-DIRECTORY 4/5 FULL PFC

Warning to operator that permanent file subdirectory is 80 percent full.

SYSTEM COMMUNICATION ERROR (1SP--1SX) 1SX

1SX called with invalid parameters.

SYSTEM ERROR IN LOADER, BAD FST	LOADERQ called LDQ to reset file status table of file being loaded, but LDQ detected error in new FST entry. This is a CP Loader error. If user has set particular parameters, this error may not be detected.	LOADERE
SYSTEM ERROR IN LOADER ...HELP...CALL CDC	LOADER is designed to fail safe; all communications with system are checked out and edited. If interface degenerates or new bug appears in LOADER, checking can produce this error comment.	LOADERE, LOADER
SYSTEM ERROR, MODULE OF LOADER NOT FOUND	LOADERV, LOADERS, LOADERE, or MAPOUT module of CP Loader could not be found when called by LOADERQ via LDQ because it is not in library or a bug exists in CP Loader.	LOADERQ
SYSTEM ERROR TAPES-TABLE	Fatal error, code 22. EST ordinal of tape unit assigned to file could not be located in tapes table in CMR. Call a CDC analyst.	1MF,1CL,1RP, 1EJ,REQ,2TO, 2SU,2TB
SYSTEM IS A RESERVED WORD	User tried to release the SYSTEM file or make it common.	1AJ
TAPE FORMAT ERROR, NOTIFY SYSTEMS	Queue name in tape label does not correspond to type-in entry. Drop job and start again with correct tape or type-in.	XXXRESQ
TERMINAL NUMBER BEYOND LAST IN DECK	Number on *INSERT, *DELETE, *ADD, *CANCEL, or *RESTORE card is greater than last sequence number in deck.	EDITSYM
TERMINAL NUMBER NOT FOUND, FUNCTION WENT TO NEXT PRIMARY	Second sequence number on *INSERT, *DELETE, *ADD, *CANCEL, or *RESTORE card does not occur in deck to be edited.	EDITSYM
TEXT AFTER COMMON IN FIRST FILE	Text decks appear in common section; run terminated.	EDITSYM
TEXT CARD CONTAINS AN ILLEGAL SEPARATOR	Only acceptable separators are + - , ( ) or blanks.	COPYN

TEXT HAS BEEN PROCESSED	LDR,2LA,LDQ

SECTION card occurs after text processed. Fatal error.

THE PARAMETERS ARE WRONG OR OUT OF ORDER	EDITLIB

Format error on TRANSFER card.

THERE HAS BEEN AN ECS PARITY ERROR AT xxxx	MTR

Permanent ECS parity error in block xxxx, absolute location divided by 1000(octal). Inform customer engineer.

THIS CARD COULDNT BE OBEYED	EDITSYM

First sequence number on *INSERT, *DELETE, *ADD, *CANCEL, *RESTORE card is not in deck to be edited.

THIS IS 1ST EDITLIB--NO RESTORE/RESET	EDITLIB

RESTORE or RESET not possible since no prior EDITLIB operations have occurred. EDITLIB call assumed.

THIS REC.NAME ALREADY IN OUTPUT FILE - xxx	EDITLIB

File xxx to be added duplicates a name already in file under construction. ADDCOS does not produce message, STITCH programs are not checked for duplication; nor will existence of STITCH program already in file cause message.

TIME LIMIT	1EJ

Job exceeded CP time limit requested on job card. Job terminated.

TOO MANY CHARACTERS IN PARAMETER	LDR,2LA,LOADERE

Level number on OVERLAY card is greater than 77(octal).

TOO MANY CORRECTION CARDS	EDITSYM

Space available for correction cards for one deck exceeded.

TOO MANY EDITSYM CONTROL CARDS	EDITSYM

Space available for EDITSYM control cards exceeded.

TOO MANY INPUT FILE NAMES ON COPYN	COPYN
TOO MANY NEW COMMON DECKS	EDITSYM

Names of new common decks exceed space available in CMNAME.

TOO MANY TEXT CARDS IN THE INPUT RECORD	COPYN

TRACE OR SNAP CARDS NOT CONTIGUOUS — LOD,LOQ

Issued by LOD. Job terminated.

TRACE OR SNAP TABLES OUT OF CORE — DEBUG

Issued when TBLNEXT and CORNEXT meet. Job terminated.

TRACE TABLES AND USERS PROG OVERLAP — TRACE

ADT tables built by TRACE overlap user's program. Job terminated.

TRIED TO WRITE A NOISE RECORD — 1MT,1WS,1NW

Fatal. No error code.

TRUNCATED LABEL COMMON xxxxxxx — 2LE

Previous declaration less than present declaration for same label common. First length and address used.

UNCORRECTABLE PARITY ERROR — 1SX

Parity error on a mass storage device could not be recovered. A display only identifies device and bad sector.

UNCORRECTABLE PARITY ERROR
PARITY ERROR POSITION LOST — 1SX

Mass storage parity error not recovered after 62 attempts to read/write PRU. Information identifying bad sector destroyed by subsequent parity error before sent to dayfile. Only first line appears in job dayfile.

UNCORRECTABLE PARITY ERROR
STATUS ssss cccc
EQ ee ADDR aaaa bbbb TRY 76
RBRxx RByyyy PRU zzzz
RECORDED ADDR pppp qqqq — 1SX

Only first line appears in job file. Error code 04. Mass storage parity error not recovered after 62 attempts to read/write PRU. Controller status ssss, converter status cccc, equipment number ee, physical address of bad PRU aaaa bbbb, logical address xx,yyyy, zzzz, and recorded physical address pppp qqqq are all octal. Last line appears only for 6638 disk unit; aaaa bbbb not equal to pppp qqqq, indicates positioning error.

UNKNOWN LFN — PFC,PFE,
lfn — PFP,PFR

File name lfn is not attached to this control point.

UNKNOWN LOADER	1AJ
Occurs when control card calls nonexistent loader.	
UNLOAD REJECTED - ACTIVE FILES ON xx	5DA
n.UNLOADxx. or n.DEVADDx,P,B. entered by operator (xx is private disk pack unit) and active files are assigned to it. UNLOAD function must be dropped from control point to be cleared.	
UNRECOGNIZABLE DISP CODE	DSP
Disposition code x for DISPOSE function must be a known disposition mnemonic. DISPOSE function will be ignored.	
UNRECOGNIZABLE EDITSYM CONTROL CARD	EDITSYM
Card at beginning of correction input or following *EDIT card is not EDITSYM.	
UNRECOGNIZABLE ID (LAST PARAM)	DSP
Identifier y for DISPOSE function must be 1 or 2 alphanumeric characters. DISPOSE function will be ignored.	
UNSAT BIT NOT PERMITTED, EXCEPT IN OVERLAY MODE	LOADER
A single program can be loaded by putting RSL or CLD name in lfn and setting SL = 0. To load absolute overlay from library, LOADER must be informed by setting overlay bit and unsatisfied external bit. At any other time setting this bit overrides error flags returned by LDR and could cause LOADER to begin loading absolute overlays on top of proper programs.	
USER ERROR, CANNOT FIND FILE NAME	LDR,2LA
User call for loader does not contain file name, or file name not found.	
USER ERROR, FIELD LENGTH TOO SMALL	LDR,2LA,LOADERE
User call to loader specified inadequate field length.	
USING LAST RB OF RBTC	PFC
Warning to operator that catalog is almost full. Also appears on system dayfile.	
WAITING FOR RBT STORAGE	3DO
No empty chain member exist.	

UNKNOWN LFN lfn	PFC,PFE, PFP,PFR

File name lfn is not attached to this control point.

UNKNOWN LOADER	1AJ

Occurs when control card calls nonexistent loader.

UNLOAD REJECTED - ACTIVE FILES ON xx	5DA

n.UNLOADxx. or n.DEVADDx,P,B. entered by operator (xx is private disk pack unit) and active files are assigned to it. UNLOAD function must be dropped from control point to be cleared.

UNRECOGNIZABLE DISP CODE	DSP

Disposition code x for DISPOSE function must be a known disposition mnemonic. DISPOSE function will be ignored.

UNRECOGNIZABLE EDITSYM CONTROL CARD	EDITSYM

Card at beginning of correction input or following *EDIT card is not EDITSYM.

UNRECOGNIZABLE ID (LAST PARAM)	DSP

Identifier y for DISPOSE function must be 1 or 2 alphanumeric characters. DISPOSE function will be ignored.

UNSAT BIT NOT PERMITTED, EXCEPT IN OVERLAY MODE	LOADER

A single program can be loaded by putting RSL or CLD name in lfn and setting SL = 0. To load absolute overlay from library, LOADER must be informed by setting overlay bit and unsatisfied external bit. At any other time setting this bit overrides error flags returned by LDR and could cause LOADER to begin loading absolute overlays on top of proper programs.

USER ERROR, CANNOT FIND FILE NAME	LDR,2LA

User call for loader does not contain file name, or file name not found.

USER ERROR, FIELD LENGTH TOO SMALL	LDR,2LA,LOADERE

User call to loader specified inadequate field length.

USING LAST RB OF RBTC	PFC

Warning to operator that catalog is almost full. Also appears on system dayfile.

WAITING FOR RBT STORAGE	3DO

No empty chain member exists.

WARNING BLANK COMMON GREATER THAN PREVIOUS DECL	LOADER

Subsequent references to blank common in a set of programs (segment, overlay, or a file loaded as a result of control card) cannot exceed first allocation. Job does not terminate; however, as no reference to blank common is truncated, it is possible for a program to destroy itself.

WPE UNRECOVERED. EOT FORCED, TYPE GO	XXXDMPQ

Permanent write error. Typing n.GO. will force EOT on bad reel and continuation reel will be assigned. Both reels must be read in when queue is restored.

WRITE NOT AT EOI ON PERMANENT FILE	4ES

Fatal error, code 22.

WRITE REQUIRES EXTEND PERMISSION	4ES

Fatal error, code 22.

xx BLANKED	1DA

Disk pack with EST ordinal xx blank labeled in response to type-in n.BLANK xx.

xxx ERRORS IN UPDATE INPUT	UPDATE

Dayfile message. First pass of UPDATE processing encountered xxx fatal errors while reading in a correction set.

xxx DECLARE ERRORS	UPDATE
xxxx KILLED MTR	MTR
xxx NOT IN PPLIB	PP RESIDENT

Resident called to load overlay not named in library. Job terminated.

xxxx OVERLAPPING CORRECTIONS	UPDATE

Dayfile message. Correction sets introduced during current run changed status of some cards more than once. Cards in question are printed.

xx RESERVED	REQ
xxx UPDATE ERRORS, JOB ABORTED	UPDATE

Dayfile message when errors encountered in reading the first card of the input file.

xx XMSN PARITY ERROR	REQ

1AJ CANT LOAD FROM ECS - IP.ECLIB = 0	1AJ

Control card request to load from ECS absolute central processor program in system library, cannot be satisfied because necessary code was omitted from system. Re-assemble system program 1AJ with IP.ECLIB non-zero.

1ST OVERLAY CARD HAS NO FILE NAME	LDR,2LA,LDQ

First character of first parameter on initial overlay card is not alphabetic. Fatal error.

1ST OVERLAY CARD LACKS 0,0	LDR,2LA, LOADERE

First overlay card does not designate level (0,0) overlay.

1ST PARAMETER MAY NOT EQUAL ZERO	LDR,2LA,LDQ

Overlay level (0,0) may not have secondary overlay levels (0,1 is illegal).

EC STATS xx aaaaaaaaaaaa bbbbbbbbbbbb	EC1

a...a last 12 octal digits of word xx. b...b last 12 octal digits of word xx+1. Gives ECS statistics accumulated since last call. Dayfile message.

EC STATS DISCONTINUED.	EC1

Dayfile message if ECS is off or delay interval is 7777.

EC STATS INTERVAL xxxx SECONDS.	EC1

Dayfile message issued each time delay interval xxxx is changed.

GO OR DROP EC STATS DELAY BEING SET.	EC1

Dayfile message informs operator of impending change in delay interval and requests approval.

NO ECS, OR ECS TURNED OFF.	EC1

Dayfile message. EC1 cannot find EST entry for ECS or ECS is turned off.

NOT AN ECS SYSTEM.	EC1

Dayfile message. EC1 initiated in system that does not define ECS.

**DUPLICATE PARAMETER cc	PF macros
**DUPLICATE PARAMETER ON PF CONTROL CARD	PFCCP
**FOLLOWING PF CONTROL CARD IN ERROR	PFCCP

**ILLEGAL OPTION cc = cccc	PF macros
**ILLEGAL OPTION ON PF CONTROL CARD	PFCCP
**ILLEGAL P.F. REQ **USER ERROR CODE XXX	PFCCP
INTERCOM real-time message indicating a bad permanent file request with corresponding error code.	
**MISSING NAME FOR OPTION cc =	PF macros
**PARAMETER VALUE EXCEEDS NINE CHARS	PFCCP
**PERMANENT FILE NAME EXCEEDS xx CHARS	PFCCP,macros
**PW OPTION HAS TOO MANY PARAMETERS	PFCCP
ADDFILE FIRST CARD MUST BE DECK OR COMDECK	UPDATE
Incorrect first card on file processed by ADDFILE directive. Fatal error.	
ADDFILE INVALID ON REMOTE FILE	UPDATE
ADDFILE directive cannot be used for a file to be entered onto OLDPL with a *READ directive. Fatal error.	
CARD NUMBER ZERO OR INVALID CHARACTER IN NUMERIC FIELD	UPDATE
Sequence number field on a correction control card is in error. Fatal error.	
CONTROL CARD INVALID OR MISSING	UPDATE
UPDATE found a format error on a control card, or a control card that could not be read. Illegal operation such as *INSERT prior to *IDENT may also have been attempted. Fatal error.	
DECK NAME ON ABOVE CARD NOT LAST DECLARED DECK	UPDATE
Informative message.	
DECK SPECIFIED ON MOVE OR COPY CARD NOT ON OLDPL. CARD WILL BE IGNORED	UPDATE
Deck does not exist on OLDPL. Informative message.	
DO/DONT IDENT xxxxxxx IS NOT YANKED/YANKED*	UPDATE
A DO card to negate the effect of a YANK references an IDENT that has not been yanked; or a DONT card to restore a YANK references an IDENT that was yanked. Informative message.	

DUPLICATE DECK NAME ON CREATION RUN UPDATE

Fatal error.

DUPLICATE DECK xxxxxxx NEWPL ILLEGAL UPDATE

UPDATE encountered active DECK or COMDECK card which duplicates a previous card. Fatal if NEWPL is being created.

DUPLICATE FILE NAME OF xxxxxxx, UPDATE ABORTED UPDATE

Same file name has been assigned to two UPDATE files. Fatal error.

DUPLICATE IDENT NAME UPDATE

During a merge run, UPDATE encountered a duplicate IDENT name which it could not make unique. Fatal error.

DUPLICATE IDENT NAME IN ADDFILE UPDATE

Name in file to be added as a result of an ADDFILE directive duplicates existing IDENT on OLDPL. Fatal error.

ERRORNOT ALL MODS WERE PROCESSED*** UPDATE

All changes indicates in the input deck were not processed. Fatal error. Names specified on corrections cards should correspond to deck or identifier names existing on OLDPL.

FILENAME OF xxxxxxx IS TOO LONG, UPDATE ABORTED UPDATE

A file name exceeded 7 characters. Fatal error.

FILE NAME ON ABOVE CARD GREATER THAN SEVEN CHARACTERS UPDATE

Fatal error.

IDENT CARD MISSING, NO NEWPL REQUESTED, DEFAULT IDENTIFIER OF .NO.ID. USED UPDATE

Informative message.

IDENT LONGER THAN NINE CHARACTERS UPDATE

Fatal error

IDENTIFIERS SEPARATED BY PERIOD IN WRONG ORDER UPDATE

Identifiers on PURGE, COMPILE, or SEQUENCE cards were in the wrong order. Fatal error.

ILLEGAL CONTROL CARD IN ADDFILE UPDATE

ADDFILE insertions cannot contain correction directives. Fatal error.

INVALID NUMERIC FIELD	UPDATE

Control card did not contain anticipated numeric information in a numeric field. Fatal error.

xxxxxxx IS NOT A VALID DECK NAME	UPDATE

Fatal error.

IT MAY BE IN A DECK NOT MENTIONED ON A COMPILE CARD	UPDATE

Unprocessed modifications were found in a Q mode UPDATE. Fatal error message.

LENGTH ERROR ON OLDPL. UNUSEABLE OLDPL OR HARDWARE ERROR	UPDATE

Card length on the OLDPL was greater than maximum allowed, or less than one. Fatal error.

NEW IDENT ON CHANGE CARD IS ALREADY KNOWN	UPDATE

An attempt was made to change an ident to one already in existence. Fatal error.

NO ACTIVE CARDS WERE FOUND WITHIN THE COPY RANGE. NULL COPY	UPDATE

Nonfatal error, UPDATE continued.

NO DECK NAME ON DECK CARD	UPDATE

Fatal error message.

NULL ADDFILE	UPDATE

The first read on the file specified by ADDFILE encountered an end-of-record. Informative message.

NULL DECK NAME	UPDATE

During ADDFILE or creation run, a DECK or COMDECK card encountered did not contain a deck name. Fatal error.

NULL IDENT	UPDATE

Fatal error.

OUTPUT LINE LIMIT EXCEEDED. LIST OPTIONS 3 AND 4 DEFEATED	UPDATE

UPDATE output exceeds line limit specified on a LIMIT card, if one is present. Non-fatal error.

PREMATURE END OF RECORD ON OLD PROGRAM LIBRARY UPDATE

Fatal error. EOR encountered in midst of card image. If a second run of this job is not succssful, the old program library probably has irrecoverable errors.

THE ABOVE CALLED COMMON DECK WAS NOT FOUND UPDATE

Fatal error.

THE ABOVE CARD AFFECTS A DECK OTHER THAN THE DECLARED DECK UPDATE

Non-fatal error. The card in question will be ignored.

THE ABOVE CARD IS ILLEGAL DURING A CREATION RUN UPDATE

Fatal error.

THE ABOVE CONTROL CARD IS ILLEGAL AFTER A DECK HAS BEEN DECLARED UPDATE

Non-fatal error.

THE ABOVE OPERATION IS NOT LEGAL WHEN REFERENCING THE YANK DECK UPDATE

Fatal error.

THE ABOVE SPECIFIED CARD WAS NOT ENCOUNTERED UPDATE

or

THE TERMINAL CARD SPECIFIED ABOVE WAS NOT ENCOUNTERED UPDATE

IT MAY BE IN A DECK NOT MENTIONED ON A COMPILE CARD

One of the first two lines of this message is printed as part of the printing of unprocessed modifications. The third line is printed when unprocessed modifications are found in a Q mode UPDATE. Fatal error message.

THE INITIAL CARD OF THE COPY RANGE WAS NOT FOUND. NULL COPY UPDATE

Non-fatal error.

THE TERMINAL CARD OF THE COPY RANGE WAS NOT FOUND. COPY ENDS AT END OF SPECIFIED DECK UPDATE

Non-fatal error.

TOO MANY CHBS -- INCREASE L.CHB UPDATE

Correction history bytes exceeded the specified limit of 100 octal for a card. Fatal error, job terminated.

UNKNOWN IDENT NAME xxxxxxxxx UPDATE

A correction control card referenced an IDENT not in directory. Fatal error.

WARNINGOLDPL CHECKSUM ERROR*** UPDATE

Dayfile message.

YANK OR SELYANK IDENT OR YANKDECK DECK NOT KNOWN UPDATE

Probably the ident referenced on a YANK or SELYANK card has been purged; applies to cards already on library. Non-fatal.

xxxERRORS IN INPUT. NEWPL, COMPILE, SOURCE SUPPRESSED UPDATE

Informative message. Fatal errors in input stream. UPDATE continues to process corrections in order to uncover further errors.

EDITSYM E

The EDITSYM program enables a user to organize his symbolic decks into new library files and to make corrections and alterations to existing library files. The symbolic decks in his library may represent source language cards in a compiler or assembler language, data decks, line images for a document, or any other type of symbolic information desired. EDITSYM produces a new library file in random format; once the symbolic material has been formed into a library file, EDITSYM provides two-level editing capability. Primary edit operations result in permanent alterations to the library file; secondary edit operations allow the file owner to keep track of changes and to evaluate the effects of alterations before making them permanent.

FILES PROCESSED BY EDITSYM

File Identifier	Function	Position After EDITSYM Use
I	Input file; supplies original source decks or corrections; also contains editor directives.	Left unpositioned.
C	Compile file; contains assembler/compiler source decks produced as output from the editor.	End-of-file mark written and file rewound.
L	List file; contains output listing of library file contents.	End-of-record written; list file not rewound if copied to job OUTPUT file.
OPL	Old program library file; input to editor for updating and correction.	Left unpositioned.
NPL	New program library file; primary output from file editor.	If written on tape, file is unlabeled. End-of-file written; not rewound.

EDITSYM CONTROL CARD

The SCOPE control card which calls the EDITSYM program is written:

EDITSYM(input file,compile file,list file,old program library file,new program library file)

The parameters may appear in any order, or they may be omitted if the standard options are elected. Each parameter may be written in several forms as follows:

File Parameter	Form	Meaning
Input	Omitted	Editor inputs on job INPUT file.
	I	Editor inputs on job INPUT file.
	I=lfn	Editor inputs on logical file name
	INPUT=lfn	Editor inputs on logical file name
Compile	Omitted	No compile file produced
	C=0	No compile file produced
	COMPILE=0	No compile file produced
	C	Output on file name COMPILE
	COMPILE	Output on file name COMPILE
	C=lfn	Output on logical file name
	COMPILE=lfn	Output on logical file name
List	Omitted	No list file produced
	L=0	No list file produced
	LIST=0	No list file produced
	L	List output on file name OUTPUT
	LIST	List output on file name OUTPUT
	LIST=L	List output on file name OUTPUT
	L=lfn	Output on logical file name
	LIST=lfn	Output on logical file name
Old Program Library	Omitted	No old program library required
	OPL=0	No old program library required
	OPL	Old program library on file name OPL
	OPL=lfn	Old program library on logical file name
New Program Library	Omitted	No new program library produced
	NPL=0	No new program library produced
	NPL	New program library on file name NPL
	NPL=lfn	New program library on logical file name

A primary function of EDITSYM is to produce a file of information in a format that can be input to any language processor, such as COMPASS, FORTRAN, etc. When this compile file is to be produced by EDITSYM, text decks are written to the file as a single logical record. The *WEOR directive can be used to force EDITSYM to write necessary end-of-record marks between logical text deck records. EDITSYM will honor *CALL directives and insert the called common deck into the compile file. Cards in the text deck marked as cancelled by secondary editing are not written to the compile file. Upon successful completion of EDITSYM operation, an end-of-file mark is written on the compile file and it is rewound.

EDITSYM DIRECTIVES

Directives must appear in the input stream to the EDITSYM program, whether it is on the INPUT file or on a logical file denoted by the user. The directives are used to identify new decks to be introduced into the library file and to denote operations the editor is to perform in correcting and updating an existing library file. The directives accepted by EDITSYM are:

*ADD	Add secondary text cards to a file.
*CANCEL	Cancel secondary text cards in a file.
*CATALOG	Produce a listing of text deck names.
*COMDECK	Identify a common deck.
*COMPILE	Produce selected compile file output.
*COPY	Selective copying of OPL to NPL.
*DECK	Identify a text deck.
*DELETE	Delete primary text cards from a file.
*EDIT	Select a deck in the file to be edited.
*END	Denotes the end of a source deck.
*INSERT	Insert primary text cards into a file.
*RESTORE	Restore selected edited text to its original state.
*WEOR	Write end-of-record mark on the compile file.

NEW DECKS

A library file is created from one or more source decks presented to the editor via the input stream. New decks may also be introduced into an existing library via the input stream. All new decks must be preceded and followed by editor directives.

A common deck is preceded by the directive

**COMDECK,deck name*

and is followed by the directive

**END*

The deck name parameter is required, and it may not exceed 7 characters. The library file section number assigned to all common decks has the value of 2. If common decks are introduced into a library file, the *COMDECK directive must precede any directives in the input stream which introduce or reference text decks.

A common deck in a library file makes it possible for decks which are common to several text section decks to appear in the library file once only. The following directive is placed within the body of a text deck at the point where the named common deck is to appear.

**CALL,deck name*

When the text deck is written on the compile file, the editor will honor the *CALL directive; the common deck will be located and written to the compile file at the point called. The *CALL directive is written to the compile file as a comment card.

A text deck is preceded by the directive

**DECK,deck name,text section number*

and is followed by the directive

**END*

In the *DECK directive, the deck name is required; and it may not exceed seven characters. The text section number is optional: the value may be 3 to 99 inclusive and is used to identify the text section of the library file in which the deck is to appear; if not present, the value 3 is assumed.

All new source decks introduced into a library file are considered to be primary level cards and all cards in each deck are given primary sequence numbers. A new deck is given an edition number of 1. Each time a deck in the library file is updated, its edition number is automatically increased by 1. The deck name, edition number and section number, along with other control information, is written in the two-word prefix of the logical record in which the deck is recorded in the library file. This information is included in the output when the deck is listed to the list file.

EDIT LEVEL CONTROL

EDITSYM provides two-level editing capability on old program library files: primary and secondary. Primary level editing operations rearrange the library and resequence the cards with primary sequence numbers. Secondary level editing operations do not result in resequencing; they do not alter the arrangement of cards with primary sequence numbers.

Primary sequence numbers are decimal integers 1 to n. Secondary sequence numbers are decimal numbers written in the form j.k where j is the primary sequence number of the preceding card and k is a secondary sequence number.

For example, in a library file containing a deck of cards sequenced from one to five, a primary level edit operation would be used to delete card number three. In the new library file produced, the deck would contain only four cards with their primary level sequence numbers changed to from one to four. If a secondary level edit operation was used to delete card three, the card image would still exist on the new file but it would be marked as cancelled. The primary sequence numbers would not be resequenced.

Similarly, new cards added to a file during a primary edit would be inserted where specified and the deck resequenced to give appropriate primary sequence numbers to the new cards. If new cards were added in a secondary edit, for instance, after card three, they would be assigned secondary level sequence numbers and the primary sequence numbers would remain unaltered.

Editor directives are used to designate corrections to be made to a library file. Primary and secondary level sequence numbers are given on the directives to specify the cards to be altered or after which new cards are to be entered. Sequence numbers are contiguous only within each deck in the file; they are not continued sequentially into the next adjacent deck; therefore, the name of the deck to be edited must also be specified.

Correction sets of one or more edit directives must be terminated with the directive:

**EDIT,deck name*

The deck name designates the deck to which the preceding directives apply.

PRIMARY LEVEL EDIT DIRECTIVES

The primary level edit directives are: *INSERT, *DELETE, and *RESTORE.

**INSERT*

The cards are inserted in the file after the card indicated by the integer primary sequence number n. The cards to be inserted follow the *INSERT directive in the input stream and are terminated by the next edit directive. All text inserted is considered to be primary text and will cause the entire deck to be resequenced if a new program library is to be produced.

**DELETE,m,n*

Cards specified by integer sequence numbers m through n are to be deleted from a deck. To delete just one card, the directive may be written:

**DELETE,m*

Any text cards following the directive in the input stream are inserted in the deck after the last card deleted. The deleted cards will not appear on the new program library; any cards inserted are primary level corrections. The entire deck will be resequenced when the new program library file is produced.

When a deck in a library file has been altered by secondary editing, designated portions of it may be restored to its original state by the directive:

**RESTORE,m,n*

All primary level cards from m through n, that have been cancelled (deactivated) by secondary editing are restored as primary text cards. All cards added by secondary editing within that range are removed from the deck.

SECONDARY LEVEL EDIT DIRECTIVES

The secondary level edit directives are: *CANCEL and *ADD.

**CANCEL,m,n*

Integer primary sequence numbers may be written for n and m; they may also be written in the form j.k where j is a primary and k is a secondary sequence number. This directive will cancel all primary cards within the range and remove all secondary level additions. Primary cards are not removed from the deck, but merely flagged as cancelled; secondary cards, however, are removed. New text cards may follow the *CANCEL directive in the input stream. Such insertions are placed in the deck immediately following the last card cancelled; the insertions are marked as secondary level additions. Cancellation does not cause resequencing of the primary level cards when a new program library is produced.

Secondary level additions to a deck are designated by the directive:

**ADD,n*

where n may be of the form j.k as defined above. The cards which follow the directive in the input stream are to be inserted in the deck after the card with sequence number n; they are inserted as secondary text. Addition does not cause resequencing of primary cards when a new program library is produced.

FILE CONTROL DIRECTIVES

EDITSYM may be directed to copy one or more text decks from the old program library file to the new program library file and/or to the compile file, through the directive:

**COPY,deck name,deck name*

Copying of the old program library will begin when the first deck name is encountered and continues up to and including the second deck name. If only one deck name appears, only that deck will be copied. If the directive is written in the following form, copying begins at the current position of the file and continues up to and including the deck name given:

**COPY,*,deck name*

If the directive is written in the following form, then copying will begin with the deck name and continue until the end of the file is reached.

COPY,deck name,

When a compile file is produced containing source language decks for different language compilers and/or assemblers, (COMPASS source decks mixed with FORTRAN source decks). The source decks must be separated with end-of-record marks.

The directive which causes EDITSYM to produce an end- of-record mark to terminate the logical record being written is:

**WEOR*

It may also be necessary for a programmer to separate decks written on the compile file in the same compiler source language when the compiler parameters differ for each source deck.

The compile file parameter given or implied on the EDITSYM control card may be overridden by the appearance in the input stream of the directive:

**COMPILE,lfn*

A compile file will be written on the logical file name (lfn). The *COMPILE directive will apply only to the deck name given on the *EDIT, *DECK, or *COPY directive which follows *COMPILE in the input stream. Once the directive has been encountered by EDITSYM, compile files needed for any decks remaining in the input stream and/or the old program library file must be requested by separate *COMPILE directives.

To produce a list of common and text deck names in a library file, this directive is required:

**CATALOG,lfn*

lfn is the logical file name of the library file to be listed; the catalog of deck names is produced on the job OUTPUT file.

EDITSYM EXAMPLES

1. Create a program library file NPL on magnetic tape from the decks supplied in the INPUT file; produce a listing of the file decks on the job OUTPUT file; no compile file is to be produced.

```
(job card)
REQUEST,NPL,MT.
REWIND,NPL.
EDITSYM(NPL,L)
UNLOAD,NPL.
7/8/9
*COMDECK,MACROS
.
.
*END
*DECK,PROGA
.
.
*CALL,MACROS
.
.
*END
*DECK,PROGB
.
*CALL,MACROS
.
*END
*DECK,FTNPROG
.
```

```
.
.
*END
*DECK,FTNSUBR
.
.
.
*END
6/7/8/9
```

2. Modify and assemble the COMPASS source deck PROGB from the library file created in example 1.

```
(job card)
REQUEST,OPL,MT.
REWIND,OPL.
EDITSYM(OPL)
UNLOAD,OPL.
COMPASS(I=COMPILE,B=PUNCHB)
7/8/9
*COMPILE,COMPILE
*INSERT,300
(Insertion cards to follow card #300)
*DELETE,2,3
(Replacements for deleted cards)
*EDIT,PROGB
7/8/9
(Data deck for execution)
6/7/8/9
```

In the above example, the compile file option was omitted from the EDITSYM control card, therefore no compile file would be produced. However, the *COMPILE directive countermands the option and a compile file containing only the modified source deck PROGB is produced. Since a new program library is not produced, the modifications to PROGB will appear in the compile file only; PROGB will remain unaltered on the old program library file.

3. Using the old library file created in example 1, produce a compile file containing selected decks modified for COMPASS assembly and FORTRAN compilation. Create a new program library containing two additional new decks; obtain a catalog listing of the new program library file contents.

```
(job card)
REQUEST,OPL,MT.
REQUEST,NPL,MT.
REWIND,NPL.
REWIND,OPL.
EDITSYM(NPL,OPL)
UNLOAD,NPL.
COMPASS(I=COMPL)
RUN(S,,,COMPL)
7/8/9
*COMPILE,COMPL
(Edit directives and correction deck)
*EDIT,FTNPROG
*COMPILE,COMPL
```

```
*COPY,FTNSUBR
*WEOF
*COMPILE,COMPL
(Edit directives and correction deck)
*EDIT,PROGA
*DECK,PROGC
(New source deck)
*END
*COMPILE,COMPL
*DECK,PROGD
(New source deck)
*END
*CATALOG,NPL
7/8/9
(Data deck for execution of COMPASS program)
7/8/9
(Data deck for execution of FORTRAN program)
6/7/8/9
```

In the above example, the *COMPILE directives select decks to be written to the compile file, COMPL. Since COMPASS decks are placed on COMPL first and followed by FORTRAN decks, an end-of-record mark must be placed between the two types of source language records with the *WEOF directive. EDITSYM will write a final end-of-file mark and rewind the compile file when it has finished processing the input stream. The COMPASS control card will cause all decks on the compile file up to the end-of-record mark to be assembled. The RUN control card will then enable the FORTRAN decks following the end- of-record mark to be compiled. A catalog listing of all the decks on the new program library file (NPL) will be written to the job OUTPUT file.

PRINT FILE CONVENTIONS F

Files with a print disposition (including OUTPUT) and files assigned to a printer, must adhere to specific format rules as follows:

1. All characters must be in display code.

2. The end of a print line must be indicated by a zero byte in the lower 12 bits of the last central memory word of the line. Any other unused characters in the last word should be filled with display code blanks (55_8). For example, if the line has 137 characters (including carriage control), the last word would be aabbccddeeff550000 in octal; the letters represent the last seven characters to be printed in the line. No line should be longer than 137 characters.

3. Each line must start in the upper 6 bits of a CM word.

4. The first character of a line is the carriage control, which specifies spacing as shown in the following table. It will never be printed, and the second character in the line will appear in the first positon; therefore a maximum of 137 characters can be specified for a line, 136 is the number of characters that will be printed. All characters apply to both the 501 and the 512 unless they are specifically designated otherwise.

CARRIAGE CONTROL CHARACTERS

Character	**Action Before Printing**	**Action After Printing**
A	Space 1	Eject to top of next page†
B	Space 1	Skip to last line of page†
C	Space 1	Skip to channel 6
D	Space 1	Skip to channel 5
E	Space 1	Skip to channel 4
F	Space 1	Skip to channel 3
G	Space 1	Skip to channel 2
H	Space 1	Skip to channel 1 (501)
		Skip to channel 11 (512)
I	Space 1	Skip to channel 7 (512)
J	Space 1	Skip to channel 8 (512)
K	Space 1	Skip to channel 9 (512)
L	Space 1	Skip to channel 10 (512)
1	Eject to top of next page	No space†
2	Skip to last line on page	No space†
3	Skip to channel 6	No space

†The top of a page is indicated by a punch in channel 8 of the carriage control tape for the 501 printer and channel 1 for the 512 printer. The bottom of page is channel 7 in the 501 and 12 in the 512.

Character	Action Before Printing	Action after printing
4	Skip to channel 5	No space
5	Skip to channel 4	No space
6	Skip to channel 3	No space
7	Skip to channel 2	No space
8	Skip to channel 1 (501)	No space
	Skip to channel 11 (512)	No space
9	Skip to channel 7 (512)	No space
X	Skip to channel 8 (512)	No space
Y	Skip to channel 9 (512)	No space
Z	Skip to channel 10 (512)	No space
+	No space	No space
0 (zero)	Space 2	No space
- (minus)	Space 3	No space
blank	Space 1	No space

When the following characters are used for carriage control, no printing takes place. The remainder of the line will not be printed.

Q	Clear auto page eject
R	Select auto page eject
S	Clear 8 vertical lines per inch (512)
T	Select 8 vertical lines per inch (512)
PM (col 1-2)	Output remainder of line (up to 30 characters) on the B display and the dayfile and wait for the JANUS typein /OKuu. For files assigned to a printer, n.GO. must be typed to allow the operator to change forms or carriage control tapes.
any other	Acts as a blank

Any pre-print skip operation of 1, 2 or 3 lines that follows a post skip operation will be reduced to 0, 1 or 2 lines.

The functions S and T should be given at the top of a page. In other positions S and T can cause spacing to be different from the stated spacing. Q and R need not be given at the top of a page as each will cause a page eject before performing its function.

INDEX

STAPLE STAPLE

FOLD FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

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

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
Documentation Department
215 Moffett Park Drive
Sunnyvale, California 94086

CUT ON THIS LINE

FOLD FOLD

STAPLE STAPLE

COMMENT SHEET

CONTROL DATA
CORPORATION

TITLE: 6000 Series SCOPE 3.3 Reference Manual

PUBLICATION NO. 60305200 REVISION B

Control Data Corporation solicits your comments about this manual with a view to improving its usefulness in later editions.

Applications for which you use this manual.

Do you find it adequate for your purpose?

What improvements do you recommend to better serve your purpose?

Note specific errors discovered (please include page number reference).

General comments:

CUT ON THIS LINE

FROM NAME: ______________________ POSITION: ______________________

BUSINESS
ADDRESS: ______________________

____

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

FOLD ON DOTTED LINES AND STAPLE

CONTROL DATA

1/2

3/4

1

1-1/4

►►CUT OUT FOR USE AS LOOSE-LEAF BINDER TITLE TAB

CONTROL DATA
CORPORATION

Pub. No. 60305200

Litho in U.S.A.