PROGRAMMING AND OPERATING MANUAL

REAL-TIME EXECUTIVE
BATCH-SPOOL MONITOR

FEBRUARY 1975

– IMPORTANT NOTICE –

This manual contains information on Hewlett-Packard Real-Time Executive Software. The reader is assumed to be a programmer familiar with one of the Hewlett-Packard programming languages, ALGOL, Assembler, or FORTRAN.

PRINTED IN U.S.A.
Part No. 92002-93001
Microfiche No. 92002-93002

HEWLETT-PACKARD COMPANY

Cupertino, California

First Edition

Printed in U.S.A.

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

PROGRAMMING AND OPERATING MANUAL

REAL-TIME EXECUTIVE

BATCH-SPOOL MONITOR

Printed: FEB 1975
Changed: MAY 1975

Manual Part No. 92002-93001

Printed in U.S.A.

HEWLETT-PACKARD COMPANY

Cupertino, California

First Edition

Printed in U.S.A.

PREFACE

The File Management Package provides both file management and batch processing capabilities. The Spool Monitor provides a spooling capability. The FMP can be used independently of the Spool Monitor, or they can optionally be combined to extend the FMP's capabilities.

The File Management Package and its file management and batch processing functions are described in Chapter I (which contains Sections I through IV). Section I is a general description of the File Management Package. including its functions, its constituents, its operating environment, and an overview of how it is used. Section II describes the subroutines that interface the user program to the FMP, as well as their calling sequences in both HP Assembly Language and FORTRAN. Section III lists the commands available that cause the FMP to perform its many functions. Section IV describes the system of error codes that inform the operator of improper actions.

The Spool Monitor and its spooling function, used to extend the batch processing capability of the FMP, are covered in Chapter II (which contains Sections V through IX). Section V is a general description of the Spool Monitor. Section VI describes its interface subroutines and their calls. Section VII tells how jobs are spooled into FMP Section VIII lists the spool commands and describes the turn-on sequence for the operator interface program known as GASP. Section IX describes the spool error codes.

Configuration of the Batch-Spool Monitor into the computer system is covered in Chapter III (which contains Sections X and XI). Section X describes the configuration process for the File Management Package. Section XI describes the configuration of the Spool Monitor.

The components of the Batch-Spool Monitor can be ordered by the following part numbers:

File Management Package:	Batch Monitor Program	92002-12001
	Batch Monitor Library	92002-16006
Spool Monitor	Spool Program	92002-12002 (RTE-II) 92060-12001 (RTE-III)

TABLE OF CONTENTS

CHAPTER I
BATCH AND FILE MANAGEMENT

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER I
BATCH AND FILE MANAGEMENT

SECTION I
GENERAL DESCRIPTION

INTRODUCTION

The HP RTE Batch-Spool Monitor is a software package that provides the following capabilities:

- File Management
- Batch Processing
- Spooling

The first two capabilities, file management and batch processing, are contained within the segment of the Batch-Spool Monitor known as the File Management Package (FMP). The spooling function is contained in the segment called the Spool Monitor (SM).

The File Management Package and its file management and batch processing functions are covered in Chapter I (Sections I through IV). For coverage of the Spool Monitor and its spooling function, refer to Chapter II (Sections V through IX).

The components of the File Management Package and the Spool Monitor are summarized in Table 1-1.

FMP CONSTITUENTS

The major segments of software that constitute the File Management Package (FMP) are as follows:

- Batch Library
- File Manager (FMGR)
- Directory Management Program (D.RTR)

The Batch Library is the group of utility subroutines described in Section II that interface the user program to the File Management Package (FMP). FMGR is the operator interface routine that allows the operator to control the file management and batch processing functions of the FMP either directly through system console commands or indirectly through user program internal scheduling. The FMGR commands are described in Section III. The Directory Management Program (D.RTR) is a program that allows the interface routines to communicate directly with the file directories. D.RTR is described further in Section II.

FILE MANAGEMENT

The File Management Package (FMP) is a programmer/operator interface to the Real-Time Executive operating system that is used to organize and systematize files to make access and maintenance easier. The FMP uses the Batch Library and File Manager (FMGR) program to maintain files on the Real-Time System on one or more discs, and to provide a uniform access method to all standard HP Input/Output devices. The FMP provides easy-to-use access methods in the form of operator commands entered through a system input device, or user program calls to one of the many FMP subroutines. All details of file access, including multiprogramming protection, are built into the File Management Package.

In the area of file management, the FMP performs the following functions:

- Provides security on system and file levels to prevent unauthorized access.
- Creates new files; modifies existing files; and purges obsolete files.
- Provides exclusive control of I/O devices, other than discs, which use standard peripheral calling sequences.

Table 1-1. Batch-Spool Monitor Components

Software Segment	Component Nomenclature	Items Included
File Management Package (FMP)	Batch Monitor Program	FMGR D.RTR
	Batch Monitor Library	Batch Library subroutines listed in Section II
Spool Monitor (SM)	Spool Program	JOB GASP SPOUT SMP DVS43 EXTND

CAUTION

To avoid disruption of data storage, the disc controller and removable disc cartridge must not be exposed to an area where a strong magnetic field may be present.

BATCH PROCESSING

The FMP extends the basic capability of RTE to provide for the processing of batch or stream operator-submitted jobs with little or no operator intervention. The FMP reads and automatically carries out operator instructions which are submitted with the related source data.

SOFTWARE ENVIRONMENT

The Batch-Spool Monitor operates under control of the Real Time Executive Operating System, either RTE-II or RTE-III. It is an option under RTE-II, but is provided automatically with RTE-III. With RTE-II, the Batch-Spool Monitor requires a minimum of 5K background memory; with RTE-III, it requires a minimum main memory partition of 7K, 6K for the program area and 1K base page.

Table 1-1 lists the software components required for the total Batch-Spool Monitor. The File Management Package (FMP) uses the Spool Monitor (SM) whenever thay are combined but will operate without the Spool Monitor. The Spool Monitor, on the other hand, requires the File Management Package for its operation.

HARDWARE ENVIRONMENT

The Batch-Spool Monitor uses essentially the same hardware as the RTE operating system under which it operates. With RTE-II, the Batch-Spool Monitor requires a 2100 or 21MX component with a minimum of 24K memory. If the Spool Monitor is omitted from the configuration, the memory can be reduced to 16K.

With RTE-III, the Batch-Spool Monitor requires a 21MX computer with a minimum memory of 32K. In either operating system, when the Spool Monitor is included, the system should also have a Line Printer. In addition, depending on the application, a total system may include a Card Reader, additional Discs, Magnetic Tape Drives, and so forth.

FILE MANAGEMENT GENERAL DESCRIPTION

As soon as anyone accumulates a few thousand of anything (except money), he may have a problem storing his accumulation; and then finding what he needs when it is required. As the collection grows and as time passes, the memory dims and the point is reached when this file grows so large that needed references cannot be found, and that forgotten items are without access. There are then, three basic problems to overcome;

	Problem	Solution
•	Where do you store vast collections?	Within the HP RTE System.
•	How do you manage the collections;	With the HP RTE File Management Package.
•	How do you remember what's in the collection?	Through a unique index scheme of your own built upon the index features of the File Management Package.

SOLUTION DEFINITIONS

HP RTE System is a multiprogramming system that allows several programs to operate concurrently, each program executing during the unused central processor time of the others. (A complete description is contained in the RTE Programming and Operating Manual.)

HP RTE File Management Package is a programmer/operator interface to RTE that is used to organize and systematize files on the RTE System to make access and maintenance easier.

Index Schemes come in many different forms for many different applications. The user must be concerned with a minimum of two indexes; one that the FMP itself forms and maintains, and one that the user should form for cross-referencing purposes. A discussion of the basic principles of indexing as they would apply to the indexing activities external to the computer system is contained in Appendix H.

ANALOGY

As shown in Figure 1-1, the HP RTE Operating System is likened to a large filing cabinet that is capable of storing a very large quantity of information. The HP RTE File Management Package (FMP) can be compared to an office clerk that is responsible for taking care of the filing system. The responsibilities of the FMP, using the office clerk analogy, would be as follows:

- To set up the entire filing system with a system security code according to the dictates of the user. The user provides the individual file names and the type of each file. The user can also flag each file with a security code so the FMP can check another user's right to examine a particular file.

- The FMP will allow the user to:

  Add new files

  Open an existing file to:
  - Add records
  - Delete records
  - Change records
  - Copy records or information

  Close an opened file

  Purge an obsolete file.

- The FMP will resolve conflicts between users who want to open the same file. A user may request that a file be opened exclusively to him. This will be granted by the FMP only if the file is not in use. A user may request that a file be opened non-exclusively, that is, he will share it. Each time a request is received to open a file, exclusively or non-exclusively, all users

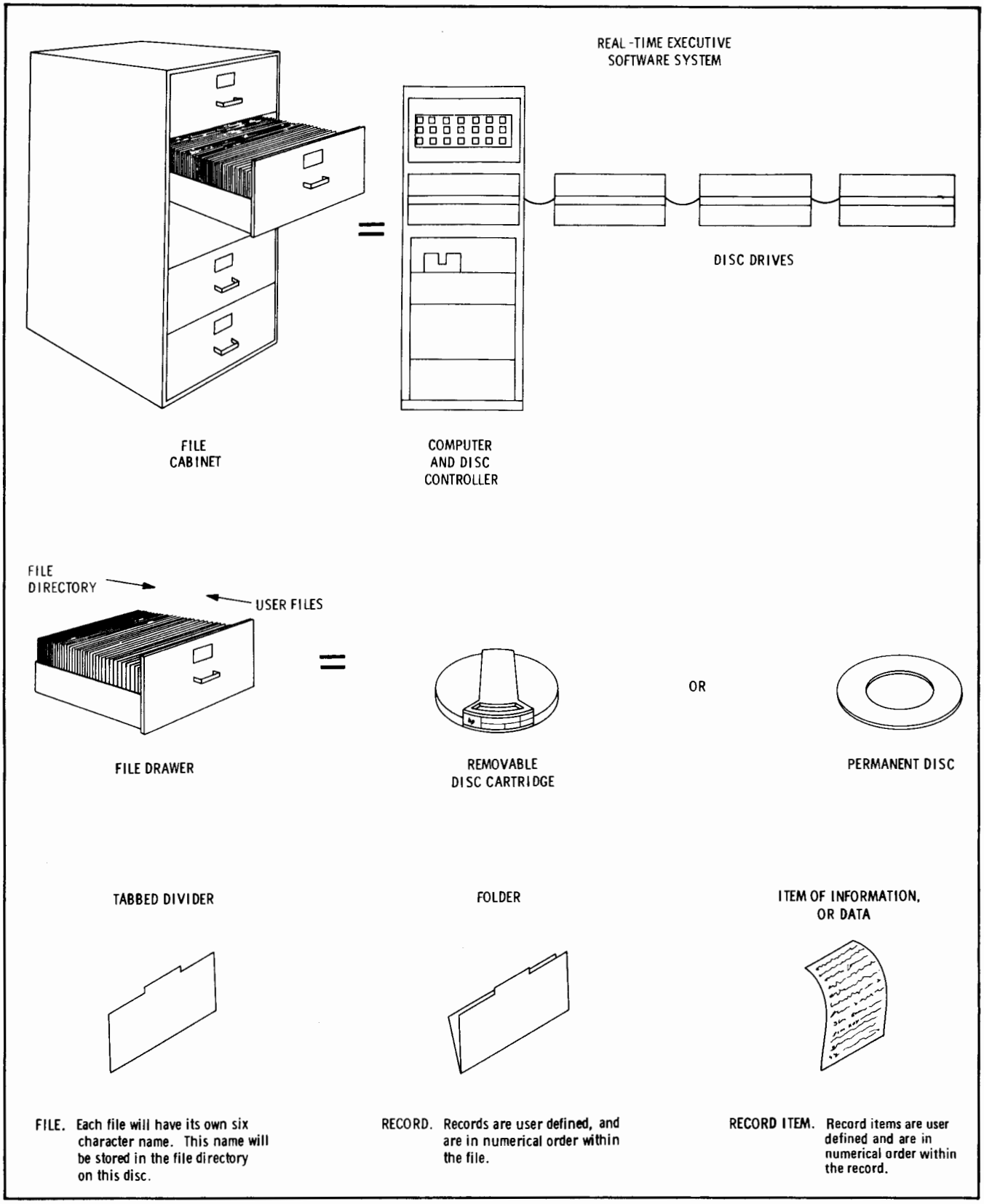


Figure 1-1. Analogy of the RTE FMP.

that currently have that file open are checked. If they are not currently using the file, it is closed to them and returned to the system. The FMP will grant the non-exclusive opening of a file only if it is not already exclusively open, or already being shared by more than seven users.

- The FMP can understand users that talk in FORTRAN IV, FORTRAN II, ALGOL, and HP Assembly Language.

- The FMP will communicate messages to the user that will inform him of mistakes he has made in filing. FMP will also communicate messages in regard to overall system performance.

The previously described responsibilities or qualifications are only a general indication of how the FMP manages and organizes user files. The FMP is versatile enough so that the user can configure the filing system to suit his individual needs. The FMP will recognize user requests via the written memo method (programming) where the user details his instructions completely, or by the simpler verbal directive method (entries through the system input device). Both methods are described in Sections II and III, respectively.

A key to effective use of the FMP is organization. How to organize large collections in order to find any single item easily. The FMP will do the actual filing and housekeeping but you must organize and index your items for efficient retrieval. Refer to Appendix H for additional information on indexing.

FMP TECHNICAL DISCUSSION

Many different terms related to the HP File Management Package appear throughout this manual. For convenience these terms are all defined below.

CARTRIDGE — This term is used interchangeably with disc and means a rotating random access storage device. Cartridge usually implies removability.

FILE — A collection of records usually terminated by an end-of-file (EOF). A file may have zero or more records in it. In the FMP, a disc file is usually designated by a unique six-character name.

RECORD — A collection of 16-bit words usually terminated by an end-of-record (EOR) mark. A record may have zero or more words in it. FMP record formats are described in detail under File Types.

EOF — An end-of-data mark of higher level than an EOR mark. The particular type of EOF mark is determined by the type of device referenced.

FMP EOF conventions:

a. When the FMP is performing a non-disc READ:

1. Magnetic tape, cassette, and card reader use bit 7 in the STATUS word, – or –
2. A zero length record and the device is up (card reader encountering a blank card), – or –
3. A zero length record and the device type is less than 10 (i.e., DVR00 to DVR09).

b. When the FMP is performing a non-disc WRITE:

1. An EOF is requested for a magnetic tape or cassette, – or –
2. A double space (two carriage return/line feeds) is entered on the teletype, – or –
3. A page eject control function is used for the line printer, – or –
4. A leader control request is used for paper tape.

c. When the FMP is performing a disc read or write, the EOF is defined by the file type (see File Types).

SECTOR — A sector is part of a disc track 64 words in length.

BLOCK — Two disc track sectors of 64 words each (total of 128 words).

FMP ORGANIZATION

FILE ORGANIZATION–Files may exist on the disc tracks known by the system (i.e., system and auxiliary discs), and on peripheral discs. Tracks on the system and auxiliary discs are FMP assigned so that only FMP routines may write on them. Files are directed to preferred discs by the user. If a particular disc is not selected in the call, the FMP will direct the file to a convenient location – the user does not have to remember track and sector addresses, just the file name and security code.

DISC ORGANIZATION–Disc files are located on contiguous tracks within the FMP area, with user files beginning on the first (lowest numbered) FMP track and working up, and directory entries for the user files beginning on the last (highest numbered) FMP track and working down. No file will be allowed to cross a disc boundary; however, the assigned file area may include several discs. All track numbers refer to relative track numbers, that is relative to the track on the given unit established as track 0 at system generation time. For fixed-head discs, physical track 0 and relative track 0 are the same. For moving-head discs, relative track 0 is established during RTGEN time.

NOTE

> For moving-head discs, all starting track numbers must be the same if the cartridges are to be exchanged LU to LU or system to system. This is due to the fact that FMP uses relative track addresses in the directory.

All files are subject to moving as part of any packing operation done on a disc. This makes files relocatable; therefore, no absolute file addresses should be retained in any file or program.

Blocks are defined as two disc sectors of 64 words each. All files are constrained to start on even sector boundaries and all accesses, except to type 1 files, are 128-word accesses. Therefore, all accesses are directed to even sectors. No cyclic checking is done, and all disc errors are passed back to the caller for action. The error code is printed on the system log device when using the FMGR operator interface routine, or passed to the user program in parameter IERR when using one of the library subroutines.

BAD TRACKS

Bad track information is entered using the FMGR Initialization Command. This information is kept in the affected disc's first directory entry. If, at creation or during a packing move, the area for the file includes a bad track, the first track of that file will be increased until the file no longer contains any bad tracks. If CREAT is to use the rest of the disc, the whole area will be above the highest bad track. If, during packing, a file is found to include a declared bad track, the file will be purged. (Note: This can only happen through an explicit declaration of a bad track by an operator who knows the system security code.) In this manner, only the PACK and CREAT routines need be aware of bad tracks. Bad tracks discovered by the FMP result in an error return to the caller.

DIRECTORIES

The FMP creates and maintains two indices. The master index, or disc directory, contains information on all discs assigned to the FMP, and the local index or file directory contains information on each file on that particular disc.

DISC DIRECTORY–The first two sectors of the last track of the system disc (LU2) that is assigned to the FMP contains information on all discs which are currently mounted and which contain tracks assigned to the FMP. This directory also contains a setup code word and the master security code. The layout of this directory is shown in Appendix A.

FILE DIRECTORY–Each disc which has tracks assigned to the FMP will contain a file directory. Each file directory entry uses 16 words. The file directory starts in sector 0 of the last track which is available to the FMP. The first entry in the file directory will pertain to the disc itself and will contain label and track information. Each subsequent entry will contain information on a file. The last entry will be followed by a zero word. The first entry and each entry for purged files has the sign bit set on its first word to indicate that it is normally to be ignored. Refer to Appendix A for the file directory format.

FILE TYPES

Files are divided into three categories shown below:

Category	Type	Description
Control	0	Non-disc file
Fixed lengths, random access, non-extendable	1	128-word record length
	2	User selected record length
Variable length, sequential access,automatic extents	3	Variable record length
	4	ASCII data
	5	Relocatable binary code
	6	RTE load module
	7	Absolute binary code
	>7	User defined

TYPE 0–A type 0 file is created with the CR Operator Command, and is used to control devices other than discs. Type 0 files also afford a measure of device independence, in that the standard file commands can be used to control the device. A type 0 file can only be created on the system disc (LU2); therefore, the file directory entry will be located in the system disc file directory, and have some special entries relating to logical unit number and end-of-file code. Cooperating programs may use type 0 files as a means of controlling access to a device (see exclusive open).

TYPE 1–Type 1 files have fixed record lengths of 128 words. Since the smallest addressable block is 128 words, short blocks will be filled from the user's buffer. Transfers to or from type 1 files are done directly to or from the user's buffer, and may be any length; thus, this type file has the fastest transfer rate. Positioning on type 1 files assumes a record length of 128 words. Any file except type 0 may be opened as a type 1 file..

TYPE 2–Type 2 files have record lengths that are user defined. Each transfer is one and only one record long, and must go through a packing buffer (the buffer in the Data Control Block); thus, the transfer rate will be slower than type 1 files.

TYPE 3–Type 3 files are packed on the disc and contain data, source, relocatable, etc. Each transfer is one and only one record long, and must go through a packing buffer (the buffer in the Data Control Block).

TYPE 4–Same as type 3, except the FMGR operator interface routine recognizes (defaults) type 4 files as containing source programs.

TYPE 5–Same as type 3, except the FMGR operator interface routine recognizes (defaults) type 7 files as containing absolute binary code.

TYPE 6–Same as type 3. This file is created by the Save Program (SP) operator command as a type 6 file, but is always accessed as a type 1 file. Refer to RECORD FORMAT heading under type 6, and the SP Command for more information.

TYPE 7–Same as type 3, except the FMGR operator interface routine recognizes (defaults) type 7 files as containing absolute binary code.

TYPE> 7–Same as type 3 – user defined.

All files, except type 0 files, may be accessed as type 1 (i.e., direct transfer through user's buffer); however, no other type modifications are allowed. The major difference between the files is the speed of positioning. A type 1 or 2 file may be positioned without any disc accessing, whereas other type files require at least one, and in some cases, several accesses. The user in no case need worry about a file or record crossing sector or track boundaries.

UPDATE FILES–Update implies that a file is to be modified in some manner. It is usually desirable to modify only a portion of the file without disturbing the remainder of the contents.

The end-of-file mark of type 1 and 2 files is the last word of the last block in the file; however, in other type files it is a special word (–1) for the length of a record. In order to preserve data in a type 2 file, it should be opened for update whenever any non-sequential accesses are to be performed, and the file is to be modified. That is to say, a type 2 file should be modified in non-update mode only when originally writing the file, or adding to the end of the file, and then only if it is to be written sequentially. Update/non-update has no effect on reading or positioning.

In files of type 3 and above, an end-of-file mark is written in the non-update mode. In either mode, records written beyond an end-of-file are written in non-update fashion. That is, replace the end-of-file with the new record, followed by an end-of-file.

EXTENDABLE FILES–An extendable file (type 3 and up) is a file that is automatically extended in response to a write request to points beyond the range of the currently defined file. The extent is created with the same name and size, and the access is continued. Extensions will be of the same size as the base file, and all extensions of any given file will be on the same disc. No flags or pointers relating to the next extent are kept in the file area. Open flags are kept only in the base file directory entry.

RECORD FORMATS OF FILES

Type 0–The type 0 file's record is defined by the device type. The following description describes type 0 files from paper tape and card devices. (These conventions are incorporated in the device driver and not the FMP.)

a. ASCII

1. ASCII records are punched according to the table in Appendix E.

2. On paper tape, records are ended with a carriage return character followed by a line feed. A rubout character anywhere in the record indicates that the record is to be ignored. Backspace characters cause the previous character and backspace character to be deleted from the record.

3. On cards, records are terminated explicitly by the highest non-blank character position, less than or equal to the read request length.

b. Relocatable Binary Records.

1. The complete format is given in Appendix F. The FMGR program uses the length and checksum words to do checksum on relocatable records.

c. Binary Records.

1. Binary records have the length in word one (first character) and the rest is unspecified. These records are generated by binary write requests to the formatter directed to a paper tape device. A checksum is not performed.

d. Absolute Binary.

1. The Absolute format is given in Appendix G. This record type is used on tapes usually loaded by the Basic Binary Loader (BBL) or the Basic Binary Disc Loader (BBDL). The FMGR program uses the first and last words to perform a checksum on these records. It is also necessary to set subfunction bits (23 octal) to read absolute records.

TYPE 1 AND 2–Type 1 and 2 files are written as presented. They are packed, and may cross sector and track boundaries. The end-of-file is the last word of the last block in the file.

TYPE 3, 4, 5, 6, 7, AND ABOVE–Type 3 and above records are preceded and followed by a length word which contains the length of the record (exclusive of the two length words). A zero length record consists of two zero words. An end-of-file is indicated by a –1 for the first length word. Words following the end-of-file are undefined.

TYPE 6–Program Save Files are created as type 6 files by the SP Command, and are always accessed as type 1 files by FMGR. The first two sectors of the file are used to record ID information on the program. The format of this information is specified in appendix A, Table A-4.

MULTIPROGRAMMING CONSIDERATIONS

FILE CONFLICT–In order to prevent two or more programs from destructively interfering with each other, a file may be opened either exclusively or non-exclusively.

EXCLUSIVE OPEN–An exclusive open is granted only to one program per file at a time.

NON-EXCLUSIVE OPEN–A non-exclusive open may be granted to as many as seven programs per file at one time. A non-exclusive open will not be granted if the file is already opened exclusively.

DIRECTORY CONFLICT–The FMP keeps open flags in the directory. This means that the directory will be modified by the following functions:

- Creating a file
- Opening a file

- Closing a file
- Purging a file
- Changing a file name.

Since all these actions may be programmed by the user, it is clear that there could be conflict. The FMP resolves this conflict by allowing only one program to write on the directory tracks. This program is the D.RTR routine. D.RTR is scheduled with wait whenever a directory change is required; that is, for all of the above functions. The D.RTR program will own outright the directory on logical unit 2, and logical unit 3, if any.

SECURITY–There are two levels of security, file security and system security.

FILE SECURITY–Each file has a security code. This code may be positive, negative or 0. If positive and the code at open does not match, the file may be read but not written on. If negative and the code at open does not match, the file will not be opened. If zero, the file may be opened to any caller with no restrictions.

SYSTEM SECURITY–During system setup, a system security code is entered. If zero, no security is given. If non-zero, the code must be known in order to get directory listings to include file security codes, and in order to redefine the FMP disc areas.

FMP PROGRAM CALLS

The programmer communicates with the FMP through the type 6 or 7 utility subroutines provided on the Batch Library tape (refer to Tables 1-1 and 2-1). The subroutines will perform the following functions:

- Position a file to a known record address
- Close a file
- Create a file
- Send RTE-II control request to a type 0 file
- Return 125 words of Disc Directory
- Return file status
- Rename a file
- Open a file
- Read or Write as specific record
- Purge a file
- Read a file
- Reset a file to first record; if type 0 file, a rewind request is generated
- Write a file
- Determine the size of the DCB buffer used
- Flush DCB buffer to disc (if needed) and clear in-core flags.

FMGR OPERATOR COMMANDS

The operator controls FMGR with commands entered through the system input device. The commands can be entered manually through the system teleprinter, or on paper tape, cards, or other form through the appropriate input device. The commands can also be stored in a file, with control being transferred to that file for execution.

The operator first gains the attention of FMGR by scheduling the software module FMGR with the ON, FMGR operator command. When FMGR runs, a colon (:) prompter is returned on the initial keyboard input device. If the input device is not a keyboard, the first command is read from the device that is the input.

If desired, each command will be echoed on the system log device (refer to SeVerity Command). If an error is detected (input or reference), an error number is printed on the log device and input control is switched to that device. This halts the process and requires operator intervention to correct the error.

BATCH UTILIZATION

The term "batch" is usually used to describe the process of placing more than one program unit (called jobs) in a card reader. From the card reader, they are processed by a computer. The running of a job and the transition from job to job takes place with a minimum of human intervention.

If the jobs are processed directly from the card reader, it is obvious that they must be taken serially. Also, if the output goes directly to a printer, punch, or other slow I/O device, the computer operating system must wait for the completion of one job before processing the next job. Batch processing serially from a card reader does not allow consideration of job priority other than by the operator manually inserting one job ahead of others. A low priority job's lengthy printout may delay execution of a higher priority job.

SPOOLING

One solution to the problem of serial processing is "spooling". (See Figure 1-2.) Spooling is the process of reading the jobs from the serial input device and placing each job in a disc or "spool" file. Since all jobs are on a random access device, it is possible for a job monitoring program to select a job for execution according to its priority. Similarly, if job print or punch output is spooled to disc files, transition between jobs is faster as one job does not have to wait for completion of a previous job's output. The output files are disposed of by a special "spool-out" routine that is independent of initiation of the next job. Additionally, as the output files are on a random access device, they may be held until the operator takes action to dump them; this is especially convenient if the output requires a special form on the line printer.

Spool files can be automatically allocated by the system or they can be designated by the programmer. Simplicity of use is achieved if the user defaults to allow the system to allocate and release spool files. On the other hand, if the user specifies an output spool file, that file not only functions as an output spool, but it can be retained as a permanent file. Similarly, if the user defines his own job input file as an input spool, the job can be retained in that file, and at a later date it can be re-run by requesting the operating system to execute the job contained in the file. This technique does away with the necessity of having to place a frequently run job on the serial input device each time the job is to run.

This ability to execute a job from a file is perhaps of more value to the minicomputer user than is the batching of jobs submitted by various programmers. Although it is entirely possible that a user may fully utilize the RTE-II system's batching capability in a "closed-shop" fashion, it is more likely that the user will elect to place standard or repetitive procedures in the form of jobs on disc files, where the operator can request their execution as needed.

Before batching became available, RTE required that all program development and execution be done interactively from the system console. Thus, a compilation with-load-and-go required that the programmer be present at the console to enter each command as required. With

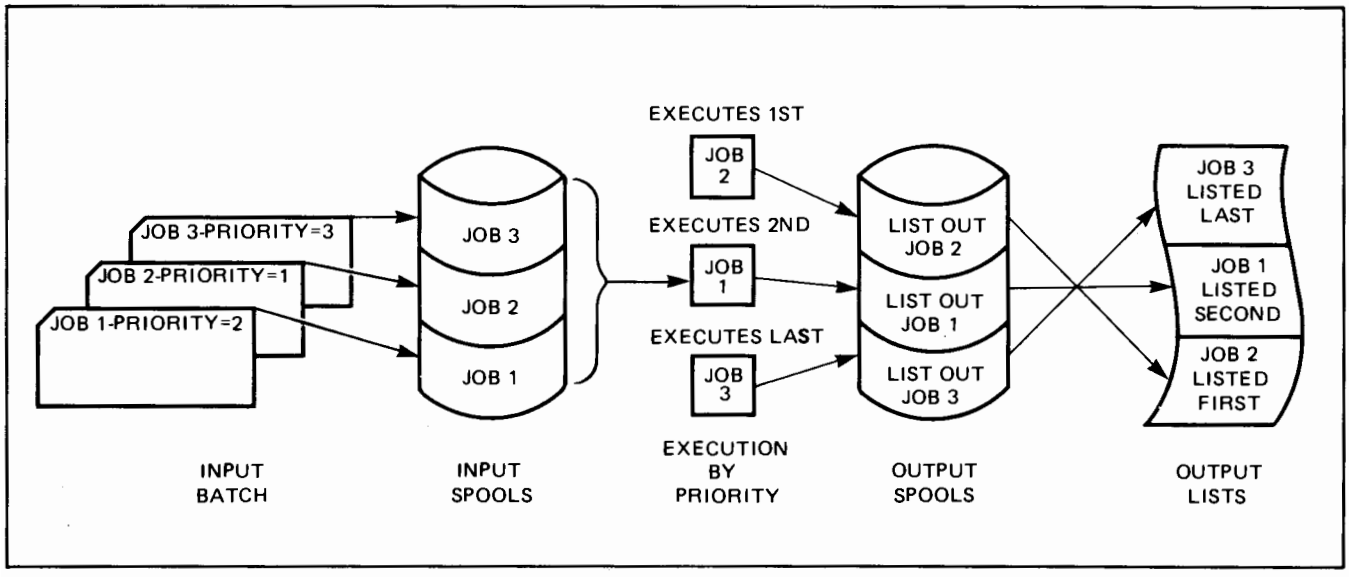


Figure 1-2. Example of Spooling

batching, the programmer does not have to interact with the system. He can submit his job, return to other work, and retrieve the output later. The job that the programmer submits may contain all of the job control commands, as well as the program source data and input data. (Refer to Figure 1-3A.) However, the job may contain simply his source, data, and a directive to execute standard job commands stored in a permanent file. (Refer to Figure 1-3B.) This file typically would contain job commands to run the appropriate compiler, output to load-and-go, run the loader, and then run the program.

Even if jobs are not batched, but rather are run singly and at random times by programmers, time is saved by not requiring the programmer to interact with the system. It is also possible to control access to the computer by requiring users to submit their work in the form of jobs to a single person who has access to the machine. This method is particularly useful where the system is actively doing data collection or other real-time functions and it would be chaotic or dangerous to allow a multitude of programmers to interact continually with the system to accomplish program development and other non-real-time tasks.

Job batching may be done either with or without spooling. Job batching without spooling can be accomplished with the File Management Package. However, job batching with spooling requires the complete Batch-Spool Monitor (FMP plus the Spool Monitor).

Procedures for using the batch processing capability, both with and without spooling, are described together with examples in Appendix L.

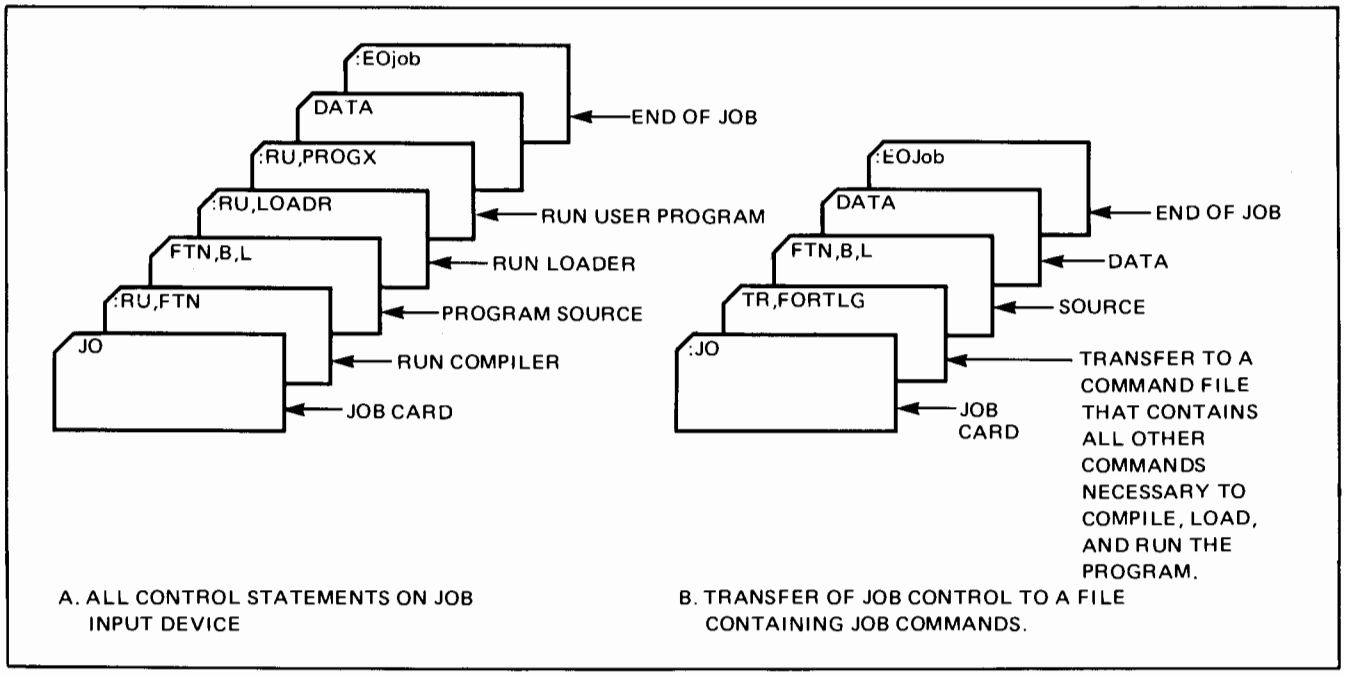


Figure 1-3. Comparison of Methods for Introducing Job Commands

SECTION II
FMP CALLS

INTRODUCTION

This section describes the basic format of HP Assembly Language calls and FORTRAN calls to the various subroutines that interface the user program to the File Management Package (FMP). Table 2-1 is a summary of the calls in this section and shows an interrelationship between the FMP, program calls, Data Control Block (DCB), and user's program. The table is intended as an aid in understanding which calls to use for a given task and shows how the DCB will be affected by the call. The error codes associated with the FMP calls are listed in Section IV. (Note that all error codes are also returned in the computer's A-Register.) Refer to Appendix C for a summary of the FMP calls and required parameters.

SUBROUTINE STRUCTURE

All the interface routines are type 6 or 7 utility subroutines and are located on a single library tape. Type 7 utility subroutines are subroutines which cannot be shared by several programs because of internal design or I/O operations. A copy of the utility routine is appended to every program that calls for it. The RTE library subroutine FRMTR, which carries out FORTRAN I/O operations, and the PAUSE subroutine are examples of type 7 utility routines. Type 6 re-entrant or privileged library routines are sharable and may be loaded in the core resident library area at system generation. The FTN IV Formatter is an example of a re-entrant routine, and .ENTR (the subroutine entry routine) is an example of a privileged routine.

When system generation creates the disc-resident RTE System, all library subroutines not included in the resident library are stored on the disc in relocatable format as utility routines. They are appended to programs by the RTE Relocating Loader in its background loading process (see "RTE Relocating Loader" in the RTE Operating System Manual).

DEFINITIONS

DATA CONTROL BLOCK (DCB)

The DCB is an array provided by the user program with one DCB required for each file opened. Once a file is open, the DCB is used to reference the file, the name no longer being needed, or used. The DCB includes a 16-word directory area and a buffer area that is used for data transfers. The DCB buffer contains a multiple of 128 words.

BLOCK

A block is two sectors of 64 words each, totalling 128 words.

SECTOR

The RTE moving head disc driver regards a track as having a user defined number of sectors of 64 words each. For programming purposes, all disc transfers are a maximum of the DCB buffer size per transfer, except for type 1 files. (Type 1 files do not use the DCB buffer area and are therefore not restricted to its buffer size.)

DISC DIRECTORY

The first two sectors of the last track of the system disc (LU2) are devoted to the master index, or disc directory. The disc directory is always located on the system disc, and contains information on all discs which are currently mounted, and which contain tracks assigned to the FMP. This directory also contains a setup code word and the master security code. The layout or format of this directory is shown in Appendix A.

FILE DIRECTORIES

Each disc which has tracks assigned to the FMP will contain a file directory. Each file directory entry uses 16 words of a

Table 2-1. FMP Calls and Data Control Block Relationship

Subroutine Entry Point	Function	DCB Status At Entry	DCB Status After Call	Directory Access
APOSN	Positions a file to a known record address	MBO	OPN	EXTENTS
CLOSE	Closes a file	MBO	CLO	YES
CREAT	Creates a file	CBO	OPNEX	YES
FCONT	Sends RTE control request to type 0 files	MBO	OPN	NO
FSTAT	Returns 125 words of Disc Directory			
IDCBS	Determines the size of the DCB buffer used	MBO	OPN	NO
LOCF	Returns File Status	MBO	OPN	NO
NAMF	Renames specified file	CBO	CLO	YES
OPEN	Opens desired file	CBO	OPN	YES
POSNT	Directs next READ/WRITE to a specific record	MBO	OPN	EXTENTS
POST	Flushes DCB buffer to disc (if needed) and clears in-core flags	MBO	OPN	NO
PURGE	Removes file and directory entry	CBO	CLO	YES
READF	Transfers one record from file to user buffer	MBO	OPN	EXTENTS
RWNDF	Resets file to first record; if type 0 file, a rewind request is generated	MBO	OPN	EXTENTS
WRITF	Transfers one record from user buffer to file	MBO	OPN	EXTENTS

Legend:

MOB = Must be open
OPN = Open
CBO = Can be open—if DCB is open, the current file opened to the DCB will be closed.
CLO = file is closed.
OPNEX = Open exclusively, update mode
EXTENTS = Directory is accessed if there is a change of extents as a result of the call.

Summary:

- As indicated from the table a file must be open to a DCB before making any of the following calls.

APOSN	FCONT	POSNT	RWNDF
CLOSE	IDCBS	POST	WRITF
	LOCF	READF	

- The only calls that result in a closed DCB being left open are OPEN and CREAT.
- The above conditions imply that the general structure of a program using files be as follows:

A. To open a DCB, call { OPEN, CREAT }

B. To do something with a file, call { APOSN, FCONT, LOCF, POSNT, IDCBS, READF, RWNDF, WRITF, POST }

C. To close a file, call CLOSE
—OR—

D. To close a file and open another, call { OPEN, CREAT }

- A file should always be closed before program termination. Failure to do so may result in the last block not being posted on the disc.

sector. The file directory starts in sector 0 of the last track which is available to the FMP. The first entry in the file directory will pertain to the disc itself and will contain label and track information. Each subsequent entry will contain information on a file. The last entry will be followed by a zero word. The first entry and each entry for purged files, has the sign bit set on its first word to indicate that it is a label and not the same name as a file (normally to be ignored). Refer to Appendix A for the file directory format. Note that while it is possible to have two files with the same name located on different cartridges, it is not possible to have two files with the same name on the same cartridge.

D.RTR

D.RTR is a program which is an integral part of the FMP. D.RTR is the only program which is allowed to write on the directories, and may only be scheduled by some other program. D.RTR is never directly called by the user, it is called by the user interface subroutines only. D.RTR is called by a "schedule-with-wait" call to the RTE EXEC, and five parameters define the function desired. D.RTR will return up to five parameters which the calling routine may access with RMPAR.

LOGICAL READ VS. PHYSICAL READ

A logical read occurs each time the user requests a record from a type 2 and above file. At that time FMP checks the specified Data Control Block (DCB) buffer area to determine if the requested record is already in core. If in core, the record is transferred to the user's record buffer without actually physically reading the disc. If the record is not present in core, the necessary disc transfers are performed (physical reads – and writes, if necessary) to bring the record into core.

LOGICAL WRITE VS. PHYSICAL WRITE

A logical write occurs each time a user requests that a record be written to a type 2 and above file. At that time, FMP determines if that record is present in the DCB buffer area; if it is, FMP simply transfers the data in the user's record buffer to the DCB buffer area and flags it as "must be written." Each succeeding read or write is treated in the same manner until one of the following occurs:

a. A logical record transfer occurs for which the record is not wholly in core.

b. The file is closed.

c. The file is repositioned to another location outside of the DCB buffer area.

d. The contents of the DCB buffer are posted to disc with the POST call.

In any one of these cases, the FMP must physically write on the disc (post) the records in the DCB buffer area. Following this posting, any remaining words are moved to the DCB buffer area.

If the record is not present in core on a write request, FMP locates the record on the disc and if update mode, transfers it physically into the DCB buffer area. The data to be written is then transferred from the user buffer to the DCB buffer area and flagged as "must be written." The read before write is necessary because records do not necessarily fall on sector boundaries on the disc. If a CLOSE, POSNT, POST, APOSN, or RWNDF request occurs, all buffers flagged are written to the disc.

NOTE

Type 0 and 1 files are transferred directly to or from the I/O device or disc (posted immediately).

PARAMETERS

Most of the 15 subroutine calls shown in Table 2-1 use common parameters. Rather than list the meanings of these parameters 15 times, they are described here in detail as a preface to all the calls. Any parameter that is unique to a call is described within the call itself.

GENERAL PARAMETERS

IDCB is a 144-word (or larger) array to be used as a Data Control Block (DCB) for each file opened. The DCB must be at least 144 words long, consisting of a 16-word control area and a 128-word buffer area. For faster processing, a larger buffer can be used. The size available is specified in the OPEN or CREAT requests. (If not given, 144 is assumed.) The buffer size actually used is the largest number which satisfies the following requirements.

a. Buffer Size (BS) = 128 x n where n is an integer greater than zero.

b. BS ⩽ given buffer size
c. m x BS = file size in words where m is an integer greater than zero.

All transfers to or from the disc transfer the full buffer area. While a file may be created using a large buffer, it may be accessed with any DCB (that is, a 144-word DCB is large enough to read any file). In the calls described in this section, the DCB is described as being 144 + n words, where n represents additional buffer area as desired. Note that the DCB is free for other use after a PURGE, CLOSE, and NAMF call. (The DCB is fully defined in Appendix A.) Once a file is open, the DCB is used to reference the file, the name no longer being needed or used.

NOTE

Since the DCB's exist in the user area and are not protected, caution must be taken not to modify them in any way.

IERR When an error occurs, its code is returned in IERR (see Table 4-1 for error codes). For successful OPEN calls, the file type is returned in IERR. For successful CREAT calls, the number of *sectors* is returned in IERR. Note that one-half of this number is the number of blocks in the file (i.e., there are two sectors in one block). This number is only useful if the CREAT call specified the rest of the disc (see CREAT) but is always returned. IERR is also returned to the A-Register.

NAME is a six-character name array for or of the file. A legal file name is six ASCII characters in length. The first character must not be a blank or a number. All characters must come from the set "blank" thru "_"* (see Appendix E) excluding special characters "+" (plus), "–" (minus), "," (comma), and ":" (colon). Imbedded blanks are not allowed. Short (less than 6 characters) names must be padded with trailing blanks. In the FORTRAN calling sequence, *name* must be converted from ASCII to octal and stored in the NAME array. Refer to the table in the RTE-II Manual for the ASCII to octal conversion. If FORTRAN IV is used, *name* can be written using Hollerith constants. Refer to the FORTRAN IV Manual.

IBUF is the read/write array. This parameter defines the buffer address to/from which reads/writes are to take place.

OPTIONAL PARAMETERS

Most of the subroutines have one or more optional parameters. These parameters always appear at the end of the calling sequence. If it is desired to use an optional parameter which is preceded by other optional parameters, then all parameters up to the desired parameter must be supplied. In general, unsupplied optional parameters are assumed to be zero. In FORTRAN calls, optional parameters are underlined.

The two most commonly used optional parameters are the security code (ISECU) and cartridge label (ICR). These are defined as follows:

ISECU The security code can be any ASCII character. If numeric characters are used, the largest allowable number is 32767. One or two non-numeric characters may be specified as a security code; the system converts the characters to their integer equivalent. When a code is not specified, it defaults to zero. The security code parameter, if given in the call, must match the file's code when truncating, renaming, or purging the file. The security code parameter is defined as follows:

ISECU<0. File is protected and cannot be opened without the correct security code.

ISECU=0. File is not protected and can be read or changed by using any or no security code.

ISECU>0. file is write protected but can still be read without security code. The 2's complement of a positive code, as well as the positive code, will open the file for both read and write.

*On some devices, the underscore (_) prints as a left-pointing arrow (←).

ICR The ICR parameter has three meanings as shown below:

ICR>0. Directs file to the cartridge with the positive cartridge reference number (CR). CR is a numeric identifier assigned to all cartridges in the system. A listing of the cartridge directory (CL Operator Command) would reveal a number under the heading CR. This is the cartridge label and is directly associated with a logical unit number.

ICR = 0. Directs file to all cartridges or, for CREAT calls, the first cartridge found with enough room. For OPEN PURGE, and NAMF calls, 0 causes the first file found with the required name to be used. Note that cartridges are searched in the order they appear in the disc directory. For more information on this order, refer to the DC Command in Section III.

ICR<0. Directs file to a disc logical unit number (e.g., ICR = −14 means logical unit number 14).

APOSN

Purpose:

This routine positions any file (except type 0) to a specified record number.

Assembly Language Call:

```
                EXT     APOSN
                 .
                 .
                 .
                JSB     APOSN   Subroutine call
                DEF     RTN     Return address
                DEF     IDCB
                DEF     IERR
                DEF     IREC
                DEF     IRB     Optional
                DEF     IOFF    Optional
RTN             return point    A-Register = IERR
                 .              B-Register = UNDEF
                 .
                 .
IDCB    BSS     144 + n         DCB buffer.
IERR    BSS     1               Error code returned here.
IREC    DEC     a               Record number of next record.
IRB     DEC     b               Relative block address of next record.
IOFF    DEC     c               Block offset of next record.
```

FORTRAN Call:

```
DIMENSION IDCB (144 + n)
IREC  =         a       Record number of next record.
IRB   =         b       Relative block address of next record (optional).
IOFF  =         c       Block offset of next record (optional).
CALL APOSN (IDCB, IERR, IREC, IRB, IOFF)
Error code is returned in IERR.
```

COMMENTS

- If a type 1 or 2 file is being positioned, the optional IRB and IOFF parameters are not required. Actually, for type 1 or 2 files, APOSN is not needed (see READF, WRITF) for positioning. However, APOSN will function correctly to allow sequential access of any cartridge file from the specified record.
- Block offset is the relative word within the block ($0 \leq IOFF < 128$) at which the record begins.
- APOSN allows random access of sequential files, typically type 3 files. The parameters needed for APOSN are relative and are not affected by packing; they are obtained using a LOCF call.
- APOSN parameters are assumed to be normalized to a 128-word DCB buffer size. However, they may have been obtained when the file was opened with a DCB of a different size (see LOCF) and are unnormalized as required for the DCB size currently in use.

SEQUENCE OF OPERATIONS

1. If DCB is not open, reject call.
2. Check if file type = 0.

3. Check for enough parameters.
4. If type 1 or 2, go to 7.
5. Call subroutine LOCF to get the current IRB.
6. Position to the new block without reading file.

   a. May imply an extent change.

   b. Implies writing current block if written on, before positioning to a new block.
7. Set current buffer pointer.
8. If record number is less than 1, error exit.
9. Record number is returned.
10. Return.

CLOSE

Purpose:

This routine closes the DCB and makes the file available to other callers. CLOSE also optionally truncates the file size.

Assembly Language Call:

```
                EXT     CLOSE
                 .
                 .
                 .
                JSB     CLOSE   Subroutine call
                DEF     RTN     Return address
                DEF     IDCB
                DEF     IERR    Optional
                DEF     ITRUN   Optional
RTN             return point    Continue execution.
                 .
                 .
                 .
IDCB            BSS     144 + n DCB buffer.
IERR            BSS     1       Error code returned here.
ITRUN           DEC     m       +m = number of blocks to be
                                    deleted from the end of
                                    the file when it is closed.

                                -m = retain main file, delete extents.

                                m = Ø  standard close.
```

FORTRAN Call:

```
DIMENSION IDCB (144 + n)
ITRUN = m                       +m    number of blocks to be
                                      deleted from the end of
                                      the file when it is closed.

                                -m =  retain main file, delete extents.

                                m = Ø standard close.

CALL CLOSE (IDCB, IERR, ITRUN)
Error code is returned in IERR.
```

COMMENTS

- The following conditions must be met for the parameter ITRUN to be recognized:

  a. The file must be opened exclusively.
  b. The current position must be in the main file (not an extent).
  c. Security codes must have matched at open.
  d. The file type must not be equal to zero.
  e. The number of blocks to be deleted must be less than or equal to the number of blocks in the file.

NOTE

If the number of blocks to be deleted equals the number of blocks in the file, then the file is purged.

EXAMPLE OF TRUNCATION

In this example, a file was created and several records were written. The file may then be truncated as follows:

```
CALL LOCF (IDCB,IERR,IREC,IRB,IOFF,JSEC)
ITRUN=JSEC/2 -IRB -1
CALL CLOSE (IDCB,IERR,ITRUN)
```

SEQUENCE OF OPERATIONS

1. Check for enough parameters.
2. If DCB is not open, reject call.
3. If block currently in core was written on, write it to the file.
4. Check file type, current extent, and security for truncate option.
5. Call D.RTR to close the file (D.RTR makes the rest of the truncate checks).
6. Return.

CREAT

Purpose

This routine closes the DCB (if open), and then creates the named file on the specified disc with the specified number of blocks. The file is left open in update mode exclusively to the caller on successful completion of the call. This call will not create a type 0 file.

Assembly Language Call:

```
            EXT     CREAT
             .
             .
             .
            JSB     CREAT     Subroutine call
            DEF     RTN       Return address
            DEF     IDCB
            DEF     IERR
            DEF     NAME
            DEF     ISIZE
            DEF     ITYPE
            DEF     ISECU     Optional
            DEF     ICR       Optional
            DEF     IDCBS     Optional
RTN         return point      continue execution
             .
             .
             .
IDCB        BSS     144 + n   DCB Buffer.
IERR        BSS     1         Error code—or number of sectors (twice the number of blocks)
                              if CREAT successful.
NAME        ASC     3,name    File name.
ISIZE       DEC     a         Number of blocks in the file. If negative, use all of disc
                              (see Comments).
            DEC     b         Record length (used for type 2 file only).
ITYPE       DEC     c         File type (see Comments).
ISECU       DEC     d         Security code.
ICR         DEC     e         Cartridge label.
IDCBS       DEC     f         Size of DCB buffer (number of words available).
```

FORTRAN Call:

```
DIMENSION IDCB (144 + n), NAME(3), ISIZE(2)
NAME(1) = xxxxxB      First two characters
NAME(2) = xxxxxB      Second two
NAME(3) = xxxxxB      Last two characters

ISIZE(1) = a          Number of blocks in the file. If negative, use all of disc.
ISIZE(2) = b          Record length (used for type 2 file only).
ITYPE    = c          File type.
ISECU    = d          Security code (optional).
ICR      = e          Cartridge Label (optional).
IDCBS    = f          Size of DCB (number of words available) (optional).
CALL CREAT (IDCB,IERR,NAME,ISIZE,ITYPE,ISECU,ICR,IDCBS)
Error code, or number of sectors, is returned in IERR.
```

COMMENTS

- The ISIZE parameter is a two-word integer array. The first word should contain the number of blocks desired in the file. If the number is negative, the rest of the disc will be used. Unused area may be returned with the CLOSE call, ITRUN parameter. The second word will be stored in the directory as the record size. It is used as the record size only if you are creating a type 2 file.

- Since all type 2 files are addressed by record number, the maximum number of records allowed in a type 2 file is limited to 32767. That is:

$$\frac{\text{number of blocks} * 128}{\text{record size}} \text{ must be } < 32768$$

- The ITYPE parameter refers to the file type and is defined as follows:

  1 – 128 word record length, random access.

  2 – user selected record length, random access.

  3 – (and greater) random record length, sequential access.

    4 – source program.
    5 – relocatable program.
    6 – RTE load module.
    7 – absolute program.
  >7 – user defined.

Note that the FMP will recognize only the above; however, the user may define any number of additional types. The FMP will treat these as type 3.

- If type 3 or greater, an EOF mark is written at the beginning of the file.

SEQUENCE OF OPERATIONS

1. Check for enough parameters.
2. Close the DCB using CLOSE (ignore not open error).
3. Check legality of NAME.
4. Check legality of ITYPE.
5. Check legality of ISIZE, and if ITYPE = 2, then check that no more than 32767 records will fit in allotted size. (This is to prevent the file from containing more records than possible to address.) If ITYPE = 1, force record size to 128.
6. Get a system track and write skeleton entry on the track.
7. Schedule D.RTR to create the file.
8. Return the track.
9. Check for D.RTR error; if any, exit.
10. Open the file to the DCB.
    a. Read the directory entry to the DCB buffer area.
    b. Transfer directory parameters to the DCB.
    c. Set current position pointers in the DCB.
    d. Set update mode bit in DCB.
11. If file type ≥ 3, set "written-on-flag" and EOF in DCB.
12. Return.

FCONT

Purpose:

This routine is used to control a peripheral device; it sends the standard RTE I/O Control request to type 0 (non-disc) files. It has no effect on other files.

Assembly Language Call:

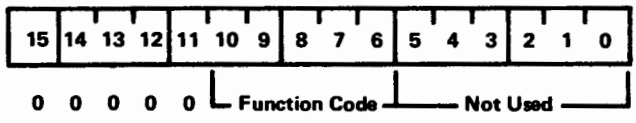
```
                EXT     FCONT
                 .
                 .
                 .
                JSB     FCONT     Subroutine call
                DEF     RTN       Return address
                DEF     IDCB
                DEF     IERR
                DEF     ICON1
                DEF     ICON2     Optional
RTN             return point      Continue execution.
                 .
                 .
                 .
IDCB            BSS     144 + n   DCB buffer.
IERR            BSS     1         Error code or file type returned here.
ICON1           OCT     conwd     See Control Word.
ICON2           DEC     m         Required for some functions (see Control Word).
```

FORTRAN Call:

```
DIMENSION IDCB (144 + n)
ICON1 = aB   Control word.
ICON2 = b    Control word (optional).
CALL FCONT (IDCB, IERR, ICON1, ICON2)
Error code is returned in IERR.
```

CONTROL WORD

- Function Code 11 (octal) (list output line spacing), requires the optional parameter ICON2. ICON2 must designate the number of lines to be spaced on the specified logical unit. A negative parameter specifies a page eject on a line printer. For further details of line printer formatting consult the appropriate line printer manual.
- The control word value (*conwd*) has one field that specifies the function code; the preceding bits must be zero. Succeeding bits (5–0) may contain any value.



- An EOF error code (−12) is retuned in the function code if bit 7 (EOF bit) is set in the device status.

Function Code (Octal)	Action
00	Unused
01	Write end-of-file (mag tape)

02	Backspace one record (mag tape)
03	Forward space one record (mag tape)
04	Rewind (mag tape)
05	Rewind standby (mag tape)
06	Actual status of device (mag tape)
07	Set end-of-paper tape
10	Generate paper tape leader
11	List output line spacing
12	Write gap in case of error (mag tape)
13	Forward space one file (mag tape)
14	Backward space one file (mag tape)
15	Conditional top-of-form

- The following functions are defined for DVR00, DVR01 and DVR02:

  20 Enable terminal—allows terminal to schedule its program when any key is struck.

  21 Disable terminal—inhibits scheduling of terminal's program.

  22 Set time out—the optional parameter is set as the new time out interval.

  23 Ignore all further action requests until:

  a) The request queue is empty
  b) An input request is received
  c) A restore control request is received.

  24 Restore output processing (this request is usually not needed).

SEQUENCE OF OPERATION

1. If DCB is not open, reject call.
2. Check if file type = 0.
3. Issue control EXEC call.
4. Return.

FSTAT

Purpose:

This routine returns information on all cartridge labels in the system.

Assembly Language Call:

```
        EXT     FSTAT
        .
        .
        .
        JSB     FSTAT   Subroutine call
        DEF     RTN     Return address
        DEF     ISTAT
RTN     return point    Continue execution.
        .
        .
        .
ISTAT   BSS     125     Buffer of 125 words.
```

FORTRAN Call:

```
DIMENSION ISTAT (125)
CALL FSTAT (ISTAT)
```

COMMENTS

- The cartridge status is contained in four-word increments in ISTAT:

Word	Meaning
0	Logical unit number (first disc).
1	Last track for FMP.
2	Cartridge label.
3	Locking program's ID segment address, or 0 if not locked.
4	Logical unit number (second disc).
.	
.	
.	
124	0

NOTE

This list is terminated with a zero.

- The same information can be obtained as a list file using the FMGR cartridge list command (CL).

SEQUENCE OF OPERATION

1. Read directory of disc to ISTAT.
2. Return.

IDCBS

Purpose:

This routine determines the size of the DCB buffer area actually used.

Assembly Language Call:

```
            EXT     IDCBS
             .
             .
             .
            JSB     IDCBS   Subroutine call
            DEF     RTN     Return address
            DEF     IDCB
RTN         return point    Continue execution
             .
             .
             .
IDCB        BSS     144 + n DCB buffer
```

FORTRAN Call:

```
DIMENSION IDCB(144 + n)
ISIZE = IDCBS(IDCB)
```

COMMENTS

- The IDCBS routine is intended for use by a program which has opened a file nonexclusively or created a file using the DCB size option and needs to find out how the DCB size has been set up. (Refer to "IDCB" under "General Parameters" at the beginning of this section.)
- The number of words actually used of the DCB buffer area is returned in the A-Register.

or

- When this routine is used as a function in FORTRAN, ISIZE = IDCBS(IDCB), the number of words used is returned in ISIZE.
- Any leftover area may be used for another file's DCB.

SEQUENCE OF OPERATIONS

1. Determine if DCB is open.
2. Extract the DCB size.
3. Return.

LOCF

Purpose:

This routine formats and returns location and status information from the DCB.

Assembly Language Call:

```
            EXT     LOCF
             .
             .
             .
            JSB     LOCF    Subroutine call
            DEF     RTN     Return address
            DEF     IDCB
            DEF     IERR
            DEF     IREC
            DEF     IRB     Optional
            DEF     IOFF    Optional
            DEF     JSEC    Optional
            DEF     JLU     Optional
            DEF     JTY     Optional
            DEF     JREC    Optional
RTN         return point    Continue execution.
             .
             .
             .
IDCB        BSS     144 + n DCB buffer.
IERR        BSS     1       Error code returned here.
IREC        BSS     1       Next record number.
IRB         BSS     1       Relative block of next read.
IOFF        BSS     1       Block offset of next record.
JSEC        BSS     1       Number of sectors in the file.
JLU         BSS     1       File logical unit.
JTY         BSS     1       File type.
JREC        BSS     1       Record size.
```

(IERR, IREC, IRB, IOFF, JSEC, JLU, JTY, JREC: See Comments)

FORTRAN Call:

```
DIMENSION IDCB (144 + n)
CALL LOCF (IDCB, IERR, IREC, IRB, IOFF, JSEC, JLU, JTY, JREC)
```

Error code returned in IERR
Next record number returned in IREC
Relative block of next read returned in IRB
Block offset of next record returned in IOFF
Number of sectors in the file returned in JSEC
File logical unit returned in JLU
File type returned in JTY
Record size returned in JREC

COMMENTS

- A further elaboration on the parameters follows:

  IREC is the number of the next record.

  IRB is the relative block number of the next record (same as IREC for type 1 files). IRB is not set for a type 0 file.

  NOTE

  IRB includes extent information (i.e., IRB = JSEC/2 X Extent-number + relative block in current extent).

  IOFF is the relative word position of the next record within the physical block (IOFF=0 for type 1 files). IRB and IOFF need not be coded for type 1 files, or for other file types if the information is not required. IOFF is not set for type 0 files. If used, IOFF must be in the range:

  $$0 \leqslant IOFF < 128$$

- IRB and IOFF together give the actual physical location of the record pointer in the file.

  JSEC is the number of *sectors* in the main file. JSEC is not set for type 0 file.

  JLU is the logical unit the file is on (disc or non-disc).

  JTY is the file type as set in DCB (will be 1 if file was forced to type 1 at open).

  JREC is the record size word. This will be as it is in the directory entry except for type 1 (actual or forced) files, in which case it is set to 128 on the first read/write access. If file is type 0, then JREC will be the read/write code. Bit 15 = 1 implies read legal; bit 0 = 1 implies write legal.

- LOCF normalizes its return parameters to a DCB buffer size of 128 words so that they may be used by APOSN on a subsequent file access with a different DCB size.

SEQUENCE OF OPERATIONS

1. Check for enough parameters.
2. Fetch DCB.
3. Check that file is open.
4. Set IREC.
5. If type 0, go to 8.
6. If type 1 or 2, compute current address from the record number.
7. Compute and set IOFF, IRB, JSEC.
8. Set JTY, JLU, JREC.
9. Return.

NAMF

Purpose:

This routine closes the DCB (if open) and then renames the specified file.

Assembly Language Call:

```
                EXT     NAMF
                .
                .
                .
                JSB     NAMF        Subroutine call
                DEF     RTN         Return address
                DEF     IDCB
                DEF     IERR
                DEF     NAME
                DEF     NNAME
                DEF     ISECU       Optional
                DEF     ICR         Optional
RTN             return point        Continue execution.
                .
                .
                .
IDCB            BSS     144 + n     DCB buffer
IERR            BSS     1           Error code returned here.
NAME            ASC     3,name      File's present name.
NNAME           ASC     3,nuname    File's new name.
ISECU           DEC     a           Security code.
ICR             DEC     b           Cartridge label.
```

FORTRAN Call:

```
DIMENSION IDCB (144 + n), NAME(3), NNAME(3)
NAME(1)     = xxxxxB    Present name, first two characters
NAME(2)     = xxxxxB    Second two
NAME(3)     = xxxxxB    Last two characters
NNAME(1)    = xxxxxB    New name, first two characters
NNAME(2)    = xxxxxB    Second two
NNAME(3)    = xxxxxB    Last two characters
ISECU       = a         Security code (optional)
ICR         = b         Cartridge label (optional)
CALL NAMF (IDCB, IERR, NAME, NNAME, ISECU, ICR)
Error code is returned in IERR
```

COMMENTS

- The file specified by *name* must not be open when this call is made. If *name* is a non-zero security, the proper security code must be included. If ICR is furnished, only that cartridge is searched for *name*; otherwise, all mounted cartridges are searched. In the case where all cartridges are searched, only the first file found with the given *name* will be changed.

- *nuname* is the new name of the same file.

SEQUENCE OF OPERATIONS

1. Check for enough parameters.
2. Check legality of *nuname*.
3. Call OPEN to open the file exclusively.
4. If OPEN errors, exit.
5. Check file security flag in DCB; if mismatch, close and exit.
6. Get a system track.
7. Write new name on track.
8. Pass track to D.RTR; rename request.
9. Return system track.
10. Close the DCB.
11. Check for D.RTR errors.
12. Return.

OPEN

Purpose:

This routine closes the DCB (if open), and then opens the named file.

Assembly Language Call:

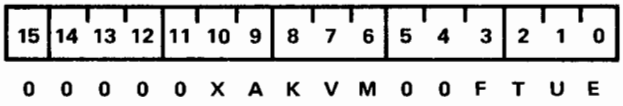
```
                EXT     OPEN
                 .
                 .
                 .
                JSB     OPEN    Subroutine call
                DEF     RTN     Return address
                DEF     IDCB
                DEF     IERR
                DEF     NAME
                DEF     IOPTN   Optional
                DEF     ISECU   Optional
                DEF     ICR     Optional
                DEF     IDCBS   Optional
RTN             return point    Continue execution.
                 .
                 .
                 .
IDCB    BSS     144 + n         DCB buffer
IERR    BSS     1               Error code, or file type if OPEN successful.
NAME    ASC     3,name          File name.
IOPTN   OCT     a               Open options (see Comments).
ISECU   DEC     b               Security code.
ICR     DEC     c               Cartridge label.
IDCBS   DEC     f               Size of DCB buffer (number of words available)
                                (optional).
```

FORTRAN Call:

```
DIMENSION IDCB (144 + n), NAME(3)
NAME(1)   =xxxxxB     First two characters
NAME(2)   =xxxxxB     Second two
NAME(3)   =xxxxxB     Last two characters
IOPTN     =a          Open control word
ISECU     =b          Security code (optional)
ICR       =c          Cartridge label (optional)
IDCBS     =f          Size of DCB buffer (number of words available) (optional).
CALL OPEN (IDCB,IERR,NAME,IOPTN,ISECU,ICR,IDCBS)
Error code is returned in IERR
```

COMMENTS

- The IOPTN (open option) parameter is defined as follows:



Where

E = 0 for exclusive open.

E = 1 for non-exclusive open.

U = 0 for non-update open.

U = 1 for update open.

T = 0 means file type defined in directory.

T = 1 means force to file type 1 (extents are not addressable).

F = 0 uses function code defined in directory.

F = 1 uses function code defined in bits 6-10 of this word for type 0 file only.

The following codes are used only if F = 1 and the file type is 0.

M = 0 for ASCII.

M = 1 for binary.

V = 1, and M = 1, causes the length of punched tape input to be determined by the word count in the first non-zero character read from the tape.

V = 0, and M = 1, the length of the punched tape input is determined by the buffer length specified in the call.

M determines the mode of the data transfer (if applicable).

K = 1 causes keyboard input to be printed as received. If K = 0, input from the keyboard is not printed.

A = 1 designates punching ASCII characters on the teleprinter (M = 0). ASCII is usually printed; but since it is sometimes desirable to punch ASCII tapes, this option is provided. (If A = 0, M determines whether punch or printer is used.)

X = When paper tape devices are used, "X," in combination with "M" and "V" will indicate an honesty mode.

On input, if "X," "M," and "V" are set, absolute binary tape format is expected and handled. If "X" and "M" are set, and "V" is not, leader is not skipped and the specified number of words are read. On output, the record terminator (usually four feed frames) is not punched.

On input, if "X" is set and "M" is not, ASCII tape format is expected. Leader is not skipped, bit 8 is stripped, but otherwise, all characters are passed to the user's buffer. The only exception is line-feed, which terminates the record. On output, carriage return and line-feed are suppressed; any trailing underscore (_)* is not (i.e., underscore is transmitted but carriage return/line feed is not).

UPDATE OPEN

Update implies that a block is read before it is modified so that existing records within the block will not be destroyed. Type 1 files are inherently update files. In type 1 and 2 files the end-of-file is the last word of the last block in the file. However, in other type files it is a special word (–1) for the length of a record. In order to preserve data in a type 2 file, it should be opened for update whenever any non-sequential accesses are to be performed, and the file is to be modified. That is to say, a type 2 file should be modified in non-update mode only when originally writing the file, or adding to the end of the file; and then only if it is to be written sequentially. Update/non-update has no effect on reading or positioning.

In type 3 and above files, an end-of-file mark is written after each record that is written in the non-update mode. In either mode (update/non-update), records written beyond

*On some devices, the underscore (_) prints as a leftpointing arrow (←).

an end-of-file are written in non-update fashion. That is, replace the end-of-file with the new record, followed by an end-of-file.

EXCLUSIVE OPEN

In order to prevent two or more programs from destructively interfering with each other, a file may be opened either exclusively or non-exclusively. An exclusive open is granted only to one program at a time. If the call is rejected because another program has the file open, you must make the call again; it is not stacked by the FMP.

NON-EXCLUSIVE OPEN

A non-exclusive open may be granted to as many as seven programs per file at one time. A non-exclusive open will not be granted if the file is already opened exclusively. Each time an open is requested for a file, all programs currently having that file open will be checked. If a program is dormant, the file will be closed to that program. This type of close will not post the packing buffer nor will it close the DCB. This type of close is required in case of a program being aborted or otherwise terminating without closing a file. Any open flag will also be cleared if it does not point to a valid ID segment. (Possible if the file was left open on a different system.)

SEQUENCE OF OPERATIONS

1. Check for enough parameters.
2. Close the DCB using CLOSE (ignore not open error).
3. Format and issue D.RTR request.
4. Call RMPAR to get D.RTR return parameters.
5. If D.RTR error, exit.
6. Open the file to the DCB.

    a. Read the directory entry for the file into the DCB buffer area.
    b. Transfer directory parameters to the DCB.
    c. Set current position pointers in the DCB.

7. If OPEN protected (i.e., security code < 0), and security code mismatches, close the file and exit.
8. If type = 0 and subfunction option present (Bit 3 = 1), replace subfunction.
9. Return.

POSNT

Purpose:

This routine causes the next read or write to access the given record in any file type.

Assembly Language Call:

```
                EXT     POSNT
                 .
                 .
                 .
                JSB     POSNT       Subroutine call
                DEF     RTN         Return address
                DEF     IDCB
                DEF     IERR
                DEF     NUR
                DEF     IR          Optional
RTN             return point        Continue execution.
                 .
                 .
                 .
IDCB            BSS     144 + n     DCB buffer.
IERR            BSS     1           Error code returned here.
NUR             DEC     a           New record number.
IR              DEC     b           Absolute vs. relative control parameter for NUR.
```

FORTRAN Call:

```
DIMENSION IDCB (144 + n)
NUR = a                 NUR > 0     Forward positioning
                        NUR < 0     Backward positioning
                        NUR = 0     No operation.

IR  =b(optional)        IR= 0       The new record indicated by NUR is relative to the present
                          or        position in the file; NUR records skipped.
                        not present

                        IR ≠ 0      The new record indicated by NUR is the absolute number
                                    starting from the first record in the file (record number 1).
                                    Note that NUR must be>0. Position to NUR record.

CALL POSNT (IDCB,IERR,NUR,IR)
Error code is returned in IERR
```

COMMENTS

- The NUR (new record) parameter is defined as follows:

  NUR > 0 Forward positioning.

  NUR < 0 Backward positioning.

  NUR = 0 No operation

  NOTE

  If a type 0 file is involved the motion requested must be legal for the device.

- The IR (absolute vs. relative) parameter is defined as follows:

IR = 0 or not present	The new record indicated by NUR is relative to the present position in the file; skip NUR records.
IR ≠ 0	The new record indicated by NUR is the absolute number starting from the first record in the file (record number 1). Note that NUR must be > 0. Position to NUR record.

- Forward positioning on a type 0 file is accomplished by reading records until one less than the desired record is read, or an EOF is read. In all cases, EOF terminates positioning.

- When backspacing on type 0 files, if the first record backspaced over is an EOF, and the only EOF, no error occurs. If an EOF is encountered anywhere else (i.e., not as the first record), a −12 error will occur, and the call will be terminated after forward spacing to position the file immediately following the EOF.

  Examples (refer to Figure 2-1):

  1. File position is immediately after EOF2 and NUR indicates backspace four records. Final file ends up immediately after EOF1 with no error.
  2. File position is at 2A and NUR indicates backspace five records. Error −12 results, the file position ends up immediately after EOF2, and the call is terminated.
  3. File position is immediately after EOF2 and NUR indicates backspace five records Error −12 results, the file position ends up immediately after EOF1, and the call is terminated.

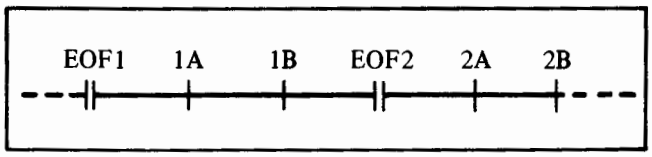


Figure 2-1. Record Positioning Example Using NUR in POSNT

- Type 3 and above files are treated as mag tape files. To be correct, a backspace should be issued after an EOF is read, and before continuing to write. This action simulates a magnetic tape, that is, the EOF is spaced over. It is also legal to continue without backspacing.

SEQUENCE OF OPERATIONS

1. Check for enough parameters.
2. Set reading flag.
3. If DCB is not open, reject call.
4. Compute the relative record number.
5. If type 0, go to 17.
6. If type 1 or 2, go to 14.
7. If forward spacing, call READF to read the required number of records (unless EOF or error).
8. If current position is EOF and EOF read bit is set, clear it and go to 9; else 10.
9. Decrement record number and backspace count; if done, exit.
10. Backspace one word and read it.
11. Backspace over the record.
12. Read length; if no match, error exit.
13. Go to 9.
14. Compute absolute record count; if less than 1, error exit.
15. Set record number.
16. Return.
17. If forward space, go to 7.
18. Check backspace legal flag.
19. Backspace–EXEC call.
20. If EOF encountered and not first backspace, go to 7 with 1 record (READF will return EOF).
21. If not done, go to 19.
22. Return.

POST

Purpose:

This routine writes the contents of the DCB buffer to the disc, if needed, and sets flags indicating no data in core.

Assembly Language Call:

```
                EXT     POST
                 .
                 .
                 .
                JSB     POST        Subroutine call
                DEF     RTN         Return address
                DEF     IDCB
                DEF     IERR        Optional
RTN             return point
                 .
                 .
                 .
IDCB            BSS     144 + n     DCB buffer.
IERR            BSS     1           Error code returned here.
```

FORTRAN Call:

```
DIMENSION IDCB (144 + n)
CALL POST (IDCB,IERR)
Error code is returned in IERR
```

COMMENTS

- The POST routine is used with the RTE system RNRQ request to allow several programs to modify a file without requiring exclusive opens. The suggested sequence is as follows:

1. Call OPEN.
2. Read file to pick up resource number (RN) (it is assumed to be in the file).
3. Call POST to clear in-core flags and force reread below (no data is transferred since none was written).
4. Call RNRQ to lock the file.
5. Read the record to be modified.
6. Modify the record and write it out (the data may or may not be posted).
7. Call POST to post any data not already posted (written to the file).
8. Call RNRQ to unlock the RN.

SEQUENCE OF OPERATIONS

1. Determine that the DCB is open.
2. If the DCB was written into, post it (write it on the disc).
3. Clear the in-core flags and the written-on flags.
4. Return.

PURGE

Purpose:

This routine closes the DCB (if open), and then deletes the named file and all of its extents.

Assembly Language Call:

```
                EXT     PURGE
                 .
                 .
                 .
                JSB     PURGE   Subroutine call
                DEF     RTN     Return address
                DEF     IDCB
                DEF     IERR
                DEF     NAMR
                DEF     ISECU   Optional
                DEF     ICR     Optional
RTN             return point    Continue execution.
                 .
                 .
                 .
IDCB    BSS     144     DCB buffer.
IERR    BSS     1       Error code returned here.
NAME    ASC     3,name  File name.
ISECU   DEC     a       Security code.
ICR     DEC     b       Cartridge label.
```

FORTRAN Call:

```
DIMENSION IDCB (144), NAME(3)
NAME(1)   = xxxxxB    First two characters
NAME(2)   = xxxxxB    Second two
NAME(3)   = xxxxxB    Last two characters
ISECU     = a         Security code (optional)
ICR       = b         Cartridge label (optional)
CALL PURGE (IDCB,IERR,NAME,ISECU,ICR)
Error code is returned in IERR
```

COMMENTS

- The file must not be open to any other program when this call is made.
- This routine will not purge type 0 (non-disc) files.
- If the file being purged is the last file on the disc (excluding a type 6 file), all area from the last unpurged or type 6 file up to and including the purged file, is returned to the system. If a purged file is not the last file then its area may be recovered only by packing, or by purging all files after it in the directory. Type 6 file area may only be recovered by packing.

SEQUENCE OF OPERATIONS

1. Check for enough parameters.
2. Call OPEN to open the file exclusively.
3. If OPEN errors, exit.
4. Check security flag in DCB; if mismatch, close and exit.
5. Close and truncate file to zero length.
6. Return.

READF

Purpose:

This routine reads a record from the file currently open to the DCB, to the user buffer.

Assembly Language Call:

```
        EXT     READF
        .
        .
        .
        JSB     READF     Subroutine call
        DEF     RTN       Return address
        DEF     IDCB
        DEF     IERR
        DEF     IBUF
        DEF     IL        Optional
        DEF     LEN       Optional
        DEF     NUM       Optional
RTN     return point      Continue execution.
        .
        .
        .
IDCB    BSS     144 + n   DCB buffer
IERR    BSS     1         Error code returned here.
IBUF    BSS     IL        Data buffer
IL      DEC     a         Read request length (see Comments).
LEN     BSS     b         Actual read length returned here (see Comments).
NUM     DEC     c         Record number to be read (see Comments).
```

FORTRAN Call:

```
DIMENSION IDCB (144+n), IBUF(m)
IL = m      Length of IBUF (optional)
NUM = x     Record number to be read (optional)
CALL READF (IDCB, IERR, IBUF,IL,LEN,NUM)
Error code is returned in IERR
Actual read length is returned in LEN (optional)
```

COMMENTS

- The IL (request length) parameter takes on different meanings depending on the file-type. Refer to Table 2-2.
- The LEN (actual length) parameter provides the actual number of words transferred to IBUF. LEN can never exceed IL; therefore, if LEN = IL the record may have been too long (unless a type 1 file is being read). If LEN = −1, then an end-of-file has been read.
- NUM is the random access record number and is used only when the file type is 1 or 2. Record number 1 is the first record in a file. If NUM = 0 or is absent, then

the transfer starts at the current record number. If NUM is positive, then the transfer starts at the record number indicated by NUM. If NUM is negative, then the transfer starts at the current record plus NUM. Therefore:

File Type	Meaning
0	Not used.
1	NUM defines the first record transferred, and IL defines the number of records through the length in words.
2	NUM defines the record to be transferred.
3 and above	Not used.

Examples (refer to Figure 2-2):

1. If NUM = 0, transfer starts at record number 5.

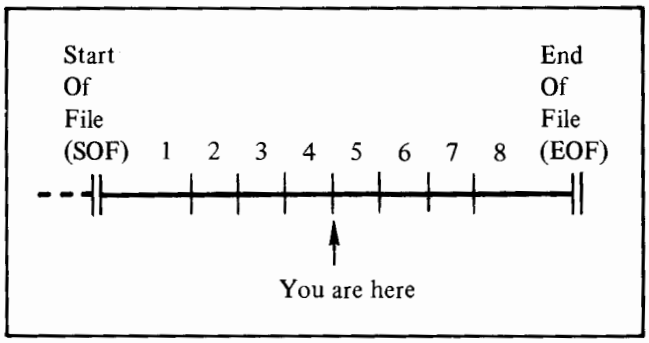


Figure 2-2. Record Positioning Example Using NUM in READF

2. If NUM = 6, transfer starts at record number 6 (same as NUM).

3. If NUM = −3, transfer starts at record number 2 (5 + (−3) =2).

Table 2.2 Read Request File Length (IL) vs. File Types

Request Length / File Type	IL> 0	IL = 0 (Indicates zero length record) (Not recommended)	IL not supplied
0	The routine will transfer up to IL words. If the record length is smaller than IL, only the record is transferred. If the record is the same size, smaller, or larger than IL, no error is indicated.	A zero length record is passed to the device; usually causes a record to be skipped and the record is counted.	A zero length record is passed to the device; usually causes a record to be skipped and the record is counted. (Not recommended)
I	The routine reads exactly IL words. Therefore, since the record length is fixed at 128 words, less than a full record or more than one record may be transferred. All of IL, however, must be within the file.	A zero length read from the disc is done, which results in no position change; the record is not counted.	128-word record is assumed.
> 1	The routine will transfer up to IL words. If the record length is smaller than IL, only the record is transferred. If the record is the same size, smaller or larger than IL, no error is indicated.	Causes a record to be skipped and the record is counted.	Actual record length is used.

- There are three levels of EOF associated with file types that are defined below.

File Type	Meaning
0	When an EOF is read, parameter LEN is set to −1 with no error. Accesses may continue beyond the EOF.
1 and 2	When an EOF is read, parameter IERR is set to −12 indicating an error condition. No accesses beyond the EOF.
3 and above	The first EOF read causes parameter LEN to be set to −1 with no error. A read access beyond this EOF will cause an error condition (IERR = −12). A write access may continue beyond this EOF with no error.

SEQUENCE OF OPERATIONS

1. Set read flag and fetch parameters.
2. Check for enough parameters.
3. If DCB is not open, reject call.
4. If read or update, set reading flag.
5. If type 1, force DCB length to 128.
6. If type 1 or 2, do random access positioning (implies EOF if initial position not in file). If new position not in currently resident block, then write the DCB buffer if it was written on.
7. If type 2 or above, and reading flag set, and DCB buffer is empty, then read to DCB.

8. If type less than 3, skip to 18.
9. If current position is at EOF set EOF encountered flag in DCB. If already set, set IERR = −12 and exit; else step record count, set LEN = −1, and exit.
10. Read the record length.
11. If IL too short, set skip count.
12. Read the record.
13. If skip count set, skip the rest of the record.
14. If type 2, set record count and exit.
15. Read the length word.
16. If lengths do not match, take error exit.
17. If reading flag not set, set EOF in buffer, set written on flag, step record count and exit; else step record count and exit.
18. If type 2, go to 11.
19. If type 0, go to 30.
20. If type 1, round up length to even 238 words and save as increment in record count.
21. Check if request is within the file.
22. If track switch, compute maximum words this access.
23. Read the record.
24. If type 0 read, do EOF test and return.
25. If type 1, check for disc errors.
26. Update for rest of record.
27. If more to transfer, go to 22.
28. Update record count.
29. Return.
30. Test read legality bits.
31. Go to 23.

RWNDF

Purpose:

This routine rewinds type 0 files, or sets disc files so that the next record is the first record in the file.

Assembly Language Call:

```
            EXT     RWNDF
             .
             .
             .
            JSB     RWNDF   Subroutine call
            DEF     RTN     Return address
            DEF     IDCB
            DEF     IERR    Optional
RTN         return point    Continue execution.
             .
             .
             .
IDCB        BSS     144 + n DCB buffer.
IERR        BSS     1       Error code returned here.
```

FORTRAN Call:

```
DIMENSION IDCB (144 + n)
CALL RWNDF (IDCB,IERR)
Error code is returned in IERR
```

SEQUENCE OF OPERATIONS

1. If DCB is not open, reject call.
2. If type 0 file, do EXEC rewind request and return.
3. Write out current block if written on.
4. If not in main file, call D.RTR to get the main file address.
5. Reset current position pointers in DCB to beginning of file.

WRITF

Purpose:

This routine writes a record on the file currently open to the DCB.

Assembly Language Call:

```
                EXT     WRITF
                 .
                 .
                 .
                JSB     WRITF     Subroutine call
                DEF     RTN       Return address
                DEF     IDCB
                DEF     IERR
                DEF     IBUF
                DEF     IL        optional
                DEF     NUM       optional
RTN             return point      Continue execution.
                 .
                 .
                 .
IDCB    BSS     144 + n           DCB buffer.
IERR    BSS     1                 Error code returned here.
IBUF    BSS     IL                Data buffer.
IL      DEC     a                 Write request length (see Comments).
NUM     DEC     b                 Record number to be written (see Comments).
```

FORTRAN Call:

```
DIMENSION IDCB (144 + n), IBUF(n)
IL    = n   Length of IBUF (optional)
NUM = x   Record number to be written (optional)
CALL WRITF (IDCB, IERR, IBUR, IL, NUM)
Error code is returned in IERR
```

COMMENTS

- The IL (write request length) parameter takes on different meanings depending on the file type. Refer to Table 2-3.

- NUM is the random access record number and is used only when the file type is 1 or 2. Record number 1 is the first record in a file. If NUM = 0 or is absent, then the transfer starts at the current record number. If NUM is positive, then the transfer starts at the record number indicated by NUM. If NUM is negative, then the transfer starts at the current record plus NUM. Therefore:

File Type	Meaning
0	Not used.
1	NUM defines the first record to be written, and IL defines the number of records through the length in words.
2	NUM defines the record to be transferred.
3 and above	Not used.

Examples (refer to Figure 2-3):

1. If NUM = 0, transfer starts at record number 5.
2. If NUM = 6, transfer starts at record number 6 (same as NUM).
3. If NUM = −3, transfer starts at record number 2 (5 + (−3) = 2).

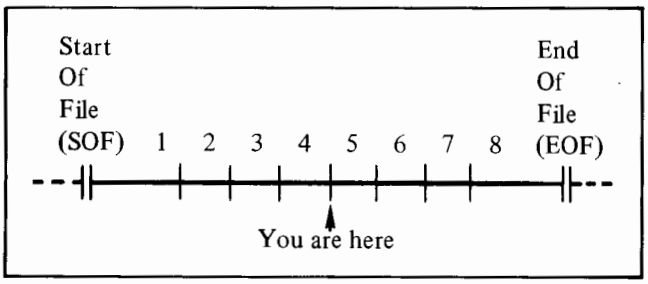


Figure 2-3. Record Positioning Example Using NUM in WRITF

SEQUENCE OF OPERATIONS

1. Set write flag and fetch parameters.
2. Check for enough parameters.
3. If DCB is not open, reject call.
4. Check security.
5. If update, set reading flag.
6. If type 1, force record length to 128.
7. If type 1 or 2 and EOF write, then exit, else do random access positioning (implies EOF if initial position not in file). If new position not in currently resident block, then write the block if it was written on.
8. If type 2 or above, and reading flag set and DCB buffer is empty, then read to DCB.
9. If type less than 3, skip to 19.
10. If current position is at EOF, then clear reading flag.
11. If EOF write: (a) then set EOF in buffer, (b) set written on flag, (c) step record count, and (d) exit.
12. If reading, then this is an update write; check lengths (must match).
13. Write the record length.
14. Write the record.
15. If skip count set, skip the rest of the record.
16. If type 2, then to 11c.
17. Write the length word.
18. If reading flag not set, go to 11a; else 11c.
19. If type 2, go to 14.
20. If type 0, go to 31.
21. If type 2, round up length to even 128 words and save as increment in record count.
22. Check if request within file.
23. If write, use round-up length.
24. If track switch, compute maximum words this access.
25. Write the record.
26. If type 1, check for disc errors.
27. Update for rest of record.
28. If more to transfer, go to 24.
29. Update record count.
30. Return.
31. Test write legality bits.
32. If EOF write, make control request and return.
33. Go to 25.

Table 2-3. Write Request File Length (IL) vs. File Type

Request Length / File Type	IL> 0	IL = 0	IL = –1	IL < –1 (Not recommended)	IL not supplied
0	The routine will write exactly IL words.	A zero length record is written.	An end-of-file is written.	The length (IL) is interpreted as a character count.	A zero length record is written.
1	Data is written in 128 word blocks. For example, IL is rounded up to 128 (1 block) if IL is between 1 and 127. IL is rounded up to 256 (2 blocks) if IL = 129 to 255.	No action.	No action.	No action.	128 is used.
2	IL is ignored and the file defined record length is used.	IL is ignored and the file defined record length is used.	No action.	No action.	The file defined record length is used.
> 2	The routine will write exactly IL words.	A zero length record is written.	An end-of-file is written.	Undefined.	A zero length record is written.

EXAMPLE PROGRAM

The following listings show a sample program (DEMO1) and subroutine (IERR). The program is designed to sort a file and place the results in another file using most of the calls described in this section. Subroutine IERR is called by the program to check if any errors were detected.

Operation is simple and straightforward. Most of the instructions are "commented" on the first page of the program. Proceed as follows:

a. Generate the code on tape or cards, or enter the code directly into a file with the Store Command.

```
:ST, 1, DEMO1S:::4:24
.
.
.
```

Use control D for end-of-file.

b. Submit a file to be sorted and call it USERF*x* where *x* is a number from 0 - 9.

```
ST,nam1, USERFx
```

c. Create a file that is the target file for the sorted data, and is called USERF*y* where *y* is a number from 0 - 9. One suggestion is to create a type 0 file referencing the line printer (list device = LU6).

```
:CR,USERFy,6,WR
```

d. Move the program into the RTE system and compile it with the FORTRAN Compiler. Once it's compiled, save the binary code in a file for future use.

e. Put the program into the system with the RTE loader.

f. Turn the program on with the required parameters.

*ON,DEMO1,*x*,*y*,*z*

where

USERF*x* is the file to be sorted (only *x* is required).

USERF*y* is the target file (only *y* is required).

z is the key word to be sorted on. For example, if the file to be sorted consisted of a group of names, *z* could be 1 indicating sort on the first two letters of the name.

```
DEMO1S T=00004 IS ON CR00100 USING 00024 BLKS R=0000

0001   FTN,L
0002         PROGRAM DEMO1(3,90)
0003   C
0004   C           THE PURPOSE OF THIS PROGRAM IS TO SORT A FILE CALLED
0005   C           USERFX AND PLACE THE RESULTING DATA IN FILE USERFY.
0006   C
0007   C           THE SORT WILL BE KEYED ON WORD POSITION Z.
0008   C
0009   C           X,Y AND Z ARE THE THREE TURN ON PARAMETERS.
0010   C
0011   C           A THIRD FILE (USERFA) IS CREATED BY THIS PROGRAM AND
0012   C           THEN PURGED AT TERMINATION.  THIS FILE WILL BE
0013   C           TYPE 2 WITH 4 WORD RECORDS.
0014   C
0015   C           METHOD OF APPROACH:
0016   C
0017   C           1. USERFX WILL BE READ AND THE ADDRESS OF EACH
0018   C              OF ITS RECORDS ALONG WITH THE KEY WORD (WORD Z)
0019   C              WILL BE SAVED IN USERFA.
0020   C
0021   C           2. USERFA WILL BE SORTED ON THE KEY WORD USING A
0022   C              "BUBBLE SORT".
0023   C
0024   C           3. THE ADDRESSES SAVED IN USERFA WILL THEN BE READ
0025   C              SEQUENTIALLY AND USED TO ADDRESS THE MATCHING
0026   C              RECORD IN USERFX.  THIS RECORD WILL THEN
0027   C              BE READ AND WRITTEN ON USERFY.
0028   C
0029   C
0030         DIMENSION IUSRX(144),IUSRY(144),IUSRA(144),IBUF(128),NUSR(4)
0031         DIMENSION IADRS(4)
0032   C
0033   C           SET UP THE FILE NAME ARRAYS
0034   C
0035         DATA NUSR/2HUS,2HER,0/,IR/43060B/,IUA/2HFA/
0036   C
0037   C           GET THE PRAMETERS AND SET UP DEFAULTS AS FOLLOWS:
0038   C
0039   C           USERFX DEFAULT IS USERF0
0040   C           USERFY DEFAULT IS USERF1
0041   C           Z      DEFAULT IS 2
0042   C
0043         CALL RMPAR(IUSRX)
0044         IUX=IR+IUSRX(1)
0045         IUY=IR+IUSRX(2)
0046         IWORD=IUSRX(3)
0047         IF(IUY-IR)20,10,20
0048    10   IUY=IUY+1
0049    20   IF(IWORD)40,30,40
0050    30   IWORD=2
0051   C
0052   C           OPEN THE USER FILES AND CREAT THE TEMP FILE.
0053   C
0054    40   NUSR(3)=IUX
0055    50   CALL OPEN(IUSRX,IER,NUSR,1)
0056   C
0057   C           NON-EXCLUSIVE OPEN FOR X SO OTHERS CAN READ IT TOO.
0058   C
```

```
0059          IF(IERR(IER,IUX))50,60
0060    C
0061    C           NOW AN EXCLUSIVE OPEN OF Y
0062    C
0063     60   NUSR(3)=IUY
0064     70   CALL OPEN(IUSRY,IER,NUSR)
0065    C
0066    C           CHECK FOR ERRORS
0067    C
0068          IF(IERR(IER,IUY))70,75
0069    C
0070    C           SET UP SIZE ARRAY FOR CREAT 100 BLOCKS,4 WORD RECORDS
0071    C
0072     75   IBUF(1)=100
0073          IBUF(2)=4
0074    C
0075    C           CREAT USERFA
0076    C
0077         NUSR(3)=IUA
0078     80   CALL CREAT(IUSRA,IER,NUSR,IBUF,2)
0079          IF(IERR(IER,IUA))80,90
0080    C
0081    C           ALL FILES ARE NOW OPEN
0082    C
0083    C           PHASE ONE BEGIN.
0084    C
0085    C           SET NUMBER OF RECORDS IN FILE TO 0
0086    C
0087     90   NOREC=0
0088    C
0089    C           FOURCE SHORT RECORDS TO TOP (I.E. DEFAULT IS A ZERO)
0090    C
0091     100  IBUF(IWORD)=0
0092    C
0093    C           GET CURRENT POSITION
0094    C
0095          CALL LOCF(IUSRX,IER,IADRS(2),IADRS(3),IADRS(4))
0096          CALL IERR(IER,IUX)
0097    C
0098    C           READ THE RECORD AND SET THE KEY
0099    C
0100          CALL READF(IUSRX,IER,IBUF,IWORD,L)
0101          CALL IERR(IER,IUX)
0102          IADRS(1)=IBUF(IWORD)
0103    C
0104    C           IF EOF THEN GO TO PHASE TWO.
0105    C
0106          IF(L)200,120
0107    C
0108    C           WRITE USERFA ENTRY
0109    C
0110     120  CALL WRITF(IUSRA,IER,IADRS)
0111          CALL IERR(IER,IUA)
0112    C
0113    C           STEP THE RECORD COUNT AND GO READ THE NEXT RECORD
0114    C
0115          NOREC=NOREC+1
0116          GO TO 100
0117    C
0118    C
```

```
0119  C           BEGIN PHASE TWO - BUBBLE SORT OF USERFA ON
0120  C           THE FIRST WORD (I.E. THE KEY WORD)
0121  C
0122  C           FIRST RE OPEN USERFA AND SET FOR UPDATE.
0123  C
0124   200   CALL OPEN(IUSRA,IER,NUSR,2)
0125         CALL IERR(IER,IUA)
0126  C
0127  C           SORT INITIALIZE.  IBUF WILL CONTAIN
0128  C           THE TWO RECORDS TO BE TESTED.
0129  C           ICUR WILL POINT AT THE CURRENT WINNER AND
0130  C           ITEST WILL POINT AT THE TEST RECORD.
0131  C
0132         ICUR=1
0133         ITEST=5
0134  C
0135  C           ISWCO WILL BE SET NON ZERO WHENEVER A RECORD IS SWAPPED.
0136  C
0137  C           START THE SORT
0138  C
0139         DO 295 K=2,NOREC
0140  C
0141  C           LAST WILL CONTAIN THE NUMBER OF THE LAST RECORD FOR
0142  C           THE CURRENT PASS.
0143  C
0144         LAST=NOREC-K+2
0145         ISWCO=0
0146  C
0147  C          READ THE CURRENT RECORD FOR THIS PASS
0148  C
0149         CALL READF(IUSRA,IER,IBUF(ICUR),4,L,K-1)
0150         CALL IERR(IER,IUA)
0151  C
0152  C           START SINK SECTION
0153  C
0154   205   DO 250 I=K,LAST
0155  C
0156  C           READ THE CURRENT TEST RECORD
0157  C
0158         CALL READF(IUSRA,IER,IBUF(ITEST),4,L,I)
0159         CALL IERR(IER,IUA)
0160  C
0161  C           TEST THE FIRST WORD AGAINST CURRENT WINNER.
0162  C
0163         IF(IBUF(ITEST)-IBUF(ICUR))225,210,210
0164  C
0165  C           ORDER O-K SET NEW WINNER.
0166  C
0167   210   J=ITEST
0168         ITEST=ICUR
0169         ICUR=J
0170         GO TO 250
0171  C
0172  C           OUT OF ORDER - SWAP THE RECORDS
0173  C           TEST TO CURRENT RECORD LESS ONE , CURRENT WINNER TO CURRENT
0174  C           RECORD POSITION
0175  C
0176   225   CALL WRITF(IUSRA,IER,IBUF(ITEST),4,I-1)
0177         CALL IERR(IER,IUA)
0178  C
```

```
0179  C           NOW THE CURRENT.  NOTE WE CAN USE A SEQUENTIAL WRITE HERE
0180  C
0181         CALL WRITF(IUSRA,IER,IBUF(ICUR))
0182         CALL IERR(IER,IUA)
0183  C
0184  C           SET THE SWAPPED FLAG
0185  C
0186         ISWCO=-1
0187   250   CONTINUE
0188  C
0189  C           END OF SINK SECTION
0190  C
0191  C           IS FILE ORDERED??
0192  C
0193         IF(ISWCO)255,300
0194  C
0195  C           NO.  NOW DO THE FLOAT SECTION
0196  C
0197   255   ISWCO=0
0198         DO 290 I=K,LAST
0199  C
0200  C           FLOAT SECTION GOES UP THE LIST SO IREC MUST BE COMPUTED
0201  C
0202         IREC=LAST-I+K-1
0203  C
0204  C           IREC IS THE CURRENT RECORD ADDRESS
0205  C           READ A RECORD
0206  C
0207         CALL READF(IUSRA,IER,IBUF(ITEST),4,L,IREC)
0208         CALL IERR(IER,IUA)
0209  C
0210  C           ITEST THE RECORD AGAINST THE CURRENT WINNER
0211  C
0212         IF(IBUF(ITEST)-IBUF(ICUR))260,260,280
0213  C
0214  C           ORDERED SO SET NEW WINNER.
0215  C
0216   260   J=ITEST
0217         ITEST=ICUR
0218         ICUR=J
0219         GO TO 290
0220  C
0221  C           NOT ORDERED SO SWAP THE RECORDS
0222  C
0223   280   CALL WRITF(IUSRA,IER,IBUF(ICUR),4,IREC)
0224         CALL IERR(IER,IUA)
0225  C
0226  C           HERE AGAIN BY CHOSING THE ORDER WE CAN USE A SEQUENTIAL
0227  C           WRITE
0228  C
0229         CALL WRITF(IUSRA,IER,IBUF(ITEST))
0230         CALL IERR(IER,IUA)
0231  C
0232  C           SET THE SWAPPED FLAG.
0233  C
0234         ISWCO=-1
0235   290   CONTINUE
0236  C
0237  C           END OF FLOAT SECTION.  WERE ANY SWAPPS MADE??
0238  C
```

```
0239         IF(ISWCO)295,300
0240   C
0241   C           YES SO CONTINUE THE SORT
0242   C
0243    295  CONTINUE
0244   C
0245   C           END OF SORT
0246   C
0247   C           TRANSFER THE FILE IN SORTED ORDER
0248   C
0249    300  DO 390 I=1,NOREC
0250   C
0251   C       READ THE ADDRESS
0252   C
0253         CALL READF(IUSRA,IER,IADRS,4,L,I)
0254         CALL IERR(IER,IUA)
0255   C
0256   C           SET FILE USERFX TO THE ADDRESS.
0257   C
0258         CALL APOSN(IUSRX,IER,IADRS(2),IADRS(3),IADRS(4))
0259         CALL IERR(IER,IUX)
0260   C
0261   C           READ THE RECORD
0262   C
0263         CALL READF(IUSRX,IER,IBUF,128,L)
0264         CALL IERR(IER,IUX)
0265   C
0266   C          WRITE THE RECORD ON USERFY
0267   C
0268         CALL WRITF(IUSRY,IER,IBUF,L)
0269         CALL IERR(IER,IUY)
0270    390  CONTINUE
0271   C
0272   C           ALL OF THE FILE IS TRANSFERED SO WRITE AN EOF AND
0273   C           CLOSE ALL THE FILES.
0274   C
0275         CALL WRITF(IUSRY,IER,IBUF,-1)
0276         CALL IERR(IER,IUY)
0277   C
0278   C           PURGE USERA USING THE TRUNCATE OPTION
0279   C
0280         CALL CLOSE(IUSRA,IER,100)
0281         CALL IERR(IER,IUA)
0282   C
0283   C           CLOSE IUSERX AND IUSERA
0284   C
0285         CALL CLOSE (IUSRX,IER)
0286         CALL IERR(IER,IUX)
0287         CALL CLOSE(IUSRY,IER)
0288         CALL IERR(IER,IUY)
0289   C
0290   C           END OF PROGRAM   TERMINATE.
0291   C
0292         END
0293         FUNCTION IERR(ICOD,ID)
0294          DIMENSION MESS(13)
0295   C
0296   C           THIS FUNCTION CHECKS FOR ERRORS AND IF ICOD IS <0
0297   C           THEN PRINTS IT ON THE SYSTEM TTY.  IT THEN PAUSES.
0298   C
```

```
0299  C           IF THERE IS NO ERROR IT JUST RETURNS.
0300  C
0301  C           IN ANY CASE THE ERROR CODE ICOD IS RETURNED AS THE VALUE
0302  C
0303  C
0304  C           SET UP THE MESSAGE
0305  C
0306        DATA MESS/2H  ,2HER,2HRO,2HR ,2H  ,2H O,2HN ,2HFI,2HLE,
0307       C           2H  ,2HUS,2HER,2H  /
0308  C
0309  C           SET THE RETURN CODE
0310  C
0311        IERR=ICOD
0312  C
0313  C           WAS THERE AN ERROR??
0314        IF(ICOD)10,99
0315  C
0316  C           YES SET UP THE MESSAGE.
0317  C
0318   10   I=-ICOD/10
0319        J=-ICOD-I*10
0320        MESS(5)=I*400B+2H00+J
0321        MESS(13)=ID
0322        CALL EXEC(2,1,MESS,13)
0323        PAUSE
0324   99   END
0325        END$
```

SECTION III
FMGR COMMANDS

INTRODUCTION

The computer system user controls the File Management Package (FMP) by FMGR commands entered through the system input device. These commands cause the FMP to perform the functions shown in Table 3-1.

COMMAND STRUCTURE

The operator gains the attention of the FMP by scheduling the software module FMGR with the ON, FMGR operator request. When FMGR runs, a colon (:) prompter is returned on the initial keyboard input device. If the input device is not a keyboard, the first command is read from the device that is the input. Each command is parsed, or resolved, by a central routine that accepts certain conventions. The command syntax (described in Table 3-2), together with the following conventions, must be followed exactly.

CONVENTIONS

- A leading plus sign (+) in a numeric parameter is ignored. The number is assumed to be positive unless preceded by a minus sign (−).
- A numeric parameter immediately followed by the letter 'B' indicates the parameter is octal.
- A numeric parameter immediately followed by a letter 'G', 'P', or 'S' is assumed to be a global reference and is replaced by the current value of that global. (Global parameters are described later in this section under "Globals".)
- Each command from a device other than a TTY must be preceded by a colon (:). On the TTY the system generates this.
- The delimiter separating the command from its parameters may be either a colon (:) or a comma (,). Thus, a null command could take either the form of a double colon (::) or a comma following the prompter colon (:,). Either form of the null command is interpreted as a TR command.
- Each parameter is tested on the assumption that it is numeric. If it fails to convert, it is then assumed to be ASCII. If the parameter is ASCII, the first six characters are saved. If less than six characters are entered, the parameter is padded to six characters with trailing blanks. For example, if 6 followed by a blank is entered as a parameter, the Parse routine would assume it to be numeric. If 6 blank x (a non-numeric character) is entered, the Parse routine would assume it to be ASCII.
- Two commas or colons in a row mean a parameter assumes its default value.
- Leading blanks and blanks on either side of a comma or colon are ignored.
- An EOF encountered on the command input file causes a TR with no parameters.
- All parameters are initially filled with zeros and flagged as not supplied, so FMGR can distinguish null (space or nothing) from zeros.
- All commands must meet the following syntax rules:

  a. After the first delimiter (separating the command from its parameters), the parameters are separated from each other by a comma (,).
  b. Subparameters (described later in this section under "*namr*") are separated from the parameter and from each other by a colon (:).
  c. Subparameters are allowed on the first two parameters only.
  d. No more than five subparameters are allowed.
  e. Subparameters 3, 4, and 5 (if supplied) must be numeric.
  f. The number of characters in a parameter must be less than:

  $$64 - 4 * \textit{parameter number}$$

  g. The maximum number of parameters that may be used in one command is 14.
- Comments may be entered in any command following the last parameter. A comment must appear to be a parameter and follow all syntax rules for parameters.

Table 3-1. FMGR Commands

Command Format	Description	Page
AB	Aborts a job and does normal end-of-job cleanup.	3-10
AN	Writes a message on the Job list file.	3-10
CA	Calculates and sets a value into a global parameter.	3-10
CL	Lists the cartridge directory.	3-11
CO	Copies all files on one cartridge to another cartridge.	3-11
CN	Issues specified control request to the indicated device.	3-12
CR	Creates a file--does not store data.	3-12
CS	Modifies or changes spool options chosen at setup.	3-14
DC	Requests FMP to logically disconnect the cartridge from the system.	3-15
DL	Lists the file directory.	3-16
DP	Displays current values of global parameters on the system console.	3-18
DU	Dumps a file--does not create a file.	3-19
EO	Ends a job.	3-21
IF	Does conditional branching in the command stream.	3-21

Table 3-1. (continued)

Command Format	Description	Page
IN	Initializes a cartridge.	3-22
JO	Job control.	3-24
LG	Allocates or releases disc tracks for load-and-go.	3-24
LI	Lists file contents.	3-25
LL	Changes assignment of output list device.	3-26
LO	Changes assignment of log device.	3-26
LS	Sets up base page pointer to logical source track	3-26
LU	Spool setup or logical unit switch.	3-27
MC	Notifies FMP a cartridge has been mounted.	3-28
MR	Moves a relocatable binary file to the load-and-go tracks.	3-29
MS	Moves a source file to the logical source tracks.	3-29
OF	Removes program; same as RTE-II system command OF, 8.	3-30
PK	Packs one or all mounted cartridges.	3-30
PA	Transfers control to specified device and prints text on log device.	3-31
PU	Purges a file.	3-32
RN	Renames a file.	3-32
RP	Restores a saved program.	3-32

Table 3-1. (continued)

Command Format	Description	Page
RT	Releases all disc tracks assigned to a program.	3-33
RU	Seeks out and executes a specified program or transfers file.	3-34
SP	Saves an RTE-II system program—creates a type 6 file.	3-34
SA	Creates a file and saves the logical source or load-and-go tracks.	3-35
SE	Sets global parameters 1G through 9G.	3-35
ST	Stores a file—creates a new file.	3-36

Table 3-1. (continued)

Command Format	Description	Page
SV	Changes error print-out severity code.	3-38
TE	Sends operator message to TTY.	3-39
TL	Sets a run-time limit for RUn executed programs.	3-39
TR —or— null	Transfers control to another file or logical unit.	3-39
??	Expands error message.	3-40
EX	Terminates FMGR.	3-41

BATCH LOGICAL UNIT SWITCH TABLE

When batch jobs are run with spooling, program references to logical units are transformed internally to system assigned logical units corresponding to spool files. This is done because the spool system reads from and outputs to disc files. For instance, a program may specify logical unit 5 as the input device, but the Spool Monitor assigns a different logical unit internally for the spool input device. Similarly, program output to logical unit 6 is transformed by the system to another logical unit for the spool output device.

The logical unit numbers associated with system-managed spool files are established at system generation and must have entries in the equipment table (EQT). The system sets up a Batch Logical Unit Switch Table that keeps track of pairs of logical units, namely, the logical unit specified in a user program and its corresponding system-assigned logical unit.

SPOOL POOL

The system spool files associated with system-assigned logical units in the Batch Logical Unit Switch Table constitute a "spool pool". These spool pool files are created by the GASP program as type three files; up to 80 spool pool files may be created, (refer to Section XI, GASP Initialization, for the dialogue used in creating these files).

Table 3-2. Conventions in Operator Command Syntax

Item	Meaning
UPPER CASE ITALICS	These words are literals and must be specified as shown.
lower case italics	These are symbolic representations indicating what type of information is to be supplied. When used in text, the italics distinguishes them from other textual words.
REad	A combination of the above shows that the RE is a literal and is specified as shown. The remainder of the characters are for information only and may or may not be entered.
[, *item*]	Items with brackets are optional. However, if *item* is not supplied, its position must be accounted for with a comma. This causes *item* to default.
[,*item 1* ,*item 2* ,*item 3*]	This indicates that exactly one *item* (or, optionally, none) may be selected from among those listed to be specified. If *item* is not supplied, its position must be accounted for with a comma. This causes *item* to default.
... .	This notation means "and so on."
item 1 *item 2* *item 3*	This indicates that there is a choice of entries for the parameter, but one parameter must be specified.
DL [, *cl* [,*msc*]]	The placement of the brackets indicates that the *cl* and *msc* parameters are both optional. However, if *cl* is not supplied, its position must be accounted for with a comma. This causes *cl* to default. Example: :DL, ,JB
namr	This is considered to be one parameter with up to five subparameters separated by colons(:).

namr = file name [*:security code* [*:label*[*:file type*[*:file size*[*:record size*]]]]]

– or –

namr = logical unit number

NOTE

File size is specified in blocks, and record size is required only for type 2 files on :CR or CREAT.

GLOBALS

Global parameters are variables that may be set, examined, and/or manipulated by the job stream. They serve to alter procedure files by supplying data required to accomplish a given function. There are three types of globals identified as 'G', 'S', and 'P' globals.

G – There are 11 G-type globals numbered 0 through 10. Globals 1G through 9G may be altered by the TR, CA, and SE Commands. Global 0G is set up to be the turn-on command unit or file and is reset when a JOB statement is processed to be the Job Input file *namr*. Global 10G is assumed to be ASCII; it is set by the RUn Command whenever a program passes back data. Notably, the LOADR passes back the program name it just loaded.

S – There are two S-type globals which are set up by the Spool Monitor as follows:

1S – The file name of the last created spool file.
0S – The actual logical unit (LU) of the file.

NOTE

0S is always numeric if defined and 1S is always ASCII if defined. 0S and 1S are both null if spooling is not being used.

G-type and S-type globals are all 4-word globals. The format is as follows:

Word 0 (type)	0 (null)	1 (numeric)	3 (ASCII)
Word 1	0	Value	Characters 1, 2
Word 2	0	0	Characters 3, 4
Word 3	0	0	Characters 5, 6

Word 0 determines the type.

P – There are five P-type globals. They represent the values returned by a RUN program as integers. They are accessed as 1P through 5P. P-type globals are always assumed to be numeric.

All globals are kept in an array in memory. Since the system checks a global access only for its actually being within the array, values other than those shown above may be used to reference this area. The following map details how this referencing might be done:

S	G	P	
Ø	-2	-48	Type
		-47	1
		-46	2
		-45	3
1	-1	-44	Type
		-43	1
		-42	2
		-41	3
2	Ø	-4Ø	Type
		-39	1
		-38	2
		-37	3
3	1	-36	Type
		-35	1
		-34	2
		-33	3
4	2	-32	Type
		-31	1
		-3Ø	2
		-29	3
5	3	-28	Type
		-27	1
		-26	2
		-25	3
6	4	-24	Type
		-23	1
		-22	2
		-21	3
7	5	-2Ø	Type
		-19	1
		-18	2
		-17	3
8	6	-16	Type
		-15	1
		-14	2
		-13	3
9	7	-12	Type
		-11	1
		-1Ø	2
		- 9	3
1Ø	8	- 8	Type
		- 7	1
		- 6	2
		- 5	3
11	9	- 4	Type
		- 3	1
		- 2	2
		- 1	3
12	1Ø	Ø	Type
		1	1
		2	2
		3	3
		4	4
		5	5

The standard values are shown within dark lines.

As the map shows, certain S-type globals are equivalent to certain G-type globals. For example, 0S is identical to −2G and 5G is identical to 7S. Also, note that the P-type globals may be used to access all of the globals as integers. In particular, they may be used to break apart a file name in a global, so that it may be passed to a program in a RUn Command.

As an example, 3G could be passed to program XYZ as follows:

:RU, XYZ, −27P, −26P, −25P

BREAKING FMGR

Most FMGR commands recognize an RTE-II BReak request. For those that don't, the BReak request is recognized between commands. If a break is recognized while a command is creating a file, the file is purged. The PacK Command will break only between discs. A break causes the message

FMGR 000

to appear on the system console, and it causes a transfer to the log device unless a job is being processed. If a break is recognized during a job, the job is aborted.

A break system command may be entered as either

BR, FMGR
—or—
AB

However, the AB Command will abort any program FMGR is running. (Refer to the RTE-II Manual for more information on the AB Command.)

NAMR

namr is the symbolic representation of the general file reference parameter for most of the FMGR commands. *namr* is considered to be one parameter, a file name, with up to five subparameters separated by colons (:).

namr = file name [:security code [:label [:file type [:file size [:record size]]]]]

—or—

namr = logical unit number

namr can be a logical unit number of a TYY, punch, photo reader, card reader, magnetic tape, or any such device except a disc.

FILE NAME

A legal file name is six ASCII characters in length. The first character must not be a blank or a number. All characters must come from the set "blank" through "_"* (see Appendix E) exclusive of special characters "+" (plus), "–" (minus), "," (comma), and ":" (colon). Imbedded blanks are not allowed. Only the first and second parameter of a FMGR command can be a *namr* or have subparameters, and only the first and second subparameters may be non-numeric.

*On some devices, the underscore (_) prints as a left-arrow (←).

NOTE

All of the following described subparameters are initially filled with zeros prior to each command; thus, omitted parameters are taken as zeros. (Two colons in a row mean a subparameter is omitted or is defaulted to zero.)

SECURITY CODE

The security code can be ASCII or numeric characters, or a combination of both. The largest allowable number is 32767. When a code is not specified it defaults to zero. The security code parameter is defined as follows:

security code = 0 – file is not protected and can be read or changed by any user.

security code > 0 – file is write protected but can still be read without security code. If 2's complement of positive code is given, then the file will be opened for both read and write.

security code < 0 – file is protected and can not be opened without the correct security code.

LABEL

The label parameter has three meanings as shown below:

label > 0 – indicates a cartridge reference (CR) number (positive integer) that is a numeric identifier assigned to all cartridges in the system. A listing of the cartridge directory (CL Operator Command) would reveal a number under the heading CR. This is the cartridge label and is directly associated with a logical unit number.

label = 0 – indicates the first available disc which satisfies the request to be used. Note that the discs are searched in the order they appear in the disc directory.

label < 0 – indicates the logical unit number of the disc to be used (e.g., –14 indicates logical unit number 14).

NOTE

The following subparameters, FILE TYPE, FILE SIZE, and RECORD SIZE, are used only if the file being described is to be created by the command.

FILE TYPE

File types are defined as follows:

0 – non-disc file

1 – 128-word record length, random access

2 – user selected record length, random access

3 – (and greater) random record length, sequential access

4 – source program

5 – relocatable program

6 – RTE load module

7 – absolute program

>7 – user defined.

FILE SIZE

File size is always specified in blocks, where two sectors equal one block (128 words). If a negative file size is specified, all of the available file space is allocated to that file. When the file is finished being loaded, the unused area is returned to the File Manager (i.e., the file is truncated to the area used). This is useful when it is desired for a file to not have any extents.

RECORD SIZE

Record size is required in the command only if the file is type 2. When a file is created (except type 0 files), the record size parameter is put in the directory and may subsequently be recovered by the user interface (see the LOCF call) or by the LI Command.

Examples of how *namr* may be specified are shown below.

10	File is on $LU10_{10}$, a non-disc file.
20B	File is on $LU20_8$ ($LU16_{10}$), a non-disc file. The B suffix indicates octal.
$XYZ:SC	File name is "$XYZ" with ASCII security code "SC."
ABS:-10:-3:2:40:6	File to be created will be named "ABS," have security code -10, be on LU3, type 2, 40 blocks long, and have 6-word records.
A23456 : : : 20	File to be created will be named "A23456" and be type 20; security and cartridge reference are not supplied.

LOCKING

When a cartridge is locked, only the locking program may access files or make directory changes on that cartridge. To other callers, it appears as if the cartridge is not mounted. A lock can only be obtained if all files on that cartridge are closed. A failure to obtain a lock is indicated by error -8.

A cartridge is locked whenever FMGR:

- is packing it,
- is initializing it,
- is removing it,
- is creating a type 0 file on it,
- is purging a type 0 file on it,
- mounts it and finds it has not been initialized.

If a user program attempts an access of a locked cartridge, error code -13 will be transmitted. For example, if FMGR is packing a cartridge in the background, and another program in the foreground requests that a file on that cartridge be opened; the -13 cartridge locked error will result. This is because the cartridge is locked by FMGR when performing the pack.

FMGR TURN-ON SEQUENCE

The File Manager operator interface routine, FMGR, can be turned on using the RTE System command ON, FMGR; or by internal scheduling with a user's program. However, before a brand new moving-head cartridge can be used on any RTE System it must be properly formatted. Formatting consists of issuing format data commands on the whole cartridge and flagging any defective tracks as bad tracks. Formatting may be done by the controller diagnostic routine or by the moving-head RTE Generator.

If the cartridge has not previously been formatted, refer to Appendix M for the cartridge formatting procedure.

ON,FMGR

Purpose:

To schedule the background program FMGR that allows communication between FMP and the operator.

Format:

*ON,FMGR[,*input*[,*log*[,*list*[,*severity code*]]]]
or
*ON,FMGR,*filename*[,*list*[,*severity code*]]

Where:

input is the logical unit number of the input device. Default is LU1 (system teleprinter).

log is the logical unit number of a two-way device for logging an error detected on a command from the input device. If an entry is not given then *log* assumes the logical unit number given for *input*, only if *input* is a TTY; otherwise, *log* assumes logical unit number 1.

filename is the file name in the form:

FI,LE,NA

may only be used if

- first two characters are not "NO".
- filename contains no digit pairs that Parse routine treats as numeric parameter.

list is the logical unit number of the list device. Default is LU6 (standard list device).

severity code is the error message severity code. Default is 0. The options are:

0 – All commands are echoed on the log device. Any error causes an appropraite error number to be printed.

1 – Inhibit command echo on log device.

2 – Inhibit error messages to log device, unless the error is severe enough to cause control to transfer to the log device for a command input.

FMGR SCHEDULE

Purpose:

To internally schedule the File Manager operator interface routine FMGR.

```
                EXT     EXEC
                 .
                 .
                 .
SCHED           JSB     EXEC        Transfer control to RTE
                DEF     RTN         Return point
                DEF     ICODE       Request code
                DEF     FMGR        Name of program to schedule
                DEF     P1   )
                 .           )
                 .           )      Up to five optional parameters
                 .           )
                DEF     P5   )
RTN                                 If not scheduled Loop
                 .
                 .
                 .
ICODE           DEC     23 or 24    23 = schedule with wait, 24 = no wait
FMGR            ASC     3, FMGR     Name of program to schedule
P1   )
     )                              Up to five optional parameters; see Comments
P5   )

DIMENSION FMGR(3)
ICODE=23 or                         schedule with wait
ICODE=24                            schedule with nowait
FMGR(1)=2HFM
FMGR(2)=2HGR                        file name
FMGR(3)=2H

CALL EXEC(ICODE,FMGR,P1,P2,. . . ,P5)

P1 through P5                       optional parameters; see Comments
```

COMMENTS

- The optional parameters are defined in the ON,FMGR Command on the previous page. If desired to turn on FMGR to read its commands from a file name, P1 through P5 may be defined as:

```
P1   ASC 1, FN   )
P2   ASC 1, AM   )   File name
P3   ASC 1, E    )
P4   DEC x           severity code
P5   DEC y           List logical unit number
```

FNAME can be any ASCII file accessible with a cartridge reference number of 0, and a positive or zero security code to allow reading (i.e., + or 0). The log unit in the above call is defaulted to LU1.

- This call schedules FMGR when it is dormant. The calling program is suspended until FMGR is scheduled successfully.

WAITING AND NO WAITING

- A schedule with waiting (ICODE = 23) causes RTE to put the calling program in waiting status. The called program runs at its own priority, which may be greater than, less than, or equal to that of the calling program. Only when the called program terminates does RTE resume execution of the original program at the point immediately following the schedule call.
- A Schedule EXEC Call with no waiting (ICODE = 24) causes the specified program to be scheduled for execution according to its priority.

PARAMETERS

- If FMGR is scheduled with wait and an MS command is executed, the scheduling program may call RMPAR

to get the LS track and logical unit. These will be returned in parameter two in the same format as the base page LS word; i.e., the sign bit set indicates logical unit 3, else 2 and the track number is in bits 14 through 7.

- Schedule with wait example:

```
      DIMENSION I(5),NAME(3)
      DATA NAME/2HFM,2HGR,2H /
        .
        .
      CALL  EXEC(23,NAME,2HMY,2HFI,2HLE,1)
C     PUT LS TRACK IN I(2)
      CALL RMPAR(1)
        .
        .
```

ABort

Purpose:

To terminate a job immediately and cause end-of-job cleanup.

Format:

:AB

COMMENTS

This command causes normal EOJ cleanup to be done. If the job is not being read from a spool file, then data will be read and flushed from the appropriate LU until a :JOb card is found or until the hopper is empty or an EOF is encountered.

ANotate

Purpose:

To write a message on the JOb list file.

Format:

:ANotate, *message*

Where:

message is the message (verbatim) that is to go into the list file.

COMMENTS

- This command is similar to the TEllop Command, except that the message goes to the list file.
- When the message goes into the list file, the whole line (including the ":ANotate," prefix as well as the words in the *message* itself) is sent to the list file.
- The length of the *message* (following :AN,) must not exceed 59 characters.
- This command must meet the same syntax requirements common to all commands.

CAlculate

Purpose:

To calculate and set a value into a global parameter.

Format:

:CA,#, *operand*1, *operation*, *operand*2, *operation*, *operand*3 . . .

Where:

is the global number to be set to the result of the calculation.

operation is one of the following codes:

+	add	(1 + 2)
–	subtract	(1 – 2)
/	divide	(1/2)
*	multiply	(1*2)
O	or	(1 or 2)
X	exclusive or	(1 XOR 2)
A	and	(1 AND 2)

COMMENTS

- The type of the result depends on the types of the operands. The types are:

  0 = null
  1 = numeric
  3 = ASCII

  In all cases except divide (/) and multiply (*), the calculation is done independently on the three-word values of the operands. In the case of divide and multiply, the first word of operand 2 is used for all three words of operand 1.

- Evaluation proceeds from left to right until a null operation code is detected. Any other precedence must be effected by multiple statements.

For example:

:CA,6,HI Sets global 6G to ASCII value "HI".

:CA,1,1G,−,1 Decrements 1G

:CA,5,15 Sets global 5G to integer value 15.

Cartridge List

Purpose:

To list the active cartridge labels and their status.

Format:

:CL

COMMENTS

- The computer lists the status information on the list file. Note that the list file can be a device (e.g., line printer), or a file (refer to the LL command). The information is preceded by the following heading:

  LU LAST TRACK CR LOCK

  Where:

  LU = logical unit number of the cartridge.

  LAST TRACK = last track on that LU assigned to FMP.

  CR = cartridge reference number.

  LOCK = the locking program's name (or blank).

COpy files

Purpose:

To copy or transfer all files from one cartridge to another cartridge.

Format:

:CO,*label1*,*label2*

Where:

label1 is the cartridge label where the files are presently residing.

label2 is the new cartridge label to which the files are being transferred.

NOTE

Both *label1* and *label2* can be either an LU(−) or CR(+).

COMMENTS

- All files being transferred must have unique names (i.e., no two files having the same name can reside together on the same cartridge). If files with the same name are located on both *label1* and *label2* the file on *label1* is not copied onto *label2*, but is reported with a FMGR −002 (duplicate name) error. For example, if files A, B, C, and D on cartridge *label2* are to be copied to cartridge *label3*, and file C already exists on *label3*, the following information would be printed on the log device:

  ```
  :CO,2,3
  A
  B
  C
  FMGR −002
  D
  :
  ```

- Restrictions associated with the COpy routine are as follows:

  a. *label1* and *label2* are both mounted.
  b. *label1* is not *label2*.
  c. Type 3 or above files must contain an EOF. This implies that all records between the beginning of the file and the end-of-file must be valid.

NOTE

The COpy routine transfers files record per record. Records longer than 128 words are truncated.

CNtrol

Purpose:

To issue the specified control request to the indicated device.

Format:

:CNtrol [,*namr* [,*function* [,*subfunction*]]]

Where:

namr is the logical unit number or type 0 file name of the device. *namr* defaults to LU8 (the recommended magnetic tape LU).

function is one of the following:

Mnemonic	Resulting Function Code	
RWind	4	
EOfile	1	
TOform	11B	
FFile	13B	(file spacing)
BFile	14B	
FRecord	3	(record spacing)
BRecord	2	
LEader	10B	
numeric		

subfunction is an additional parameter required only with TOform.

Top of form	–	Line printer
2 spaces	–	TTY/CRT
Leader	–	Tape punch

or as defined when the file was created if it is a type 0 file.

COMMENTS

- The function code resulting from the mnemonic function correspond to the codes in the call FCONT (Section II).
- If the device type is a number greater than 16, *function* defaults to RWind. If the device type is a number less than or equal to 16, *function* defaults to FMGR EOF, where EOF is defined as follows:
- TOform requests require an additional parameter (*subfunction*) for correct implementation. The default (*subfunction* = –2) gives top-of-form on a line printer and 2 spaces on a TTY/CRT. Changing the *subfunction* with TOform will result in the functions described in the "Line Printer Formatting" appendix in the *Real-Time Executive II (RTE-II) Software System* Programming and Operating Manual.
- The numeric function allows the user to issue a control request not in the mnemonic list. These codes are listed in the FCONT description in Section II.

CReate file

Purpose:

To create a file name on a cartridge.

Format:

:CR,*namr*

Where:

namr is the file name and parameters.

COMMENTS

- Only a file name is created, no data is transferred.
- Parameters not included in *namr* default to 0. The file type must be specified and not be type 0, and the size must be greater than 0.
- If the file type is 3 or greater, an EOF mark is written at the beginning of the file.

CReate type 0 file

Purpose:

To create type 0 file with special directory entry for device control.

Format:

```
                 ,REad   [ ,BSpace   [ ,EOf      [ ,Binary  ] ] ]
:CR,namr, lu,    ,WRite    ,FSpace    ,LEader     ,AScii
                 ,BOth     ,BOth      ,PAge       ,numeric ]
                                      ,numeric
```

Where:

namr is the file name and security code parameter. The file type parameter is 0 which is default.

lu is the logical unit number of the device (not a disc) to be controlled.

The *REad, WRite,* and *BOth* parameters indicate the legal input/output.

REad	indicates an input request is legal (implies *FSpace*).
WRite	indicates an output request is legal (implies no *FSpace*).
BOth	indicates both input and output requests are legal.

The *BSpace, FSpace* and *BOth* parameters indicate legal spacing. Default is no spacing.

BSpace	indicates backspacing is legal.
FSpace	indicates forward space is legal.
BOth	indicates both backspace and forward spacing is legal.

The *EOf, LEader,* and *PAge* parameters specify the control subfunction for the end-of-file WRITE request. Default is *LEader* if the driver number is 02 (octal). If the driver number is greater than 16 (octal), default is *EOf;* otherwise the default is *PAge.*

EOf	specifies and end-of-file mark (magnetic tape).
LEader	specifies paper tape leader.
PAge	specifies page eject for the line printer, or two line feeds for a TTY.
numeric	user specified control subfunction. The number entered is interpreted as decimal (unless followed by a B) with only the last five bits used. (See FCONT in Section II for values.)

The following parameters specify the subfunction portion of the input/output request. Default is *AScii.*

BInary	specifies binary data transfer. (*BI* is used here. *BN* is used in Dump and Store.)
AScii	specifies ASCII data transfer.
numeric	user specified control subfunction. The number entered is interpreted as decimal (unless followed by a B) with only the last five bits used. (See OPEN in Section II for values.)

COMMENTS

- A type 0 file may exist only in the file directory for the system disc (LU2). When created, the type 0 file will be entered prior to all files as the first entry in the directory. This means that in order to assume this position the system disc must be locked when creating the type 0 file.

- A type 0 file has a special directory entry (see Appendix A), and is used to control devices other than discs which use standard peripheral calling sequences. Type 0 files may exist only in the system discs' file directory and may only be created and purged by the operator interface FMGR. Cooperating programs may use type 0 files as a means of controlling access to a device (see exclusive open in Section I). Type 0 files also afford a measure of device independence in that the standard file commands can be used to control the device.

Change Spool Options

Purpose:

To modify or change spool options set up by the LU command.

Format:

```
                RWind
:CSpool,lu,PUrge
                SAve
                PAss
                ENd
                BUffer
                NBuffer
                NPass [,lu#] [,priority]
```

Where:

lu	is the logical unit defined at setup.
RWind	resets the file to the first record.
PUrge	changes a *SAve* flag to purge.
SAve	changes a *PUrge* flag to SAve.
PAss	removes the *HOld* option.
ENd	writes a final EOF and terminates the spool. Spool file will be placed in an outspool queue, if this has not already been done.
BUffer	changes from no buffering to buffering.
NBuffer	changes from buffering to no buffering.
NPass	changes the logical unit and/or priority information.
lu#	is the new logical unit.
priority	is the new priority.

COMMENTS

- See the LU Command for a description of spool options.

- The default is *ENd*.

- If no spool file is involved, the LU map will be cleared by an ENd. In that case, other options have no effect.

Dismount Cartridge

Purpose.

To make a cartridge unavailable to the FMP, and remove its entry from the disc directory.

CAUTION

The DC command must be issued before a cartridge is removed.

Format:

:DC,*label*

Where:

label is the parameter relating to *namr*.

label>0 indicates CR.

label<0 indicates a logical unit number

label =0 is not allowed

COMMENTS

- The cartridge designated by *label* is first locked (refer to the heading LOCKING). The cartridge's directory entry is then removed form the cartridge directory, and the last FMP track reported on the log device. It is suggested that this last track number be written on the platter to facilitate the future restoration of the cartridge with the MC Command.

- The File Manager will not allow cartridges assigned to LU2 or LU3 to be physically removed with the DC command. This is because LU2 and LU3 are both located in the RTE System track assignment table, and must stay mounted in order to properly configure the table when the system is initially started (booted). If the DC Command is issued to LU2 or LU3, the cartridge is locked and removed from the disc directory list. The cartridge is then immediately remounted at the bottom of the list. With this feature, the DC Command can be used to change the order of the cartridge directory list. For example, the directory list shown below is the order of cartridges immediately after the MC Command was used to mount the disc at LU14.

LU	LAST TRACK	CR	LOCK
02	0201	00002	
03	0201	00003	
14	0201	00055	

To place the peripheral disc (LU14) at the top of the directory list, issue the following commands (for this example only):

```
:DC,2
:DC,3
```

The File Manager will change the directory list as follows:

LU	LAST TRACK	CR	LOCK
14	0201	00055	
02	0201	00002	
03	0201	00003	

APPLICATION

The restriction on physically removing the cartridge assigned to LU3 can be circumvented if the following requirement and procedure are adhered to.

REQUIREMENT

All cartridges to be mounted at LU3 must be initialized to use the same first track. It is recommended that track 0 be selected as the first track to avoid the possibility of the loader or system placing a program in the area.

PROCEDURE

Enter the following commands.

:DC,−3 (Insures all files are closed)

Remove the cartridge from the drive and insert the replacement.

:MC,−3 (Places new cartridge label in the disc directory)

The above procedure will work only if both cartridges have been initialized to use the same first track (recommended track 0). If the new cartridge has not been initialized, FMGR will lock it. An attempt to initialize the new cartridge at this point will result in FMGR error 59 because the directory tracks are already assigned to D.RTR. This is solved as follows:

*RT,D.RTR (RTE command to release D.RTR tracks)

*ON,FMGR (FMGR will reassign D.RTR tracks on LU2, then terminate)

*ON,FMGR
:MC,-lu
:IN,*master security code*, -etc. (Initialize LU3)

This completes the procedure.

APPLICATION

Operator commands that use *namr* can take advantage of the default characteristic of the *namr* subparameter *label*. For example, when creating a file with the CR Command, and *label* is allowed to default to 0, the file is placed on the disc at the top of the directory list. Refer to the heading NAMR for more information on *label*.

Directory List

Purpose:

To list the contents of file directory of one or all of the mounted cartridges.

Format:

:DL[,*label*[,*master security code*]]

Where:

label is the cartridge identification, positive for CR or negative for logical unit number.

master security code is the two-character FMP master security code designated at initialization time.

NOTE

If the master security code is 0, default in command will not obtain long list – a code (any code) must be supplied.

COMMENTS

- The *label* parameter indicates which cartridge file directory is to be examined. If *label* is not present, or it is a 0, all mounted cartridge directories will be listed. There are two list formats. Figure 3-1 shows the format when the master security code is not provided.
- Figure 3-2 shows the format when the master security code is provided. In addition to the information described in Figure 3-1, the file security code plus track and sector addresses are provided.

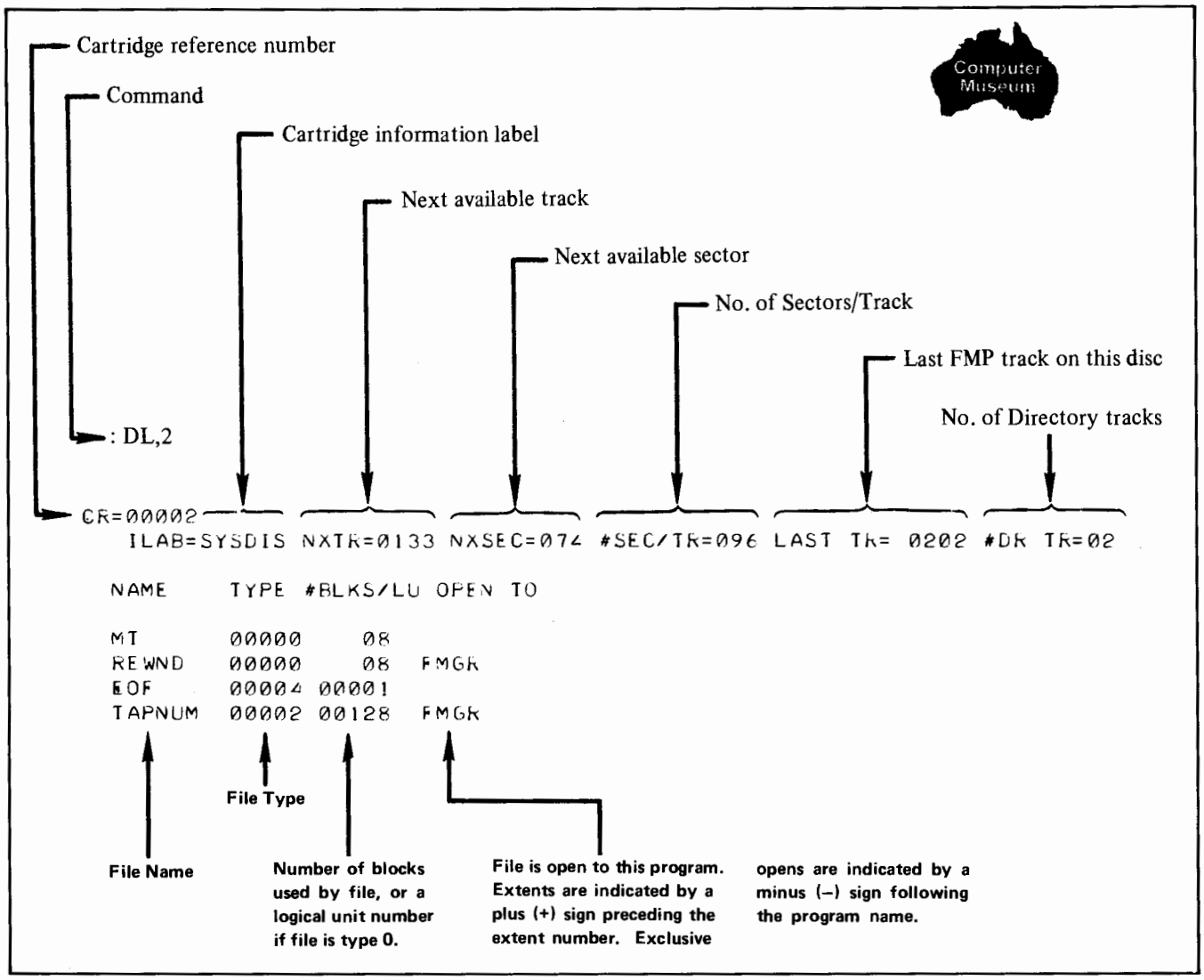


Figure 3-1. Short List

```
: DL,2,55

CR=00002
 ILAB=SYSDIS NXTR=0133 NXSEC=074  #SEC/TR=096 LAST TR= 0202 #DR TR=02

NAME    TYPE   #BLKS/LU SCODE   TRACK SEC   OPEN TO

MT      00000      08    00000
REWND   00000      08    00000                    FMGR
EOF     00004   00001    00000   0100 000
TAPNUM  00002   00128    12345   0100 002    FMGR
```

File security code — Track and sector address

Figure 3-2. Long List

DP

Purpose:

To display current values of global parameters on the log device.

Format:

:DP,[*P1* [,*P2*. . . .] . .]]

Where:

P1 and *P2* are any legal FMGR parameters. They are commonly used to display current values of global parameters.

COMMENTS

- This command causes the current value of its parameters to be displayed on the log device. For example, to display the current value of 5G and 1P, enter :DP,5G,1P.
- The severity code does not inhibit this printout.
- DP also returns the value of any parameters specified; for instance,
  :DP,HELLO
  prints the message "HELLO" on the log device.
- Numeric parameters are printed as decimal values when they are positive, as octal values when they are negative.

DUmp

Purpose:

To copy the contents of a file to another file.

Format:

:DU,*namr1*,*namr2* [,*record format*,*EOF control* [,*file #* [,*#files*]]]

– or –

:DU,*namr1*,*namr2* [,*record format* [,*file #* [,*#files*]]]
[,*EOF control*]

Where:

namr1 is the name of the file, or logical unit number from which data is to be transferred.

namr2 is the name of an existing file, or logical unit number (other than a disc) to which *namr1* is to be transferred.

record format is the record format as applied to *namr2*. Refer to Comments for more information.

EOF control saves or inhibits end-of-file marks on *namr2*. Refer to Comments for more information.

NOTE

Only one comma is required when *record format* and *EOF control* parameters are omitted. No comma is required if only one of the parameters is omitted.

file # indicates the relative file to be written in on *namr2*. This parameter must be greater than 0 if supplied. Default is 1. For example, if *file #* is 3, then 2 files will be skipped on *namr2*, and the data will be placed in the third file. If *namr2* is a disc file, the indicated number of zero length records are skipped.

#files indicates the number of files or zero length records to be transferred from *namr1*. This parameter must be greater than 0 if supplied. Default for #files is as follows (refer to Figure 3-3 in Comments):

namrl is a Disc File.

a. *file #* not supplied, and *#files* not supplied.
#files defaults to 9999 files and all of *namr1* is transferred.
b. *file #* is supplied, and *#files* not supplied.
#files defaults to 1 sub-file.

namr1 is a Non-Disc File.

a. *file #* supplied or not supplied, and *#files* not supplied.
#files defaults to 1 sub-file.

In the event it is not known how many files or sub-files exist, *#files* can be an exceptionally large number and only what is required will be transferred.

COMMENTS

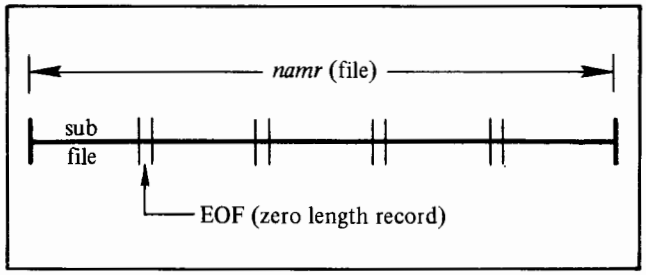


Figure 3-3. Sub-File Example

- *namr1* and *namr2* must be defined. If either one refers to an illegal logical unit number, FMGR will be aborted.

- The third parameter (*record format*) selects the record format. If this parameter is missing, default is derived from the file type as specified in the file directory entry for *namr1*. If the file type for *namr1* is not specified, final default is ASCII. The record format choices are:

  AScii indicates that ASCII records are to be transferred.

  BReloc indicates that binary relocatable records are to be transferred. A checksum is done.

  BNary indicates that binary records are to be transferred without checksum.

  BAbs. indicates that binary absolute records are to be transferred. A checksum is done.

  MTape indicates that magnetic tape ASCII records are to be transferred.

  MS indicates that magnetic tape SIO (System Input/Output) records are created on *namr2*. Standard records are expected on *namr1*.

  MSBR indicates magnetic tape SIO binary relocatable records (same as MS + BR).

  MSBA indicates magnetic tape SIO binary absolute records (same as MS + BA).

- The *EOF control* parameters *(IHibit,SAve)* controls end-of-file (EOF) marks when transferring data. (Refer to Section I, FMP Technical Discussion for more information on EOF marks.) If this parameter is missing, an EOF is written at the end of the data (end of file on input). In addition, if *namr2* is a logical unit number and refers to a punch device, or if *namr2* refers to a type 0 file which was created with the *LEader* option, then an EOF (leader) is written at the beginning of the file. In either case, any zero length records (disc file), or imbedded EOF's (non-disc file) on *namrl* are ignored. For example, DU,8,9,AS,1,3 would merge three files on LU8 into one file on LU9.

  IHibit inhibits writing an end-of-file mark after the data. Useful only when *namr2* is not a disc file. *IHibit* also inhibits leader punching.

  SAve saves any EOF's in *namrl* on *namr2*. In disc files, EOF's are saved by writing a zero length record, and interpreted by reading a zero length record. This is effective when several tapes or magnetic tape files are to be stored in one disc file and then separated again when the file is dumped. On magnetic tape the separation is by end-of-file marks, and on paper tape by leader. On relocatable files saved from the load-and-go area the EOF's will be between modules.

- The DUmp and STore Commands differ as follows:

  a. STore creates a file (unless logical unit version of *namr2* is used), while DUmp does not.
  b. STore applies *file* # and *MS* against the input file; DUmp against the output file.

  NOTE

  The DUmp routine transfers file record per record. Records longer than 128 words are truncated.

- Normally, when a TTY is used for output, it is treated as a list device. This means that its EOF default is a double space. If the TTY is to be used to punch tape, this default must be avoided using one of the following means.

  a. Use the 1H option – an EOF is not written. However, leader is not punched.
  b. Create a type 0 file referring to the TTY, and specifying *LEader* as the EOF parameter. In dumping to the type 0 file (TTY) leader will be punched before and after the data. For example:

```
:CR,TTY,I,BO,,LE
:DU,namrl,TTY
```

- The DU Command can be used to concatenate files. For example, to add file B to the end of file A the command would take the form of:

```
:DU,B,A,,2
```

File B may be any legal *namr* and the omitted parameters may be as required to correctly read file B.

EOjob (End of Job)

Purpose:

To indicate the end of a job.

Format:

```
:EOjob
```

COMMENTS

The EOjob Command causes the following actions:

1. Prints an end of job message on the list device.
* 2. Closes the list spool file.
3. Resets logical units to the default devices of the pre-job status:
   LU1 – system console (input device)
   – log device (same as input device)
   LU6 – list device
4. Sets severity code to 0.
5. Releases the Load-and-Go area (if any).
6. Clears globals 1G through 10G.
7. Resets Batch Logical Unit Switch Table to generation values.
* 8. Changes job status to CS (completed and waiting for outspooling).
* 9. If job input was from a spool file, this file is returned to the available pool.
*10. The job file (JOBFILE) is searched for a job to execute (unless end of job was requested by the JOb command) and returns to JOb command processor.
11. If job is not spooled, check the input unit; if input is from the card reader, FMP terminates, otherwise, the next command is read.

*Starred steps are performed only if the current job is spooled.

NOTE

If an EOF is encountered on the command device, the FMP will force a TR Command which will terminate the FMP.

IF

Purpose:

To allow conditional branching in the job command stream.

Format:

```
          EQ
:IF ,p1,NE ,p2[,#]
          LT
          GT
          GE
          LE
```

Where:

p1	is the first parameter which is being compared to the second parameter, *p2*.
EQ	is equal to (=).
NE	is not equal to (≠).
LT	is less than (<).
GT	is greater than (>).
GE	is greater than or equal to (≥)
LE	is less than or equal to (≤).
p2	is the other parameter being compared with *p1*
#	is the number of commands to be skipped in the command file.

COMMENTS

- The stated relationship of *p1* to *p2* is examined, and if true, then #commands are skipped in the command file. Default # is 1.

- # may be either positive or negative. −1 causes the command to be re-executed; −2 causes the preceding command to be executed; etc.
- The IF Command cannot skip beyond EOF or before start-of-file (SOF). Attempts to do so will go to EOF or SOF, but an error will not be reported.
- The IF Command is illegal from an interactive device. If a negative skip is used, the command device must be a file or it must recognize backspace record commands. Jobs that are spooled in recognize backspace record commands.
- *p1* and/or *p2* may be ASCII, null, or numeric. The following relationships hold with mixed parameter types:

  null<numeric<ASCII

- p1 and p2 may be globals.

Within ASCII parameters, the ASCII alphanumeric order sequence is used, (refer to Table E-2, Appendix E).

INitialize

Purpose:

To initialize cartridge parameters and clear a cartridge directory, or change the master security code.

NOTE

This command does not format a previously unformatted cartridge. Refer to Appendix M, Cartridge Formatting.

Format (initialize parameters, clear directory):

:IN, [*master security code*],*label1*,*label2*,*id* [,*lst trk* [,*#dir trks* [,*#sec/trk* [,*bad tracks*]]]]

Format (change master security code):

:IN,*master security code*−−*new security code*

NOTE

−−are two minus signs.

Where:

master security code is the two-character FMP master security code designated at initialization time. When changing codes, *new* implies a new code and is separated from the old code by two minus (−) signs. See caution below.

label1 is the cartridge reference label, positive for CR or negative for logical unit number If the cartridge has not been initialized, a CR label will not have been established; therefore, *label1* must be a negative logical unit number.

label2 is the new cartridge reference label and must be >0.

id is the cartridge information label (1LAB - refer to Figure 3-1/3-2), and must have the characteristics of a legal file name. Refer to text under the heading NAMR for a description of a legal file name. *id* is printed in the head of the directory listing.

1 st trk is the first track to be used on the cartridge (relative to the first track assigned to RTE; see Comments). Null defaults to track 0.

#dir trks is the number of directory tracks. Null defaults to 1 directory track. This parameter must be less than 48.

#sec/trk is the number of 64-word sectors per track. Null is allowed only for cartridges on same channel as LU2 or LU3 in which case this parameter is ignored.

bad tracks is the bad track list. Up to six track numbers may be entered, separated by commas.

INITIALIZE PARAMETER COMMENTS

- Before a new cartridge can be initialized with the IN Command, it must first be mounted with MC Command. For a brand new cartridge, refer to Appendix M.

- On re-initialization, FMGR first checks that all files are closed (refer to LOCKING). If not, FMGR error −8 results. FMGR then checks the new track parameters entered. If the new parameters raise the first track, or lower the directory into an existing file, the message FMGR 060 is printed. This is a caution message that allows you to abort the initialization if incorrect track parameters have inadvertently been entered. Entering NO causes the initialization to be aborted (the ?? Command also aborts the initialization). **Entering YES causes all files on the disc to be purged.**

- Initialization and re-initialization of the system and auxiliary discs (LU2 and LU3) also present possible track assignment conflicts. If additional tracks are requested (i.e., first track parameter lowers FMP area into RTE system area), FMGR checks on the availability of the required tracks, and assigns them if they are available. If all the required tracks are not available, the highest required but not available track number is reported with a FMGR 059 error, and the IN Command is aborted. The IN Command can then be re-entered with the first track parameter equal to the track reported plus one. If fewer tracks are assigned during re-initialization, the excess tracks are returned to the RTE System. Note that if the first track is lowered, the disc must be packed to use the new area (see the PK Command).

- Bad track information is returned to the user during RTGEN or, if discovered by the FMP, as an error return to the caller. Bad track information is entered using the FMGR Initialization Command. This information is kept in the affected disc's first directory entry. If, at creation or during a packing move, the area for the file includes a bad track, the first track of that file will be increased until the file no longer contains any bad tracks. If CREAT is to use the rest of the disc, the whole area will be above the highest bad track. If, during packing, a file is found to include a declared bad track, the file will be purged. (Note: This can only happen through an explicit declaration of a bad track by an operator who knows the system security code.) In this manner, only the PACK and CREAT routines need be aware of bad tracks. Bad tracks discovered by the FMP result in an error return to the caller.

- When the system disc is initialized the first time, the first track must be at least 8 more than the last system track used at generation.

NOTE

See Section X for first time initialization.

CHANGE SECURITY CODE COMMENTS

- The current master security code is changed when two new characters are supplied for the *new security code parameter.* Exceptions are as follows:

  a. Colon, comma, or a leading blank are not allowed.
  b. Non-printing characters are acceptable.
  c. The master security code may be changed to zero (no security) by specifying blanks.

- If the current master security code is zero (no security) then any two characters may be used for the security parameter; however, two characters must be entered to maintain positional relationship.

Be sure to remember the new code. Once it is entered, it cannot be obtained with any FMGR command.

JOb

Purpose

To define the beginning of a job and to set up parameters for that job.

Format

:JOb[,*name*[:*hr:min:sec*] [,*job priority* [,*spool priority* [,*NSpool*]]]]

Where

name	is the job name.
hr:min:sec	is the *Execute* time limit for the job, expressed in hours, minutes, and seconds. (Default no limit.) An executing job is terminated when it runs longer than the specified time.
job priority	is the job priority. Priority ranges from 1 (highest) to 9999 (lowest). Default priority is 9999.
spool priority	is the output spool priority. Same limits as job priority. Default is *job priority.*
NSpool	is a key word entry and may be anywhere after the job name. This parameter, if present, terminates the parameter list The function of *NSpool* is to inhibit automatic creation of a list output spool.

COMMENTS

The JOb Command causes the following actions:

1. If the previous job was not terminated with an EOj command, the EOj processor is called.
2. Sets logical units to default devices:
   LU1 - system console (default input)
   log device (same as input device)
   LU6 - list device
3. Clears globals 1G through 10G.
4. Resets Batch Logical Unit Switch Table to generation values.

* 5. Unless NSpool is present, a list output spool is assigned to LU6; a JOB, ON message is sent to this file to be printed on a new page.

* 6. If the JOb command was read from disc, a job input file is assigned to logical unit 5.

*Starred steps occur only if the job is spooled.

LG

Purpose:

To allocate or release a group of disc tracks for load-and-go operations.

Format:

:LG[,*#tracks*]

Where:

#tracks	is a number indicating how many tracks are to be reserved for Load-and-Go binary code

COMMENTS

- The Load-and-Go area is cleared at each EOJ and also when assigned The area is returned to the system when an LG Command without parameters is received.

- This LG Command is the same as the RTE system LG Command and may generate the same error messages.

LIst

Purpose:

To print the contents of a file on the current list file.

Format:

:LI,*namr* [,*Source* | ,*Binary* | ,*Directory*]

Where:

namr is the name of the file, or a logical unit number.

Source indicates ASCII source format.

Binary indicates binary format.

Directory indicates that only the header containing file information (directory entry) is to be printed.

NOTE

Only the first letter is required for the above format parameter.

COMMENTS

- If *namr* refers to a file, the file is listed with the following heading:

  file name T = *file type* IS ON CR *cartridge label* USING *nnnnn* BLKSR = *record length*

- If *namr* is a logical unit number, then asterisks (*) are printed in place of the file name as shown below.

  ***** T = 0 IS ON LU*xx*

- The default format specifications are related to file types. If the format is not specified, source is assumed for type 0, 3 and 4 files. Binary is assumed for all other type files.

- If the list device is a TTY, the listing starts in column one; otherwise, each line is preceded by two blanks.

- Source format consists of a line number followed by the ASCII text.

- Binary format for each record consists of:

  a. A blank line.
  b. The record number: REC#*nnnnn*
  c. A blank line.
  d. As many lines as needed for the record with eight words per line. Each word is printed in octal followed by ASCII if a printable ASCII character is found (non-printing and lower-case characters are filled with blanks). The octal and ASCII representations are separated by a vertical column of asterisks. Refer to Figure 3-4 for an example printout.

  Zero length records will be printed out as just a record number (parts a, b, and c only).

```
REC# 00001

010400 020000 ..... 000000*     CDVR12
000007 000000 ..... 000000*
000000
```

Figure 3-4. Example List of a Binary Record

- List does not support absolute binary files from paper tape.
- Binary format for a full line is 72 characters on a TTY. On any other device, 74 characters per line are sent. If necessary, multiple lines are printed up to a maximum of 128 words. Lines are truncated after the last non-blank character. The asterisk is treated as a blank for this test.
- List does not support records of more than 128 words, and if source mode, truncates the total line to 72 characters.

LLu (list file change)

Purpose:

To change the current assignment of the list file.

Format:

:LL,*namr*

Where:

namr is the name of a file, or a logical unit number.

COMMENTS

- *namr* may refer to any existing file or logical unit.
- The list file is where the commands AN, LI, CL, DL, EO, and JO direct their output. The list file is also where job messages are printed.

LOglu (log device change)

Purpose:

To change the logical unit number of the system log device.

Format:

:LO,*lu*

Where:

lu is the logical unit number of the new TTY type log device (i.e., DVR00 or 05). Note that *lu* can not be a file name.

COMMENTS

- None

LS

Purpose:

To set up base page word 1767B to point at the specified disc and track.

Format:

:LS[,*lu*,*track*]

Where:

lu is the logical unit number of the disc containing the source file.

- *lu* = 2 or 3 system or auxiliary disc units.
- *lu* = 0 eliminate the current source file designation.

track is the starting track number of the source file (in decimal).

COMMENTS

- If the *lu* or *track* parameters are missing, or if *lu* = 0, the word is cleared.
- This LS Command is the same as the RTE system LS Command, and it can generate the same error messages.

NOTE

LS format is not the same as type 3 and above files.

LU (Spool Setup)

Purpose:

To set up for inspooling or outspooling from/to other than the standard input or standard output devices.

Format (Input only):

:LU,*lu*[,*namr*]

Format (Output and Output/Input):

:LU,*lu*,[*namr*],[*attribute*],[*lu#*],[*priority*]

Where:

lu is the logical unit with which the spool file is to be accessed, (0-77).

namr is the file name to be used in the spool. If 0, then the *lu* reverts to system definition.

If *namr* is defaulted, then a system assigned spool file is used. (*save* is illegal in this case.)

attribute can be a combination of up to three of the following. (Default is read-only, no buffering, outspool, no-hold, and save.)

HO implies a hold on the file. It is not to be queued for outspooling until either the hold is removed or the spool is closed. Normally, spool files are queued as soon as possible.

WR implies write-only, no hold.

BO implies read-and-write and hold.

WH implies write-only and hold.

BU implies buffering.

PU implies the file is to be purged. *namr* must be supplied.

ST implies a standard file, (defined in Section V, Formats of Spool-Files).

lu# implies the file is to be outspooled if supplied. This is the destination logical unit, and it is not translated.

priority is the outspool priority. Default is the NSpool priority defined by the JOB command.

COMMENTS

- Internally the spool system will assign a logical unit to the spool. The LU Command sets up a translation of the required spool *lu* to the assigned one for batch jobs only. The real lu may be accessed by referring to global 0S. The name of the spool file can be accessed by reference to global 1G.
- If *namr* is an lu, then the translation is set up for batch jobs no spool access is implied.
- Attention should be paid to actual device types corresponding to the logical units for the first parameter (*lu*). If the user wants to simulate a punch, for instance, it may be necessary to use the actual punch logical unit since the system simulates the device type of the specified logical unit. If an unassigned logical unit is specified, the device type defaults to magnetic tape. The first *lu* must not be 2, 3, 5 or any DV30, 31, 32, 33 or 34 type device or spool pool *lu*.
- For standard format files to be outspooled, the outspool *lu#* may contain a subfunction code that is used by the outspool program (SPOUT) to write lines to the selected output device. For example:

```
:JO,LUEX1
:LU,4,RELOC,WRSTSA,104B    Set up file for punch output to be outspooled; 104B is subfunction.
:LU,30,TEST,SA             Sets up input to ASMB
:RU,ASMB,30,6,4
:EO
```

- If the current job is not spooled, LU will substitute one logical unit for another. For example, if list output is specified to logical unit 10, but the user wants list output to go to logical unit 8, the command is specified as:

:LU,10,8

In this case, no file access is performed.

Examples

1. :LU,4,,WRST,4 — Create a spool for the punch using a system assigned spool file.

2. :LU,15,DATA,WRSAST — Establish a spool reference to the existing file DATA. It will be used for writing and will be saved when closed at the end of the job.

3. Assume a created file called TEST containing the program to be assembled:

```
:JO,LUEX2,NS
:LU,30,TEST,SA      Set up Assembly input file
:LU,6,LIST,WRSA     Set up Assembly list file
:LU,4,WR            Set up punch output file;
                    do not save punch output
:RU,ASMB,30,6,4     Assemble
:CS,6,END
:CA,1,1             Set up loop to print listing
                    3 times
:LU,30,LIST,SA,6    Set up to outspool Assembly listing
:IF,1G,EQ,3,2       Check if done
:CA,1,1G,+,1        Increment counter
:IF,,EQ,,-4         Loop back
:EO
```

Mount Cartridge

Purpose:

To notify the FMP that a cartridge has been mounted and is available for use.

NOTE

A previously unused cartridge must be formatted before it is mounted. Refer to Appendix M, Cartridge Formatting.

Format:

:MC,*lu*[,*last track*]

Where:

lu is the logical unit number of the cartridge being mounted.

last track is the last track on the cartridge available to the FMP.

NOTE

For a new cartridge, the *last track* parameter establishes the last track. For a previously initialized cartridge, use the last track reported in the DC Command when the cartridge was removed.

COMMENTS

- The following conditions apply to the *last track* parameter

  a If *lu* is 3, then *last track* is ignored and the RTE System-defined last track is used
  b If *last track* is absent and the disc driver is DVR31 (moving head), the *last track* parameter is defaulted to the RTE System-defined last track
  c *last track* must be supplied if the disc driver is DVR30 (fixed head), and *lu* is not 3

- This command places the cartridge at the bottom of the disc directory.

NOTE

If you desire to have this cartridge moved to the top of the directory, refer to the DC Command

- The *lu* parameter may be negative (to conform to the *label* definition) or positive, but in either case it is a logical unit and must refer to a disc.
- If a cartridge is already mounted at *lu*, or the cartridge being mounted has the same label as a currently mounted cartridge, FMGR error 012 results and the command is aborted. Refer to the DC Command.
- FMGR checks the cartridge's directory (at the last track) to determine if the cartridge has been previously initialized. If the cartridge has been previously initialized, the cartridge label must be unique. If not unique, FMGR 012 results. If the cartridge has not been previously initialized, it is locked (i.e., made unavailable). An INitialize Command must be issued to initialize and unlock the cartridge.

Move Relocatable

Purpose:

To transfer the file at *namr* to the load-and-go area.

Format.

:MR,*namr*

Where:

namr is the name of the file, or logical unit number to be transferred.

COMMENTS

- A checksum is performed on the data transferred.
- *namr* may contain one or more relocatable modules.
- If an EOF is detected on *namr* and the last record transferred was not an END record (see Appendix F), FMGR will print:

FMGR 006

and suspend. (If suspension occurs within a job, FMGR will be aborted.) The operator should load the remainder of the relocatable module and enter

*GO,FMGR

on the system TTY.

Move Source

Purpose

To transfer a file to a logical source (LS) track before operating on it with EDITR or the Assembler or one of the compilers.

Format

:MS,*namr* [,*prog name* [,*IH*]]

Where

namr is the name of the file, or logical unit number to be transferred.

prog name is the name of a program. If given, the LS tracks are assigned to that program, if not given, the LS tracks are assigned to the program EDITR.

IH is a literal. If supplied, the LS word on the base page is not set, and the LS command is required.

COMMENTS

- When this command is typed, the first LS track is reported on the log device as follows:

FMGR 015

LS LU *n* TRACK *nnn*

If FMGR was internally scheduled with waiting (refer to ON, FMGR), the scheduling program may obtain the LU and track number by calling RMPAR after it regains control. The LU and track numbers are returned in RMPAR parameter number two in the same format as the base page LS word. That is:

Bit 15 = 0 is logical unit 2

Bit 15 = 1 is logical unit 3

Bits 14-7 is the track number

Bits 6-0 is the sector address (always = 0).

- The MS command will not support files with record lengths greater than 128 words.
- It is not advisable to use FMGR as *prog name*, because all tracks assigned to FMGR are released upon entry to the MS command, and on termination of FMGR.
- The Assembler and compilers (ASMB, FTN, FTN4, and ALGOL) also release any tracks assigned to them upon termination. Consequently, if another program in the RTE System is requesting tracks, your source program could be lost. (It is still accessible in the file, however.)

OFf program

Purpose:

To immediately terminate the named program and, if temporary, to remove the program from the system.

Format:

:OFf,*prog*

Where:

prog is a program that obtained its ID-segment within the current job

COMMENTS

- This command clears *prog's* ID-segment and returns its program tracks to the system. It also returns tracks it may have obtained dynamically.
- This OFf Command is the same as the RTE System OF,*name*, 8 Command, and it may generate the same messages. (For example, if the conditions are such that the use of OF,*name*, 8 would generate an abort message on the system console, the use of the OFf Command will likewise produce the same abort message.)
- Removal of a program by means of the OFf Command will also cause an abort message to appear on the system console, unless the program is a background segment.

PacK disc files

Purpose:

To close up gaps in between files and utilize the tracks left from purged files.

Format:

:PK[,*label*]

Where:

label is the cartridge reference label, positive for CR or negative for logical unit number. Note that if *label* = 0, or is not entered (default = 0), then all cartridges are packed. Exceptions are described in Comments.

COMMENTS

- The PK Command purges all files which contain bad tracks. The bad tracks must have been previously declared in the Initialize Command. Refer to "FMP Technical Discussion" in Section I for more information on bad tracks.

NOTE

If you do not want the file with bad tracks purged, save it on another cartridge.

- PK moves each file, if necessary, and updates the file's directory entry to reflect the new address. When the end of the file directory is sensed (i.e., all files have been packed), the PK Command then packs the directory removing any entries for purged files.
- A non-lock error (−8) or a disc not found error (−6) will be followed by an additional message indicating which disc could not be locked or found. When the *label* parameter is not specified, the cartridge will be reported as a −LU. If *label* is specified, the report will be the cartridge reference number (CR). Examples:

:PK,30 (30 not previously defined)
FMGR −006
FMGR 030

```
:PK
FMGR -008 (not lockable)
FMGR -002 (LU2)
```

In the last example LU2 will not be packed because there is at least one open file on it; however, all other mounted cartridges will be packed

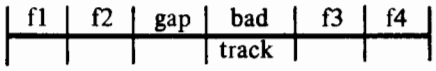
- If during execution of the PK Command, the system should fail, it is possible to lose, at most, one file. The following data is provided to aid in determining which file, if any, may have been lost.

a. A system failure during any part of a pack will leave the disc locked. This lock may be cleared by issuing a Dismount Cartridge Command (of course, you must then issue a Mount Cartridge Command unless the cartridge is LU2 or LU3).

b. Files are moved into gaps (created by purges) one at a time as follows:

1. If the file is smaller than the gap, then the file exists in two places. The original place still contains good data, and is accessible through the directory - no file is lost in this case.

2. If the file is larger than the gap, then when the gap is filled, a portion of the file being moved is overlayed. If a system failure occurred at this time, the directory could point to bad data. The affected file may be found by examination of the length, first track, and sector addresses the directory list provides. The file following any gap is the affected file. Its size may be compared with the gap to see if it was lost. Remember, the file's directory entry is updated last and any bad tracks affect the gaps left between files as shown:



If f3 fits in the gap, f3 will be transferred; if not, the gap will be retained.

3. The directory entry is updated in place after the file is successfully moved.

c. After all files are moved, the directory is packed as follows:

1. The directory entries are first written on a disc track obtained by FMGR from the system. Only good entries are transferred; purged directory entries are omitted. A system failure at this time would lose nothing.

2. This track is passed to D.RTR which in turn writes it into the directory area, overlaying the old directory. A system failure at this time could result in the directory containing duplicate entries This condition may be corrected by

a. Storing the affected files in new files with either different names or on different cartridges.

b. Purging the affected files

c. Changing the names of the moved files back to the original names.

PAuse

Purpose:

To suspend execution of the current job and transfer control to a specified device, and, optionally, to print additional text.

Format:

:PAuse[,*lu*,[,*message*]]

Where:

lu	is the logical unit number of the device to which control is being transferred. Default is to the log device.
message	is an optional message.

COMMENTS

- This command causes a TR, *lu* where the default *lu* is the log device. The whole line is also printed on the log device.
- When job processing is suspended with the PAuse Command, it may be continued by entering TR on the device to which control was transferred

PUrge disc file

Purpose:

To remove a file and all its extents from the system. Note that this command is the only method available to remove a type 0 file.

Format:

:PU,*namr*

Where:

namr is the name of the file to be purged.

COMMENTS

- If the file being purged is the last file on the disc (excluding a type 6 file), all area from the last unpurged or type 6 file up to and including the purged file, is returned to the system. If a purged file is not the last file, then its area may be recovered only by packing, or by purging all files after it in the directory. Type 6 file area may only be recovered by packing.

ReName disc file

Purpose:

To change a file name to a new name.

Format:

:RN,*namr,nuname*

Where:

namr is the existing file name and parameters.

nuname is the new name unique to the cartridge.

COMMENTS

- The file specified by *namr* must not be open when this command is issued. If *namr* has a non-zero security, the proper security code must be included. If *namr* includes a cartridge reference number, only that cartridge is searched for *namr;* otherwise, all mounted cartridges are searched. In the case where all cartridges are searched, only the first file found with the given *namr* will be changed.

- *nuname* is the new name of the same file. Subparameters cannot be changed with this command.

Restore Program

Purpose:

To restore a program (file) saved by the FMP, so that it may be run in the RTE System.

Format (to restore a program):

:RP,*namr*

Format (to assign *program's* ID segment to *namr*):

:RP,*namr,program*

Format (to release *program's* ID segment):

:RP,,*program*

Where:

namr Is the name of a type 6 file on LU2 or LU3 which was created on the current system using the SP Command. Note that a logical unit number is not allowed.

program is a program name.

COMMENTS

- The program assigned as an RTE System program may be accessed by all the usual RTE System Commands. The program is given the same time parameters and priority assignment as when it was saved.

- *namr* must refer to a type 6 file on LU2 or LU3 which was created on the current system using the Save Program Command. (Note that *namr* may not be

a logical unit number.) The first five characters of *namr* define the program name which need not be the name the program had when it was saved with the SP Command For example, a program with an actual name of SMPLE (NAM statement in program) is saved as SMPLE The file name is later changed from SMPLE to TEST01 with the

RN,SMPLE,TEST01

command. When it is restored with the RP Command, TEST01 (the type 6 file name) would be used. Then when the program is scheduled in the RTE System,

*ON,TEST0

would be used.

- If the program saved is still in the system, and the program is restored using the same name it was saved with, FMGR error 023 results (Duplicate Program Name).

- In the first Format example shown, the first five characters of the file name will be the program name. A blank ID segment is established to point to the tracks belonging to the named file. If FMP cannot find a blank ID segment, an FMGR error 014 results.

- In the second Format example shown, the program called *namr* would use *program's* ID segment. *Program* must be inactive and must have been set up by a previous RP Command. *Program* is removed from the RTE System. If *program* does not have an ID segment, an FMGR error 009 results and RP reverts to the first Format. If *program* is not inactive, FMGR error 018 results. This problem is solved by issuing the RTE System Command:

OF,*program*,1

or the FMGR OFf Command.

- If the *program* has completed and it is desired to return the ID segment to the RTE System, the third format is used. If *program* did not have an ID segment, FMGR 009 results but does not transfer to the log device.

- The file *namr* may be purged while an ID segment points to it, and while it is running in the system. The program is not affected in any way (in the RTE System) by this action. To recover the file's space on the cartridge (with the PacK Command), the file's ID segment must first be released with the command:

:RP,,*program*
or
:OF,*program*

Failure to return the ID segment before issuing the PacK Command results in the FMGR 011 error.

- The RP Command does not permanently modify the RTE System area of the cartridge Therefore, any program assigned will not be present on rebooting of the system.

Release Tracks

Purpose:

To release all disc tracks assigned to a program.

Format:

RT, *name*

Where:

name is the name of the program that is to have its tracks released.

COMMENTS

- If the program is not dormant, the request is illegal.

- If the program is dormant, all tracks assigned to that program are released.

- If any tracks are released as a result of this request, all programs in disc track allocation suspension are rescheduled.

RUn

Purpose:

To initiate execution of a specified program or transfer a file.

Format:

:RUn,*name*[,*parameters*]

Where:

name	is either a system program or a legal FMGR name.
parameters	are the parameters to be passed to the program when it is run. Up to 5 parameters may be specified depending on the program; up to 9 if used as globals (see Comments). A parameter is a 1-word integer or 2 ASCII characters converted by the system to an integer.

COMMENTS

- This command causes a search of the system ID-segments for the named program. If not found, an RP Command is attempted. If a file is found that is not type 6, the RUn Command will be interpreted as a TRansfer Command and the parameters will be set into the appropriate globals (see also the TR and SE Commands). When interpreted as TR, parameters may be any legal FMGR parameters. If a program is set up from a file, it will have its ID-segment deallocated when it is finished running and any tracks assigned by the program are released.

Save Program

Purpose:

To place a disc resident program into a type 6 file.

Format:

:SP,*namr*

Where:

namr	is the name of a disc resident program. Note that a logical unit number is not allowed.

COMMENTS

- *namr* is created as a type 6 file, and the current system program with the same name as *namr* (first five characters), is stored in *namr*.

- A program's name is a maximum of five characters long (e.g., SMPLE). A file name can be six characters long. Therefore, it is possible to create a type 6 file called SMPLE1 for one version of the program, and another called SMPLE2 for a second version of the same program. If the program's name is less than five characters long, the program must be saved using only the characters in the name. For example, if a program's name is LOCF, it must be saved using the

  :SP,LOCF

  command. To change the program name use the RN Command, thus,

  :RN,LOCF,TEST01

  When restoring the program use

  :RP,TEST01

  When rescheduling the program in the RTE System use

  *ON,TEST0

- The SP Command is usually used in conjunction with the RP Command. For example, you have a source program that you have debugged, edited, compiled, and loaded, and have successfully run in the RTE System. You want to save this program in its present loaded form for a future use, without having to recompile, reload, etc. Issue the SP Command using the program's name, and a type 6 file with the same name is created. Later, to rerun the program, use either the RU Command, or both the RP and RU Commands. The program can be saved while it is running. After it has been saved, it can be aborted with the command:

  :OF,*program*

NOTE

The SP Command will store the program file on any mounted cartridge (through the *label* subparameter of *namr*). However, the file must be located on LU2 or LU3 before the RP Command will accept the file. The file may be moved with the ST Command.

- The following *namr* parameters are treated as shown:

  security code is defaulted to 0, otherwise user selected.

  label is defaulted to −2.

  file type is forced to type 6.

  file size is forced to the required size.

  record size is forced to 128.

SAve

Purpose:

To save the logical source (LS) or load-and-go area (LG) in a file.

Format:

:SA$\begin{matrix},LS \\ ,LG\end{matrix}$,*namr*

Where:

LS indicates the logical source area.

LG indicates the load-and-go-area.

namr is the name of a file (created by this command), or a logical unit number.

COMMENTS

- The SA,*LG* implies a checksum. After each end record, including the final end record, either a zero length record (disc file), or an EOF (non-disc file) is written, depending on the file.
- The file size, if not specified, is defaulted as follows:

  a. The SA,*LG* Command computes the maximum possible file size from the amount of LG area used, and uses this size to create the file. Therefore, extents will never be created, and the file will only be as long as needed.
  b. The SA,*LS* Command sets the size to one-half the number of blocks on a system disc track.

- After the EOF is written, its position is checked. If extents were created, the file size is not shortened. If the EOF is within the main file, the remaining unused disc space is returned to the system.
- The file type, if not specified, is defaulted as follows:

  :SA,*LS* – defaults to type 4 (source)

  :SA,*LG* – defaults to type 5 (relocatable)

- If there is not enough cartridge space available to accommodate the file, an FMGR −6 error results, and any portion of the file already saved is purged.

NOTE

The SAve Command does not support records longer than 128 words.

SEt global parameters

Purpose:

To allow the user to set global parameters 1G through 9G.

Format:

:SEt[,*p1*[,*p2* . . . [,*p9*]] . . .]

Where:

p1 through *p9* are the parameter values to be converted to global parameters. (See "Comments" for details.)

COMMENTS

- This command sets the values P1 through P9 into 1G through 9G, respectively. If any parameter is absent, its current value remains unchanged. If all parameters are absent, 1G through 9G are nulled. (1G, is equivalent to ,, if 1G is null.)

STore

Purpose:

To transfer or store records from a file or logical unit number, to another file or logical unit number. Note that *namr2,* if not a logical unit, is created by this command.

Format:

:ST,*namr1*,*namr2*[,*record format*,*EOF control*[,*file #*[,*#files*]]]

– or –

:ST,*namr1*,*namr2* [,*record format*[,*file #*[,*#files*]]] / [,*EOF control*]

Where:

namr1 is the name of the file, or logical unit number from which data is to be transferred.

namr2 is the name of the file, or logical unit number to which *namr1* is to be transferred. *namr2* is created by this command.

record format is the record format as applied to *namr1*. Refer to comments for more information.

EOF control saves or inhibits the end-of-file marks contained in *namr1.*

NOTE

Only one comma is required when *record format* and *EOF control* parameters are omitted. No comma required if only one of the parameters omitted.

file # indicates the relative position the file is to be read from on *namr1*. This parameter must be greater than 0 if supplied. Default is 1. For example, if *file #* is 3, then 2 files (or zero length records) will be skipped on *namr1*, and the third file will be read.

#files indicates the number of files or zero length records to be transferred from *namr1.* This parameter must be greater than 0 if supplied. Default for *#files* is as follows (refer to Figure 3-5 in Comments):

namr1 is a Disc File.

a. *file #* not supplied, and *#files* not supplied.
#files defaults to 9999 files and all of *namr1* is transferred.
b. *File #* is supplied, and *#files* not supplied.
#files defaults to 1 sub-file.

namr1 is a Non-Disc File.

a. *File #* supplied or not supplied, and *#files* not supplied.
#files defaults to 1 sub-file.

In the event it is not known how many files or sub-files exist, *#files* can be an exceptionally large number and only what is required will be transferred.

COMMENTS

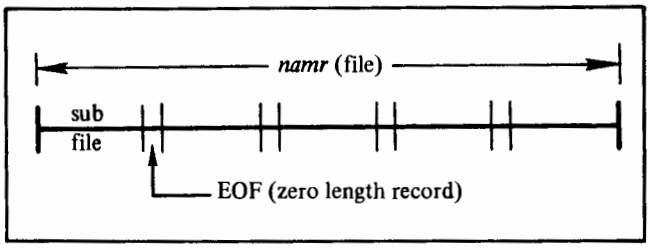


Figure 3-5. Sub-File Example

- *namr1* must be defined. If either *namr* refers to an illegal logical unit number, FMGR will be aborted.

- The file type parameter for *namr2* is defaulted as follows:

  To *namr1*'s type if *namr1* is a disc file.

  To 3 when using MT, MS, AS.

  To 5 when using MSBR, BR.

  To 7 when using MSBA, BA.

  To 3 when none of the above is specified.

  File size parameter for *namr2* is defaulted to one-half a system disc track.

- The *namr2* parameter can be a logical unit number (non-disc file), but not a type 0 file.

- After the EOF is written, its position is checked. If extents were created, the file size is not shortened. If the EOF is within the main file, the remaining unused disc space is returned to the system.

- The third parameter selects the record format. If this parameter is missing, default is derived from the file type as specified in the file directory entry for *namr1*. If the file type for *namr1* is not specified, final default is ASCII. The record format choices are:

  AScii indicates that ASCII records are to be transferred.

  BReloc. indicates that binary relocatable records are to be transferred. A checksum is done.

  BNary indicates that binary records are to be transferred without checksum.

  BAbs. indicates that binary absolute records are to be transferred. A checksum is done.

  MTape indicates that magnetic tape ASCII records are to be transferred.

  MS indicates that magnetic tape SIO (System Input/Output) records are expected on *namr1*. Standard records are written on *namr2*.

  MSBR indicates magnetic tape SIO binary relocatable records (same as *MS* + *BR*).

  MSBA indicates magnetic tape SIO binary absolute records (same as *MS* + *BA*).

- The *IHibit, SAve* portion of the third parameter controls end-of-file (EOF) marks when transferring data. (Refer to "FMP Technical Discussion" in Section I for more information on EOF marks.) If this parameter is missing, an EOF is written at the end of the data (end-of-file on input). In addition, if *namr2* is a logical unit number and refers to a punch device, then an EOF (leader) is written at the beginning of the file. In either case, any zero length records (disc file), or imbedded EOF's (non-disc file) on *namr1* are ignored. For example, DU, 8, 9, AS, 1, 3 would merge three files on LU8 into one file on LU9.

  IHibit inhibits writing an end-of-file mark after the data. Useful only when *namr2* is not a disc file.

  SAve saves any EOF's in *namr1* on *namr2*. In disc files, EOF's are saved/interpreted by writing/reading a zero length record. This is effective when several tapes or magnetic tape files are to be stored in one disc file and then separated again when the file is dumped. On magnetic tape the separation is by end-of-file marks, and on paper tape by leader. On relocatable files saved from the load-and-go area the EOF's will be between programs.

- The DUmp and STore Commands differ as follows:

  a. STore creates a file (unless logical unit version of *namr2* is used); DUmp does not.

b. STore applies *file* # and *MS* against the input file; DUmp against the output file.

NOTE

The STore routine transfers file record per record. Records longer than 128 words are truncated.

- The input file can be declared the system TTY (*namr1* = LU1). For example, the ST Command is entered on the system TTY as follows:

:ST,1,FILE1 : : : 4:2

After the last subparameter, carriage-return, line-feed is entered. When this is completed the TTY just sits there waiting for your input (the colon prompt is absent). Type in as many lines as desired (extents are automatically made if required). When the file is completed, enter Control D for the EOF mark. Control will then return to FMGR (as indicated by the colon prompt).

SeVerity code change

Purpose:

To change the system log severity code to a new number.

Format:

:SV,*number*

Where:

number is the new severity code number.

0 – All commands are echoed on the log device. Any error causes an appropriate error number to be printed.

1 – Inhibit command echo on log device.

2 – Inhibit error messages to log device, unless the error is severe enough to cause control to transfer to the log device for a command input.

If a job is active and an error would cause transfer to the log device, the job will be aborted.

3 – Same as 2, except the transfer to the log device will take place even though the job is active (i.e., the job is not aborted).

COMMENTS

- If an error is detected, and the echo is inhibited, the erroneous command will be printed prior to the error message.

TEll operator

Purpose:

To send a message to the operator via the System console.

Format:

:TE, *message*

Where:

message is the message (verbatim) that is to be sent to the operator.

COMMENTS

- The whole line is sent to the system console.
- The message must meet command syntax requirements.

Time Limit

Purpose:

To set a run-time limit for programs executed by using the RUn Command.

Format:

:TL:*hr:min:sec*

Where:

hr:min:sec is the run-time limit being established, expressed in hours, minutes, and seconds.

COMMENTS

- The limit is the remainder of the job time (from JOb card) unless this command sets a smaller limit.

TRansfer control

Purpose:

To transfer FMGR control to a file logical unit, passing parameters as globals.

Format:

:TR [,*namr* / ,*–integer*] [*parameters*]

Where:

namr is the name of a file, or a logical unit number.

–integer is a negative integer that denotes a transfer back that many files.

parameters are the parameters to be set into the globals (1G through 9G).

COMMENTS

- There are three variations to the TRansfer Command.

  a. TR,*namr*. This command transfers control to *namr* (either a file or a logical unit number). Before the transfer is executed, the *namr* in control is saved in a stack. If *namr* is a disc file, the current record number is also saved.
  b. TR (no *namr*). This command causes control to be returned to the previous input *namr*. If it was a disc file, the records already read are skipped. If there was no previous input *namr*, the FMGR is terminated.
  c. TR,*-integer*. This command causes the specified number of transfers (e.g., TR,-2) with no *namr* (jump back through stack) to be executed.

- Up to 10 input *namrs* may be stacked and then returned to.
- If an error is detected at any time, and it is severe enough to transfer to the log device, a :TR,LOG (unless current input is from the log device) is executed. As many commands as desired may be entered from the log device followed by a TR Command with no *namr* to get back to the statement following the erroneous statement.

- The TR Command also sets globals (optional). If no parameters are supplied, the globals are not nulled as they are in the SE Command. The TR Command can set globals in all cases (e.g., the TR,,1Ø will transfer back and set 1G to 1Ø).

- A null command is interpreted as a TR command thus:

```
:TR,X
```

is the same as

```
:,X
—or—
::X
```

TRANSFER COMMAND EXAMPLES

The following examples show a dialogue between the system and operator. For clarity, the operator responses are shaded.

a. Suppose several subroutines have been saved in the file system and are often needed by programs being developed. Suppose these subroutines are named SUB1 through SUB10 and that SUB1 calls SUB2 through SUB5 and that SUB6 calls SUB1 and SUB7 through SUB10.

Given this, we create the following files:

```
file #1     SUBS1
            :MR,SUB1
            :MR,SUB2
            :MR,SUB3
            :MR,SUB4
            :MR,SUB5
            :TR (or ::)

file #2     SUBS6
            :MR,SUB6
            :TR,SUBS1 (or ::SUBS1)
            :MR,SUB7
            :MR,SUB8
            :MR,SUB9
            :MR,SUB10
            :: (or :TR)
```

Then, with a program requiring SUB1 in the load-and-go area, we can run the FMGR as follows:

```
*ON,FMGR
:LG,3
:RU,FTN4,99
:TR,SUBS1 (or ::SUBS1)      Transfer to SUBS1
:MR,SUB1
:MR,SUB2
:MR,SUB3
:MR,SUB4                    Echo of SUBS1 Commands
:MR,SUB5
:TR (or ::)                 Transfer back to TTY
:RU,LOADR,99
:RU,10G
:EX                         Terminate FMGR
$END FMGR
```

If we needed SUB6, the dialogue would have been:

```
*ON,FMGR
:LG,3
:RU,FTN4,99
:TR,SUBS6 (or ::SUBS6)      Transfer to SUBS6
:MR,SUB6                    Echo from SUBS6
:TR,SUBS1 (or ::SUBS1)      SUB6 transfers to SUBS1
:MR,SUB1
:MR,SUB2
:MR,SUB3
:MR,SUB4                    Echo from SUBS1
:MR,SUB5
:TR (or ::)
:MR,SUB7
:MR,SUB8
:MR,SUB9                    Continuation of SUBS6
:MR,SUB10
:TR (or ::)                 Return to TTY
:RU,LOADR,99
:RU,10G
:EX                         Terminate FMGR
$END FMGR
```

b. We could also save a segmented program and then to restore it, we generate the file:

RSSEG

```
:RP,MAIN
:RP,SEG1
:RP,SEG2
:TR (or ::)
```

which might be invoked to restore the program.

c. If we inadvertently transfer to, for example, SUB10) from Example a), from the log device:

```
:TR,SUB10                   Transfer
```

A%&#ZZX@$SUB10WT@	Echoes ASCII translation of NAM record.
FMGR 010	Error Message
A?	Error Message (colon missing)
:	On log device

A TR at this point would just go back to SUB10; what we need is to go back to the file prior to SUB10, so we enter:

:TR,–2(on log device)	Transfer back to file prior to error file
: etc. . .	

??

Purpose:

To expand the last error message.

Format:

:??[,*number*]

Where:

number is the error code number.

If *number* = blank–	Last error code issued is expanded.
If *number* = *xx*	The *xx* error code is expanded.
If *number* = 99	All error code messages are printed on the list file.

COMMENTS

- The ?? Command may be issued after any error number and will expand that message. If issued in response to the INitialize Command error 60, the message is printed and the INitialize Command is aborted.

EXit

Purpose:

To terminate the File Manager (FMGR).

Format:

:EX

COMMENTS

- After the exit message

  $END FMGR

  is printed on the log device, FMGR checks for the presence of a ready inspooled job. If one is found, it is executed.

OPERATOR COMMAND EXAMPLES

The operator commands presented in this section can be applied by two different methods. In the first method, the operator manually enters each command on the TTY to accomplish a given task. Many commands could be involved, requiring a great amount of manual labor to enter. The second method uses the TRansfer Command. The operator lists all commands required to do his task in program form (that is, serially) as shown in the following examples. These commands can then be punched on paper tape or cards (or typed directly into a file). A file can be created and these commands stored in it; control is then transferred to this file and the task is executed. Also, if desired, the commands can be entered through the paper tape reader or card reader by a TRansfer Command to the appropriate logical unit number.

The following examples describe in tutorial form one of the many different procedures that can be used to accomplish a task. Once the user becomes familiar with the HP File Manager and its vast powers, he can see how to alter these procedures in many different ways.

EXAMPLE NO. 1

This example shows a method of recording a set of programs on magnetic tape (hereafter called "mag tape") as relocatable binary in SIO format for use in generating future RTE Systems. As new or updated programs become available, the File Manager will be used to update and rewrite the mag tape.

Assume you will start with each of the relocatable binary programs stored on the disc as a separate file under control of the File Manager. For illustrative purposes, the files are named:

EXEC
SCHED
RTIOC
DVR00
DVR31
ASMB
LOADR
UPROG1
UPROG2

Now, plan to dump the disc files onto mag tape. Enter each file in turn, with due consideration of order of loading for later system generation (refer to the RTE Operating System manual). Assume the mag tape logical unit number is 8, and it is enabled.

```
:CN                    (Rewind)

:CN,,EOF               (Transfer the EOF file to
                        the mag tape)

:DU,EXEC,8,MSBR,IH     (Transfer first program to
                        mag tape)
```

The third parameter, MSBR, transfers binary relocatable records onto mag tape (LU8) in SIO format. The fourth parameter, IH, inhibits writing an end-of-file after EXEC This is necessary for subsequent use of the mag tape for RTE System generation.

```
:DU,SCHED,8,MSBR,IH
.
.
.
```

Enter the commands to transfer each program onto the mag tape, including the two user programs. Make one change to enter the last program. To provide an EOF at the tail end of the program sequence, omit the inhibit (IH) parameter.

```
:DU,UPROG2,8,MSBR
```

The mag tape is now complete and can be used in an RTE System generation. An extension of the process shown here could create a mag tape, holding a library of many programs, from which could be drawn a sub-set of programs selected for a particular RTE System, and easily duplicated for use on other copies of that system.

Now, assume that the time has come to update the mag tape. The third program of the sequence on the mag tape, which happens to be RTIOC, is to be replaced with an updated version. The process to be used is to separate the program set after the third program, saving the first three programs in a temporary file, dropping out the obsolete program, picking up the remaining programs, and storing them into another temporary file. A separate file is assumed to contain the new version of RTIOC. The various files will be recombined and transferred from the disc onto the mag tape as a complete, updated set.

There is a problem to be solved first. Remember that the programs were written onto the mag tape without any EOF separators between them. An easy way to replace these separators is to let FMGR insert EOF's (actually zero-length records). Remember that FMGR inserts these zero length records after each END record on transfers made from the load-and-go tracks. Turn on the FMGR (ON,FMGR) and enter the command to read the mag tape into the disc file TEMP1 (the assign load and go tracks) and from here, to the LG tracks. Note that the MR Command cannot be used directly because it does not accept SIO records.

```
:ST,8,TEMP1,MSBR    (Create the file TEMP1)
:LG,50              (Reserve load-and-go tracks)
:MR,TEMP1           (Transfer file to load-and-go
                     tracks)
```

Now, move the load-and-go tracks to a new file, TEMP2.

```
:SA,LG,TEMP2        (Create the file TEMP2)
```

In the Save process, FMGR inserted an "EOF" between each program. The programs are now separable. TEMP1 can be purged, if desired, as it is no longer needed

Transfer the first two programs of file TEMP2 into another temporary disc file, TEMP3.

```
:ST,TEMP2,TEMP3,BR,1,2   (Create the file TEMP3)
```

Starting with the first file, two files are stored in TEMP3.

The third program (RTIOC) is skipped, and the fourth through the last program is stored in TEMP4.

```
:ST,TEMP2,TEMP4,BR,4,10 (Create the file TEMP4)
```

The first three programs were skipped by setting the file number parameter to 4. TEMP4 now contains the remainder of the set, starting with the program DVR00. Refer to Figure 3-6. Note that the *SAve* parameter was not specified; therefore, TEMP3 and TEMP4 no longer contain the zero length records.

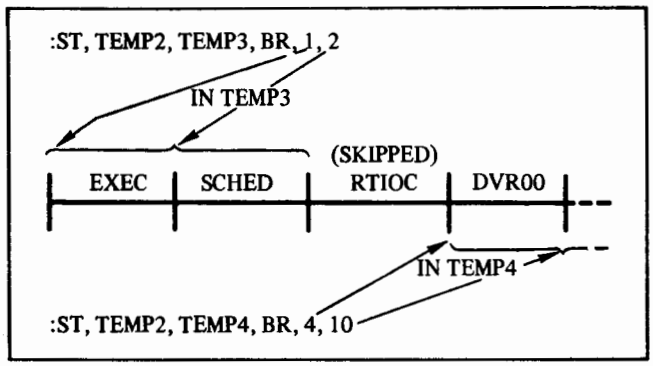


Figure 3-6. Example File Structure

The final parameter (10) is the number of files to be transferred and can be any number greater than the remaining set.

Now, combine the first section of the old mag tape; the new version of RTIOC; and the remainder of the old mag tape into one updated version. Assume the mag tape unit is enabled. Put an EOF at the head end, as before.

```
:CN

:CN,,EOF

:DU,TEMP3,8,MSBR,IH      (Dump TEMP3 to tape)
```

Add on the new RTIOC. Assume it is in a file named RTIOC.

```
DU,RTIOC,8,MSBR,IH       (Dump new RTIOC to
                          tape)
```

The inhibit (IH) is needed to prevent an EOF from being written after the files, since we intend to add more files. Now, add the remaining programs stored in TEMP4. Omit "IH" to have the final program end with an EOF.

```
:DU,TEMP4,8,MSBR         (Dump TEMP4 to tape)
```

The updated mag tape is ready. Purge all old files that are left over from the operations just performed.

EXAMPLE NO. 2

This second example relates the commands required to have the File Manager store, list, and edit a source program. Then compile, load, save, and later run the program without any intervening paper tapes.

Assume for purposes of illustration that you have written a program named "JEVA" in FORTRAN IV, and have prepared it as a punched paper tape. It is your ultimate aim to have the program JEVA stored in a file on the system disc. You plan to store the program as a load module so that it can be brought into core from time to time to execute in the background disc resident area. First, you wish to file temporary copies of the source program through various stages of editing. Second, you wish to keep a copy, in the form of a file, of the relocatable binary code that results from compilation.

The first action is to load the paper tape into a source program file (type 3) on the system disc. (Note that for simplicity, it is assumed that file security is not used on this system. Also, the system disc is assumed to be first in the disc directory, so that parameters will automatically default to the system disc.)

Place the paper tape in the photoreader, and enter the following commands at the system console. (It is assumed that all is in readiness for the system to operate.)

```
*ON,FMGR

:ST,5,JEVA1              (Create a file called
                          JEVA1)
```

The paper tape is read and transferred to a disc file JEVA1 created as a result of this command. Default values for parameters 3, 4, and 5 fit this case and simplify the entry of the command.

In preparation for editing the source program, use the FMGR LIst Command to obtain a listing complete with line numbers.

```
:LI,JEVA1                (List the file)
```

The listing is printed on the list device (LU6) since default parameters were used.

```
:RU, EDITR               (Run the Editor)

Source File?

/JEVA1

FTN, L
  .
  .
  .
edit JEVA1
  .
```

```
.
.
/ELR                    (Replace updated program leaving a copy in LS area)

:LG, 3                  (Assign load-and-go area)

:RU, FTN4, 2, 99        (Run compiler)
```

The system replies:

```
$END, FTN
```

Parameter "2" in the :RU, FTN4 Command directs the compiler to compile the contents of the LS tracks. Parameter "99" causes the results of the compilation, the relocatable binary program, to be transferred to the load-and-go tracks previously defined. The binary program should now be saved in a new file.

```
:SA, LG, JEVAR          (Create the file JEVAR)
```

Next, it is planned to load the program with the RTE Loader. After the loading operation, the resulting object code could be executed as a background disc resident program (refer to the RTE Operating System manual). However, in this case the object code will instead be stored as a load module with the Save Program Command, and later returned to core by the RUn Command. (Refer to the SP and RU Command).

Since the LG pointer is still directed to the program copy on the load-and-go tracks, that copy will be loaded. If the copy were no longer available, then the file JEVAR could be recalled to the LG tracks. To load the program, call for the RTE Loader:

```
:RU, LOADR,99           (Load the program)

/LOADR: JEVA READY      (NAM of program is JEVA)

/LOADR: END
```

The loader prints:

JEVA READY-LOADING COMPLETE

Note that the program name rather than the file name is reported by the loader in this case. Should the RP Command have been used to return a load module stored in a disc file to the load-and-go tracks, the first 5 characters of the file name would define the program name. To save the program, use the SP Command.

```
:SP,JEVA                (Create type 6 file called JEVA)
```

The program JEVA is now saved in the new file. Note that the file could be created on any of the discs under FMP control. However, when it is desired to return the load module to core, JEVA must be residing on LU2 or LU3.

Later to restart JEVA, the RU Command is used.

```
*ON,FMGR

:RU,JEVA                (Restore JEVA as a system program, run it, and after run, purge it from the system.)
```

The system assigns a blank ID segment (without tracks assigned if possible), set up to point to the file area.

The program is then run; and upon completion, it is removed from the system.

SECTION IV
FMGR ERROR CODES

FMGR AND FMP ERROR CODES

The error code structure is divided into positive and negative error codes. A negative error code is usually the result of an improper interface routine call. A positive error code is usually the result of an improper FMGR command. Since many of the FMGR operator commands use an interface routine (e.g., the SAve Command uses the CREAT Routine to create the *namr2* file), negative errors can be generated when using the FMGR operator commands. Due to the internal interaction of calls within the FMP, an improper command may cause FMGR to abort (returns control to RTE System), and leave files that are empty or otherwise incomplete on the system. An example of this sort of error is:

```
:ST,4,XYZ
```

Here 4 is assumed to be the punch. The command will create file XYZ, and then FMGR will be aborted when it tries to read from 4; XYZ will therefore be empty and corrupt (i.e., no end-of-file is contained in it). XYZ should be purged as soon as practicable. You must issue the ON,FMGR command to regain control and then purge the corrupt files.

An error will cause a transfer to the log device without aborting FMGR; except that if a job is active and the severity code is less than 3, the job will be aborted. This is an actual TR command generated by the FMP to notify you that an error has been made, and remedial action is required. From this point, input control is recognized only from the log device. As many commands as desired may be entered from the log device, followed by a TR command (no parameters) to return to the statement following the erroneous statement.

When packing with the PK Command, a non-lock error (-8) or a disc-not-found error (-6) will be followed by an additional message indicating which disc could not be locked or found. When the *label* parameter is not specified, the cartridge will be reported as a -LU. If *label* is specified, the report will be the cartridge reference number (**CR**).

EXAMPLES

```
:PK,30          (30 not previously defined)
FMGR -006
FMGR 030

:PK
FMGR -008       (not lockable)
FMGR -002       (LU2)
```

In the last example, LU2 will not be packed, because there is at least one open file on it; however, all other mounted cartridges will be packed.

Negative error codes are shown in Table 4-1. All routines (except STATUS) are listed across the top of the table, while errors are listed down the left side. This forms a matrix that shows which interface routines can be involved in an error. For example, error -8 can occur in the PURGE, OPEN, or RENAME routines.

All positive errors are listed in numerical order following Table 4-1 and are self explanatory, Note that all interface routine error codes are also returned in the computer's A-Register.

Table 4-1. Negative Error Codes

Error Code	Error Description	Creat	Purge	Open	Close	Read	Write	Locate	*Aposition	Rewind	Position	Rename	*Control	Post	IDCBS*
*≥0	None	X	X	X	X	X	X	X	X	X	X	X	X	X	X
−1	Disc down	X	X	X	X	X	X		X	X	X	X		X	
−2	Duplicate name	X										X			
−3	Backspace Not Legal (Type 0)										X				
−4	File too long or REC Size Error (Type 2)	X													
−5	Attempt to read or position to a record not written, or on update write an illegal record length.					X	X		X		X				
−6	Cartridge not found or file not found or no room.	X	O	X	X		X								
−7	Invalid security code		X	X			X					X			
−8	File currently open: eight PGM or exclusive or LOCK rejected		X	X								X			
−9	Attempt to open type 0 as type 1 or to use APOSN on type 0			X					X						
*−10	Not enough parameters	X	X	X	X	X	X	X	X		X	X			
*−11	DCB not open				X	X	X	X	X	X	X		X	X	X
−12	SOF or EOF read or sensed					X	X		X		X		X		
−13	Cartridge locked	X	X	X								X			
−14	Directory full	X					X								
−15	Illegal name	X										X			
−16	Illegal Type or Size = 0 (Creat)	X	X												
−17	Attempt to Read or Write on type 0 which does not support the operation.						X				X				
−101	Illegal parameter in D.RTR call. Possible operator error. Recheck previous entries for illegal or misplaced parameters.														
−102	Illegal D.RTR call sequence (lock not requested first or file not opened exclusively first). Possible operator error (e.g., physical removal of cartridge without issuing DC Command).														
	X – implies a TRANSFER to LOG device. O – implies no TRANSFER. * – implies error or function is never reported on the log device.														

Table 4-2. FMGR Error Messages

Error Number	Associated Routine	Meaning and Action	Additional Information
001	General	Disc Error	Logical unit.
002	System Initialize	Initialize LU2.	–
003	System Initialize	Initialize LU3.	–
004	System Initialize	Illegal response to 002 or 003.	–
005	System Initialize MS	Required track not available.	Relative TAT position.
006	MR,ST,DU	FMGR is suspending itself; ready device and GO	–
007	MR,SA,DU,ST	Checksum error.	–
008	System Initialize	D.RTR not found in ID segments. Load D.RTR.	–
009	RP	ID segment not found (no implied transfer).	
010	Parse Routine	Input error, re-enter statement. Caused by: Missing initial colon (non-TTY input). Supplied initial colon (TTY input). Command undefined. ASCII subparameter in subparameters 3 thru 5. Subparameters in other than first two parameters. more than 5 subparameters. Command too long.	Portion of line up to and including error is printed, followed by a "?."
	AB	No job active	
011	PK	Some system ID segments point to the disc to be packed. Do RP,,xxxxx, or OF ,xxxxx,8 on all named programs; Then re-enter PK Command.	List of program names.
012	MC	Duplicate logical unit or label.	–
013	TR	Transfer stack overflow.	–
014	MS,SP,RP	Program not found in system ID segments. Also, RP, no blank ID segments.	– –
015	MS	Logical Source track report follows (No implied transfer).	Track and LU of created LS file.
016	RP	The named file is not on logical unit 2 or 3, Move file to LU2 or 3 and re-issue.	–
017	RP	The named ID segment was not set up by FMGR– cannot be used or cleared.	–
018	RP	The named ID segment shows the program is not dormant; issue OF,XXXXX,1 and then re-issue RP.	–

Table 4-2. FMGR Error Messages (Continued)

Error Number	Associated Routine	Meaning and Action	Additional Information
019	RP	Checksum or setup code failed. File was not set up by an SProgram command on the current system.	–
020	CR	The given logical unit is illegal (creating a type 0 file).	–
021	CO	One of the specified discs is not mounted or both specifications refer to the same disc.	–
022	CO	The copy has been terminated. (NOTE: this is true of most copy errors regardless of this message being printed.)	–
023	RP	The given program name is already defined in a system ID segment.	–
024–046		Not defined.	–
047	JO, LU	Spool setup failed.	–
048	CA,SE, TR	Global set is out of range.	–
049	RU	Couldn't schedule a program setup from type 6 file.	–
050	IN,MR,SA,SP	Illegal number of parameters; usually too few.	
051	IN,DL	Illegal master security code.	–
052	IN,MC	Wrong logical unit. In response to 002 or 003, or no disc on given LU.	–
053	IN	Illegal disc label. Reference label must be positive non-zero integer. Information label must be a legal file name.	
054	IN,DC,PK,DL	Disc not mounted. Issue MCartridge command.	–
055	IN,MC,DC,CR, DU,ST	Missing parameter; re-enter command; could be a required parameter which is usually optional.	–
056	IN,DU,ST,CR, LO,SA,LI	Bad parameter. No file tracks (i.e., all directory or last track below first track). ASCII code undefined.	–
057	IN	Bad track error. Track not within file area or track is in directory area.	–
058	SA	Load and go area is undefined or empty.	–

Table 4-2. FMGR Error Messages (Continued)

Error Number	Associated Routine	Meaning and Action	Additional Information
059	IN	Track not available for re-INitialization (aborts Initialization).	Highest non-available track.
060	IN	INitialization will cause loss of all files on this disc. Do you really want to? Answer YES or NO. Caused by: a. First track is larger. b. Directory will extend into a file.	--

CHAPTER II
SPOOL MONITOR PROGRAM

SECTION V
SPOOL SYSTEM GENERAL DESCRIPTION

INTRODUCTION

The Spool Monitor (SM) is an optional software segment of the complete HP RTE Batch-Spool Monitor. (Refer to the Preface.) Unlike the File Management Package (FMP), the Spool Monitor cannot be used alone as a free-standing software system. However, when combined with the FMP, the SM provides the spooling capability for the Batch-Spool Monitor to accomplish batch processing operations in the spooling mode. For a further understanding of the important role that spooling plays in batch processing, refer to "Batch Utilization" in Section I.

The Spool Monitor (SM) provides the following major functions:

- It allows I/O to or from a disc file using standard EXEC I/O calls. Standard I/O calls here mean calls described for non-disc I/O transfers in the Real-Time Executive System Programming and Operating Manual for RTE-II or RTE-III.
- When a SPOOL is closed, SM controls its dump on an I/O device available to standard I/O calls.
- As an extension to the File Management Package, it modifies two files used by the spool system, JOBFIL and SPLCON, to reflect the current states of jobs and spools in the system.

The SM can do any or all of the above for a configuration-limited number of files on a time-sharing basis; that is, several files may be active in the Spool system at any given time.

The important fact to remember is that the inspooling processing, and outspooling operations take place independently and simultaneously.The major components of the Spool Monitor system and the flow of job input through the system are shown in Figure 5-1, and Table 5-1.

The program JOB provides a spool-in capability for batch jobs. JOB is described in further detail in Section VII, Inspooling.

The program GASP provides set up or initialization capability in addition to providing an operator interface to allow monitoring and/or changing of the status of spools and/or jobs on the system.

When spooling, the File Manager (FMGR) is directed to a job input file by the contents of a job control disc file named JOBFIL. This file is set up by JOB and will be modified by FMGR to reflect the current state of the job. The contents of this file are described in detail in Appendix 1. The FMGR reads the first command from the Job file and, if acceptable, sets up a list spool and an input spool for the job. FMGR then reads the job file performing the indicated operations and finally terminating the job. At this time, it returns to the JOBFIL for further direction. FMGR continues to run until there is nothing left in JOBFIL.

The program JOB runs independently of FMGR and accepts input files and commands which it uses to build entries in the file JOBFIL, which causes FMGR to execute the given job.

The program SMP also runs independently maintaining information in a file called SPLCON on all active spools in the system (refer to Appendix J for SPLCON organization). In addition it monitors the activities of the outspooling program (SPOUT).

SPOUT runs independently and continuously being first scheduled by the SMP. It dumps outspool files to their respective devices and can handle several at a time.

The SM will, as an option, do batch input checking. This consists of special processing of records containing a colon in column one, so that they are not available to any program but the File Manager (FMGR).

The SM also aborts spool-out functions to allow for abnormal job termination.

The Spool system transfers to or from the disc at a maximum rate of approximately 5K words/second. This rate is well above most other I/O devices and thus the limit is usually imposed by those devices.

The spooling of the line printer decreases its idle time and thus relieves the limits previously imposed by printer speed. Maximum gain in throughput is the percentage of printer idle time without the Spool system.

The Spool system will, in general, contend with other users for use of the disc, thus forcing these other users to run slower. This effect is minimized by:

a. Using a separate disc drive for spooling.
b. Shutting down the spool system during critical periods.
c. Not using spooling.

SOFTWARE ENVIRONMENT

The Spool Monitor (SM) is an optional extension to the File Management Package to provide spooling capability. When the Spool Monitor is combined with the FMP, it requires the same software environment as the FMP does alone, plus the EQT Extension and Resource Number (RN) features of RTE as well as Class I/O and logical unit (LU) lock.

The SM uses some of the FMP's subroutines and such library routines as they call. In addition, the following subroutines are provided to allow the user convenient access to the Spool Monitor for OPEN and CLOSE spool operation:

SPOPN	(Spool Open)
SPCLS	(Spool Close)

Batch operating procedures for jobs run with or without spooling are described in Appendix L.

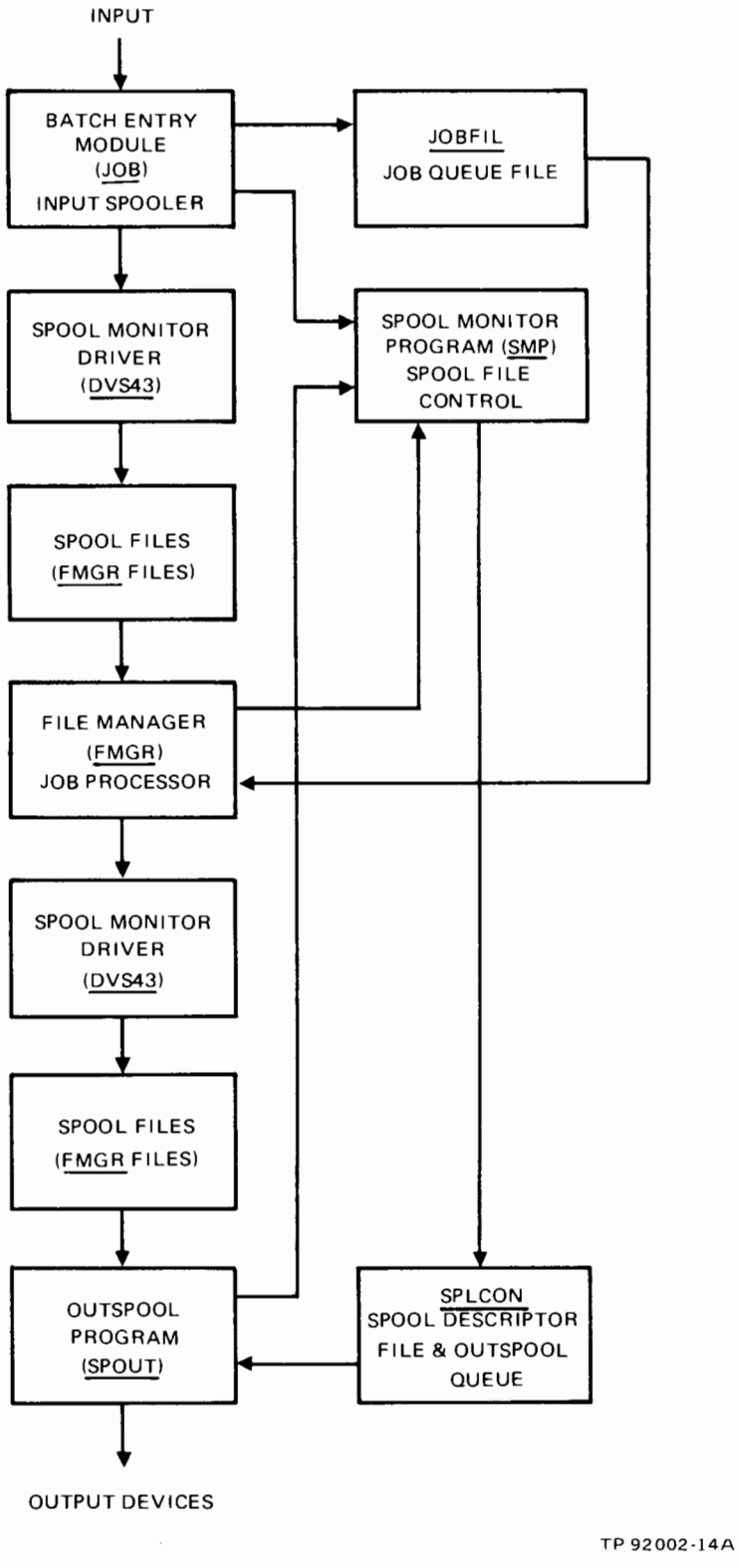


Figure 5-1. Spool Monitor Flow Diagram

Table 5-1. Event Chart

INSPOOLING (*JOB*)	PROCESSING (*FMGR*)	OUTSPOOLING (*SPOUT*)
JOB IS STARTED: READS INPUT . . . *JOB* FINISHES READING A JOB: UPDATES *JOBFIL* QUEUE, SCHEDULES *FMGR'*		
JOB STARTS READING NEXT JOB . . . *JOB* FINISHES READING A JOB. . . . . *JOB* CONTINUES OPERATING UNTIL THERE IS NO MORE INPUT.	*FMGR* STARTS – LOOKS AT *JOBFIL* QUEUE AND FINDS JOB TO PROCESS – SETS UP READ SPOOL. SETS WRITE SPOOLS. *FMGR* CONTINUES TO PROCESS THIS JOB. . . . . *FMGR* FINISHES JOB – CLOSES SPOOL FILES AND GETS NEXT JOB. *FMGR* CONTINUES PROCESSING UNTIL THERE ARE NO MORE JOBS LEFT IN THE *JOBFIL* QUEUE.	*SPOUT* SCHEDULED WHEN FIRST OUTSPOOL IS SET UP. STARTS DUMPING TO DEVICES AS SOON AS A FILE IS QUEUED AND OUTPUT EXISTS.

HARDWARE ENVIRONMENT

The Spool Monitor (SM) requires no additional hardware beyond the minimum system requirement for the File Management Package (FMP), except for the additional memory space. Refer to Hardware Environment, Section I.

SPOOLING PHILOSOPHY

The "spooling" of inputs and outputs onto disc files is an effective way of extending output buffering capacity. Input spooling of the job stream onto the disc increases the number and length of jobs that can be backlogged for processing with a given amount of time.

Spooling of output to the disc proceeds at a maximum speed of about 10,000 bytes per second, instead of at the much slower speed of the peripheral device. Thus, the rate of throughput is increased, because the executing job does not have to wait for data to be output to a line printer, plotter, or terminal. With the disc storage being supervised by the RTE Batch-Spool Monitor, all of the peripheral devices can be kept busy most of the time, particularly during periods of heavy workload.

The use of spooling for batch processing operations is optional. It is set up when the Great Automatic Spool Program (GASP) is scheduled for the first time after RTE system generation. Thereafter, it is controlled automatically by jobstream commands through RTE. After setup, the operator can turn on GASP to issue the monitoring and spooling control commands described in Section VIII, Spool Commands.

SPOOLING TECHNICAL DISCUSSION

ORGANIZATION

The Spool Monitor (the Spool portion of the HP RTE Batch-Spool Monitor) includes the following programs:

Name	Function
Batch Entry Module (JOB)	Job input monitor.
Great Automatic Spool Program (GASP)	Operator interface to the spool system.
Spool Monitor Driver (DVS43)	Handles I/O requests destined for spool files.
Spool Monitor Program (SMP)	Handles spool file manipulation.
Outspool Program (SPOUT)	Dumps outspool files onto the destination devices.
Extend Program (EXTND)	Gets file extensions for DVS43.

Two of the program parts (SMP and SPOUT) call each other to effect spooling. EXTND is called by DVS43 to get file extensions. The spool file must first be set up, after which standard I/O operations may be done on it. (Refer to Figure 5-1 to see the interrelationship of these parts of SM.)

The Batch Entry Module (JOB) is described in Section VII. Inspooling and the Great Automatic Spool Program (GASP) are described in Section VIII, Spool Commands.

SETUP OF A SPOOL–A Spool is set up by a call to SMP. SMP analyzes the setup parameters and establishes the spool. As part of the setup, an LU number is assigned and passed back to the caller.

SPOOL I/O–Spool I/O is effected with standard EXEC I/O calls directed to the LU assigned during Spool setup.

SPOOL CLOSE–The spool is closed with a call to SMP directed to the assigned LU, SMP calls DVS43 to post any pending data. Then SMP performs the functions indicated at setup time, among which is passage of the file to the outspool program for outspooling. Any file extents are deleted at close or, if file is outspooled, when outspooling is completed.

OUTSPOOLING–Outspooling consists of reading from a file and writing on an LU, such as the line printer. The Outspool Program (SPOUT) will outspool either immediately or at close, or not at all, depending on how the spool is set up.

RECORD FORMATS OF SPOOL FILES

Two different formats are written by the SM on output. The choice of one or the other is dependent on the destination of the file. In both cases this format is imbedded in the standard type 3 file format (see Section I).

OUTSPOOL FORMAT–Each record in these files is preceded by a two-word header as follows:

Word	Contents	Comments
0	Control Word	Contains function and subfunction used in original I/O call.
1	Length/Extra	Length in + words or – characters or if a control request, then the extra control word passed to DVS43 by the system.

NOTE

The control word is from the EQT and not from the user request; thus it does not contain the original LU.

The rest of the record is the data to be output. The standard list output spool is set up automatically for a spooled job and is written in outspool format. This is necessary in order to preserve form feed and spacing information for the list device (usually a line printer).

STANDARD FORMAT—These files are written as presented, with no header. This format can be used for some devices, such as the punch, or for files not to be outspooled to a device.

SECTION VI
SPOOL CALLS

INTRODUCTION

This section describes the basic format of HP Assembly Language calls and FORTRAN calls to the various subroutines that interface the user program to the Spool Monitor (SM). SM uses some of the File Management Package (FMP) subroutines and such library routines as they call. In addition, a Spool Monitor subroutine, SPOPN, allows the user convenient access to SM for the Open Spool operation.

The Spool Calls are classed in the following categories:

- SMP Calls
- Spool I/O Calls
- Outspooling Calls

SMP CALLS

Spool control is accomplished by scheduling the Spool Monitor Program (SMP) with a particular request. The various requests, which are used largely by the FMP, are as follows:

- Setup (or Open request)
- Change Purge to Save
- Change Save to Purge
- Pass Now
- Close the Spool and Pass
- Modify Pass Information
- Set Buffering Flag
- Clear Buffering Flag
- Get Current Disc Position in Spool File
- Change Starting Record Position

Setup (or Open Request)

Purpose:

This procedure makes a spool file active and ready for use by the SMP.

Assembly Language Call:

```
                EXT     EXEC
                 .
                 .
                 .
                JSB     EXEC
                DEF     RTN
                DEF     D20     class write/read request
                DEF     D0      on logical unit Ø
                DEF     IBUFR   set up buffer
                DEF     IBUFL
                DEF     D0
                DEF     D0
                DEF     ICLAS
RTN1            return point
                 .
                 .
                 .
                JSB     EXEC
                DEF     RTN2
                DEF     D23     schedules the SMP
                DEF     ISMP
                DEF     D0
                DEF     ICLAS   uses class setup from previous call
                DEF     IBCHK
RTN2            return point
                 .
                 .
                 .
D18             DEC     18
D0              Dec     0
IBUFR                           16-word buffer (see Comments)
IBUFL           DEC     16
D23             DEC     23
ISMP            ASC     3,SMP
ICLAS           DEC     Ø       class number returns here
```

FORTRAN CALL:

```
CALL EXEC (20,0,IBUFR,IBUFL,0,0,ICLAS)

CALL EXEC (23,ISMP,0,ICLAS,IBCHK)
```

Where:

ISMP	is the 3-word program name, SMP, used for the schedule call.
IBCHK	is batch checking information which equals the ID segment address of the caller if batch checking is desired or 0 otherwise.

COMMENTS

- SMP will allocate a record for the given spool in a file called SPLCON (see Appendix J) which describes the spool files currently active or queued for outspooling. An EQT entry will be initialized, and its corresponding LU number will be assigned to the spool file. SMP passes back the LU number if successful.

- The buffer IBUFR that is ultimately passed to SMP by means of the class WRITE/READ call should be formatted as follows:

0	0 if no batch input checking desired. ≠0 if batch checking desired.
1	not set by user – spool LU storage
2,3,4	file name
5	security code (file identifiers)
6	cartridge ID (file identifiers)
7	driver type (file identifiers)
8	disposition flags bit 0: set → save, clear → purge bit 1: hold the file until close bit 3: set → spool pool file bit 4: set → standard file, clear implies record headers. bits 8 and 7: 00 → write and read 01 → read only 10 → write only bit 14: set → batch input sign bit (15): set → buffering
9	spool priority
10	spool status (used by SM and GASP)
11	job # or 0 if not applicable
12,13,14	0
15	outspool LU#

- SMP passes back one parameter which contains the LU# if successful, 0 if unsuccessful. A call to RMPAR (or LDA B,I) will retrieve this value.

NOTE

The SPOPN subroutine may be used instead of this procedure to effect this call.

Change Purge to Save

Purpose:

This procedure causes the file to be saved at the end of operations.

Assembly Language Call:

```
                EXT     EXEC
                 .
                 .
                 .
                JSB     EXEC
                DEF     RTN
                DEF     D23
                DEF     ISMPA
                DEF     D1
                DEF     LU
RTN             return point
                 .
                 .
D23             DEC     23      system schedule code
ISMPA           ASC     3, SMP  SMP name
D1              DEC     1       request code for this procedure
LU              BSS     1       must be set to spool LU assigned by Setup request
```

FORTRAN Call:

```
CALL EXEC (23,ISMPA,1,LU#)
```

COMMENTS

- This request is not honored if the file is a spool pool file.

Change Save to Purge

Purpose:

This procedure causes the file to be purged at the end of operations.

Assembly Language Call:

```
                EXT     EXEC
                 .
                 .
                 .
                JSB     EXEC
                DEF     RTN
                DEF     D23
                DEF     ISMPA
                DEF     D2
                DEF     LU
RTN             return point
                 .
                 .
                 .
D23             DEC     23      system schedule code
ISMPA           ASC     3,SMP   SMP name
D2              DEC     2       request code for this procedure
LU              BSS     1       must be set to spool LU assigned by Setup request
                                (Open)
```

FORTRAN Call:

```
CALL EXEC (23,ISMPA,2,LU)
```

COMMENTS

- In the case of a spool pool file, the file will be returned to the available pool, but will not be purged from the disc.

Pass Now

Purpose:

This procedure causes the file to be queued for outspooling (removes a hold) if the file is to be outspooled.

Assembly Language Call:

```
                EXT     EXEC
                 .
                 .
                 .
                JSB     EXEC
                DEF     RTN
                DEF     D23
                DEF     ISMPA
                DEF     D3
                DEF     LU
RTN             return point
                 .
                 .
D23             DEC     23      system schedule code
ISMPA           ASC     3,SMP   SMP name
D3              DEC     3       request code for this procedure
LU              BSS     1       must be set to spool LU assigned by Setup request
```

FORTRAN Call:

```
CALL EXEC (23,ISMPA,3,LU)
```

COMMENTS

- If the file has already been queued, this call is a NOP.

Close the Spool and Pass

Purpose:

This procedure causes an EOF to be written on the file and the file to be queued for outspooling if not already on the outspool queue or in process of outspooling.

Assembly Language Call:


```
            EXT     EXEC
             .
             .
             .
            JSB     EXEC
            DEF     RTN
            DEF     D23
            DEF     ISMPA
            DEF     D4
            DEF     LU
RTN         return point
             .
             .
D23         DEC     23      system schedule code
ISMPA       ASC     3,SMP   SMP name
D4          DEC     4       request code for this procedure
LU          BSS     1       must be set to spool LU assigned by Setup request
```

FORTRAN Call:

```
CALL EXEC (23,ISMPA,4,LU)
```

Modify Pass Information

Purpose:

This procedure changes outspool LU or parameter or outspool priority.

Assembly Language Call:

```
            EXT     EXEC
             .
             .
             .
            JSB     EXEC
            DEF     RTN
            DEF     D23
            DEF     ISMPA
            DEF     D5
            DEF     LU
            DEF     NOL
            DEF     NPR
RTN         return point
             .
             .
D23         DEC     23          system schedule code
ISMPA       ASC     3,SMP       SMP name
D5          DEC     5           request code for this procedure
LU          BSS     1           must be set to spool LU assigned by Setup request
NPR         DEC     new spool priority
NOL         DEC     new outspool logical unit
```

FORTRAN Call:

```
CALL EXEC (23,ISMPA,5,LU,NOL,NPR)
```

COMMENTS

- This call will have no effect on files currently being dumped by SPOUT.

Set Buffer Flag

Purpose:

This procedure sets the buffering flag in the spool EQT.

Assembly Language Call:

```
                EXT     EXEC
                 .
                 .
                 .
                JSB     EXEC
                DEF     RTN
                DEF     D23
                DEF     ISMPA
                DEF     D6
                DEF     LU
RTN             return point
                 .
                 .
D23             DEC     23      system schedule code
ISMPA           ASC     3,SMP   SMP name
D6              DEC     6       request code for this procedure
LU              BSS     1       must be set to spool LU assigned by Setup request
```

FORTRAN Call:

```
CALL EXEC (23,ISMPA,6,LU)
```

COMMENTS

- Ordinarily, buffering is not recommended.

Clear Buffer Flag

Purpose:

This procedure clears the buffering flag in the spool EQT.

Assembly Language Call:

```
                EXT     EXEC
                 .
                 .
                 .
                JSB     EXEC
                DEF     RTN
                DEF     D23
                DEF     ISMPA
                DEF     D7
                DEF     LU
RTN             return point
                 .
                 .
D23             DEC     23      system schedule code
ISMPA           ASC     3,SMP   SMP name
D7              DEC     7       request code for this procedure
LU              BSS     1       must be set to spool LU assigned by Setup request
```

FORTRAN Call:

```
CALL EXEC (23,ISMPA,7,LU)
```

Get Current Disc Position in Spool File

Purpose:

This procedure retrieves position information – pointers to the next record that is to be read (or written).

Assembly Language Call:

```
                EXT     EXEC
                 .
                 .
                 .
                JSB     EXEC
                DEF     RTN
                DEF     D23
                DEF     ISMPA
                DEF     D8
                DEF     LU
RTN             return point
                 .
                 .
D23             DEC     23      system schedule code
ISMPA           ASC     3,SMP   SMP name
D8              DEC     8       request code for this procedure
LU              BSS     1       must be set to spool LU assigned by Setup request
```

FORTRAN Call:

```
CALL EXEC (23,ISMPA,8,LU)
```

COMMENTS

A call to RMPAR (CALL RMPAR(PRAM)) upon return from this EXEC call must be performed in order to retrieve the information on current disc position contained in the first three words of PRAM, as follows:

word 1: track/sector
word 2: sector offset
word 3: extent number

This information is needed for parameters ISEC, IOFST, and IEXT in the next spool call, "Change Starting Record Position".

Change Starting Record Position

Purpose:

This procedure changes the position information which indicates where the next access is to start.

Assembly Language Call:

```
                EXT     EXEC
                 .
                 .
                 .
                JSB     EXEC
                DEF     RTN
                DEF     D23
                DEF     ISMPA
                DEF     D9
                DEF     LU
                DEF     ISEC
                DEF     IOFST
                DEF     IEXT
RTN             return point
                 .
                 .
D23             DEC     23      system schedule code
ISMPA           ASC     3,SMP   SMP name
D9              DEC     9       request code for this procedure
LU              BSS     1       must be set to spool LU assigned by Setup request
ISEC            BSS     1       track/sector
IOFST           BSS     1       sector offset
IEXT            BSS     1       extent number
```

FORTRAN Call:

```
CALL EXEC (23,ISMPA,9,LU,ISEC,IOFST,IEXT)
```

Where:

ISEC, IOFST, and IEXT are set to the values passed in the previous Spool Control call, "Get Current Disc Position in Spool File."

SPOOL I/O CALLS

I/O REQUESTS TO THE SPOOL MONITOR DRIVER

After a spool has been set up, I/O EXEC requests may be issued for the assigned LU. These are processed by the Spool Monitor Driver (DVS43) which does any necessary file handling.

CONTROL REQUESTS TO DVS43

The following control requests may be issued for a spool LU which is being written in standard format. (Refer to Section V under "Record Formats of Spool Files.") They will be ignored in the case of files containing headers. Other control requests (such as TOFORM) to files containing headers will be posted to the file and interpreted later by the Outspool Program (SPOUT).

Control Function	Action
01	Writes EOF
02	Backspace 1 record
03	Forward space 1 record
04	Rewind (set to start of file)
05	Same as 04
06–13	NOP
14	Same as 04

QUEUEING FOR OUTSPOOLING

PASSING TO THE OUTSPOOL

When a file is ready for the Outspool Program (SPOUT), its SPLCON record will be put on an outspool queue dependent upon its destination LU. The Outspool Program will be scheduled, if it is not already scheduled.

OUTSPOOLING BY THE SM OUTSPOOL PROGRAM (SPOUT)

SPOUT is capable of spooling to several devices at a time and will receive information on files to be outspooled from SMP which gets its information from the file SPLCON. The SMP will modify status and queue information in SPLCON as necessary. Once scheduled, SPOUT runs continuously, receiving information on files to be processed by means of class requests from SMP. When SPOUT is not busy, it will be suspended on its class GET request.

SPOOL OVERFLOW CONSIDERATIONS

Spool files will be extended automatically as much as is possible. If an overflow occurs, the following actions will be taken by the DVS43:

1. The last line to be written which is causing the overflow will be ignored and an EOF written at the end of the file.

2. All future write requests to that spool will cause an IO07 error.

3. If the Input Spooler (JOB) detects an overflow while writing a file, JOB will be aborted with an IO07. JOB will re-use the affected spool file next time it is run.

4. An outspool overflow occuring during operation of the Batch Monitor causes the break flag to be set upon detection of the EOF error. The FMGR will abort the current job.

5. An outspool overflow occuring during execution of a program scheduled by FMGR will, in general, cause that program to be aborted (by the IO07 error) which will in turn cause the current job to be aborted.

EOF CONDITIONS

On read, an end of file will cause a zero-length transmission log with the EOF bit set in the status. Reads beyond the EOF will cause the program to be aborted (IO07 error).

BATCH INPUT CHECKING

An attempt to read an image with a colon (:) in column one from a batch input file by a program other than the FMP will be treated as an EOF.

SPOOL OPEN SUBROUTINES

The Spool Monitor also has a callable subroutine which will accomplish the same OPEN function described under "Setup (or Open Request)".

SPOPN

Purpose:

This routine allows user access to the Spool Monitor for OPEN spool operation.

Assembly Language Call:

```
                EXT     SPOPN
                 .
                 .
                 .
                JSB     SPOPN       Subroutine call.
                DEF     RTN         Return address.
                DEF     IBUFR       Setup buffer
                DEF     LU          Returned logical unit number of spool
                                    from SMP
        RTN     return point        Continue
                                    execution.
```

FORTRAN Call:

```
        CALL SPOPN (IBUFR,LU)
```

COMMENTS

- This routine makes the call to SMP and the class WRITE/READ to pass the buffer.
- The buffer must be set up as described for spool setup (or open) request.

SECTION VII
INSPOOLING

INTRODUCTION

The Input Spooler (JOB) is used to spool jobs into the system for batch processing. This process is accomplished by using the disc file JOBFIL which is described in Section V of Chapter II, Spool Monitor Program. JOB will obtain a spool file and read the job from the input device into the spool file (using the Spool Monitor). It will then make an entry in JOBFIL for the job and schedule FMGR if it is not already scheduled. FMGR searches JOBFIL for jobs to process. (Refer to Figure 5-1.)

The Input Spooler is started by the system RU Command as follows:

RU,JOB[, *in*]

where:

in is the input unit for JOB. Default is 5.

– or –

RU,JOB,*FI,LE,NA,pr*

where:

FI LE NA is the file name of a job file; pairs of integers must not be used in this format.

pr is the job file's priority.

NOTE

This form is the same as entering a :XE,*namr*, *pr* command

OPERATIONAL ASPECTS

When JOB is run to enter a batch job, the input device is read until one of the following JOB control cards is encountered.

- A :JOb card is read (see Section III, FMGR Commands, for JOb card syntax). This will cause a JOBFIL entry to be made for the JOB with status showing "READ IN", a spool file to be allocated from the spool file pool, and the remainder of the job to be read and written into the allocated spool file. The deck must be terminated with an :EOj card. Detection of an :EOj card or a JOB control card (:JOb card or :XEq card) prior to an :EOj card will cause the spool file to be closed and JOBFIL to be updated to show the job as waiting for execution. If FMGR is dormant, it will be scheduled. JOB will then act on the JOB control card, if any, or read the next card in the reader, if any. If there is no further input, JOB will terminate.

- An execute card with syntax:

:XEq,*namr,pr*

where:

namr is as defined in Section III

pr is job priority

is read. This will cause the named file to be entered as a job in JOBFIL with the given priority.

- An EOF or hopper-empty condition prior to reading a legal JOB control card, or after an :EOj or XEq card, and prior to a new legal JOb control card, will cause termination of JOB.

- An EOF encountered in a job deck will be entered as a zero-length record. An EOT or hopper-empty will be entered as an :EOj command.

In a batch system, JOB may run in the foreground disc resident area, thus interfering as little as possible with the FMP. If other use is made of the foreground, it may be desirable to make the JOB core resident.

If JOB is turned on with an interactive input device, it prompts with a semicolon(;). The operator enters each job card, including any required colons(:). Input may be terminated before the first :JOb card or after either a :EOj card or a :XEq card by entering Control D (D^C). A Control D (D^C) at any other time will enter a zero-length record in the job spool file. (See Example 2 under "Examples of JOB Usage".)

The following conditions will cause JOB to suspend until the needed resource is available:

a. An empty job control record in JOBFIL does not exist.

b. There are less than two free spool pool files, and one is required for this job's spool-in (JOB will not use the last spool file.)

c. There is not an EQT available for inspooling. (One of the EQTs assigned to DVS43 must be available to handle all but the :XEq job.)

The following error messages may be generated by JOB:

JOB WAIT ON PT

This message means that JOB was reading a job from the paper tape reader and detected EOT between the JOB card and the EOj card. Ready the reader with the rest of the job and type *GO,JOB.

END JOB ABNORM

This message is caused by one of the following:

a. Not being able to open the file JOBFIL.

b. An FMP write error while writing on JOBFIL.

c. An FMP read error while reading JOBFIL.

IO07 – JOB

If the spool file being written by JOB overflows available disc space, the spool driver will abort JOB with an IO07 error.

EXAMPLES OF JOB USAGE

EXAMPLE 1:

The following deck might be entered to do a FORTRAN Load-and-Go:

```
:JOB,TEXT
:TR,FORTLG
FTN,L          }
  –            }
  –            }
  –            }  FORTRAN Source Deck
  –            }
  –            }
END            }
END$
1.0            }
  –            }
  –            }  Data
  –            }
  –            }
:EO            }
```

This job is entered from LU5 by:

RU,JOB

If more jobs are to be batched, they may be put after :EOj. JOB need be turned on only once in this case.

NOTE

When a job is entered from a second terminal under control of the Multi-Terminal Monitor, the input logical unit must be specified, for example:

RU,JOB,5

EXAMPLE 2:

This example illustrates how to enter a job file from a TTY (assumed to be LU1). Operator input is shaded.

```
* RU,JOB,1
; :JOB,EXAMII                    Job card
; :ST,5,ASMBF                    Store following file
; :MS,1G                   }     File ASMBF to
; :RU,ASMB,1G              }       be created
; D^C        (control D)         End of ASMBF
; :TR,ASMBF,INFIL                Execute ASMBF; store INFIL
                                    in global 1G
; :EO                            End of job
; D^C        (control D)         End JOB
```

SECTION VIII
SPOOL COMMANDS

INTRODUCTION

The spooling function is controlled through the Great Automatic Spool Program (GASP). GASP is a background program that provides the operator a means to examine and modify the status of ongoing jobs and/or spools. GASP also initializes the JOBFIL and the Spool Control files.

COMMAND STRUCTURE

The operator commands used in the spooling function are structured much the same as the equivalent FMGR commands in the File Management Package. (Refer to Section III, FMGR Commands.) Command syntax for the spooling commands is the same as that outlined in Table 3-2, except that *namr* is not used.

SPOOL FILE CONVENTIONS

The user is responsible for creation of spool pool files for input and output spooling. Creation of these files is accomplished by use of the interactive program, GASP, the first time the user schedules it.

These user-created files have the following characteristics:

- The names created are of the form:

  SPOL01
  SPOL02
  .
  .
  .
  SPOL80

- Up to 80 such files may be defined, however, the user need only define as many as he actually needs.
- All files are the same size.
- All files are type 3.

The FMGR, SM, and JOB programs manage the use of these files for spooling purposes. They are reused and extents deleted to prevent packing problems.

RUNNING GASP

The Great Automatic Spool Program (GASP) is run by using the RU,GASP command. First, the operator must "get the system's attention" by striking any key on the system console keyboard. Then the RU,GASP command. together with its parameter (as shown in the following command format display), is typed in by the operator. GASP responds with an upward-pointing arrow (↑) prompt character and accepts any of the commands listed in Table 8-1. On some terminals, such as the HP 2640, the up-arrow prompt is printed as a caret (^).

RU,GASP

Purpose:

To initiate GASP.

Format:

RU,GASP [,*lu*]

Where:

lu	is the TTY logical unit the operator wishes to use. The default is the system TTY logical unit 1. No default *lu* is provided when command is entered from a second terminal under control of Multi-Terminal Monitor.

Table 8-1. GASP Operator Commands

AB

Purpose:

To abort a job before it runs.

Format:

↑AB, *job numb*

Where:

job numb is the number of the job to be aborted.

COMMENTS

- This command may only be issued for jobs in "RH" or "R" status.
- This command sets the job's status to active ("A"), which causes FMGR to remove it from the job queue the next time it looks for a job.
- If FMGR is dormant, it is scheduled to check the job queue.

CJ

Purpose:

To change job status.

Format

↑ CJ, *job numb*, *job priority* / *H* / *R*

Where:

job numb is the job number.

job priority is the new job priority.

H indicates the job is to be held.

R indicates that a held job is to be released.

COMMENTS

It is illegal to use this command to alter a job which is active or which is completed and waiting for outspooling to finish.

CS

Purpose:

To change spool status.

Format:

↑ CS, *name*, *sp priority* / *H* / *R*

Where:

name is the spool file name.

sp priority indicates a new spool priority.

R indicates release a held file.

H indicates Hold.

COMMENTS

- Holding an active file will cause a pause in I/O to that device. This pause ties up the device until the hold is released or the file is KIlled or ReSpooled. This occurrence will be indicated by the status becoming "AH" instead of "H", refer to "Outspool File States" later in this section.

- It is illegal to change the priority of an active spool file.

DA

Purpose:

To deallocate all spool files and spool control files.

Format:

```
↑DA
KILL SPOOLING? YES
```

Where:

KILL SPOOLING?	is printed out by the system after the operator has entered DA.
YES	is the reply that must be entered by the operator before the DA Command will execute.

COMMENTS

- Before using DA, SD command should be used to shut down spooling.

- This command effectively removes spooling from the system. Completion of this command is indicated by the message:

```
SPOOL IS DEAD!
END GASP
```

- If any file is open, it indicates that the spool system is still active. GASP reports the file name and the purge error code and its meaning, and then aborts the command. The operator may then enter either an SD Command or a KS Command, as needed, or re-issue the DA Command.

- This command must not be issued if GASP was scheduled by FMGR (with the :RU,GASP instruction), as it leaves JOBFIL open.

DJ

Purpose:

To display job status.

Format:

↑ DJ [, *job numb* | *name*]

Where:

job numb	is the job number.
name	is the job name.

COMMENTS

- This command displays the outspool status of job number, or of all jobs named *name*, or of all jobs if no parameter is given. The display is formatted as follows:

##	NAME	STATUS	SPOOLS
1	ABCD	STATUS	SPOOL NOS.

Where the first two entries are the job's number and name and the last entry is a list of spool pool files it is using (i.e., a list of two-digit file numbers). The status field will be coded as follows:

SPPPP NN

Where:

S	indicates the job source as "S" for a file source and "D" for a direct Logical Unit source.
PPPP	is the four-digit job priority.
NN	is a one- or two-letter code indicating the job status as:

I	In process of receiving input from JOB.
RH	Ready for running but in hold.
R	Ready for running.
A	Active (i.e., running).
CS	Job has completed and is waiting for outspooling.

- If the spool system (or batch) is shut down, a message to that effect is printed after the status messages.

DS

Purpose:

To display spool status.

Format:

↑ DS [,*lu*]

Where:

lu is the logical unit whose spool status is to be displayed.

COMMENTS

- This command displays the status of all current spools or the outspool queue for *lu*, if given, as follows:

lu spoolfile priority jobnumb status

Where:

lu is the logical unit number, or it is "- -" if the file is not an outspool file.

Spoolfile is the name of the file being spooled.

priority is its priority.

jobnumb if present, is a number indicating that the Spool is for the indicated job.

status is an ASCII character indicating the status as follows:

A	file is being outspooled.
W	file is waiting for the device.
H	file is being held (by operator).
AH	file is active and in hold status.
--	file is not an outspool file.

- If the spool system (or just outspooling) is shut down, a message to that effect is printed after the status messages.

EX

Purpose:

To terminate GASP.

Format:

↑ EX

KS

Purpose:

To purge a Spool from the output queue.

Format:

↑ KS, *number* / *name*

Where:

number indicates the logical unit on which an ongoing outspool is to be aborted.

name indicates the Spool file to be purged from the queue and/or aborted.

COMMENTS

- Killing a file that is being outspooled causes the I/O to that device to be terminated; and, if any other files are queued for the device one of them will be started.

- It is illegal to kill an active file whose job is also still active. The job must be aborted first by means of a system abort command (*AB); then the spool file may be killed.

- Caution should be exercised when using the KS, *number*, as it is possible to abort the wrong file. This error is especially likely if the current file is nearly finished.

RS

Purpose:

To restart an outspool from the beginning.

Format:

↑ RS, *name* [,*lu*]

Where:

name is the Spool name.

lu is an optional new logical unit to use for the outspool.

COMMENTS

- If the file is active (currently being dumped to a device), it will be restarted from the beginning of the file. If the file is waiting for outspooling, the only effect will be a possible logical unit change.
- Files that have finished outspooling may not be respooled. Refer to "Outspool File States" at the end of this section for use of RS to change spool file status.

SD

Purpose:

To shut down the Batch-Spool Monitor.

Format:

↑ SD, $\left[\begin{matrix} B \\ S \end{matrix}\right]$

Where:

B indicates that a hold is to be put on all pending jobs.

S indicates that a hold is to be put on all pending outspools.

COMMENTS

- If *B* or *S* is not given, a hold is put on both jobs and spools.
- Active jobs and spools are allowed to run to completion.
- Jobs may be submitted even though the system is shut down.

SU

Purpose

To start up a system that has been shut down.

Format:

↑ SU, $\left[\begin{matrix} B \\ S \end{matrix}\right]$

Where:

B indicates that batch jobs shut down with the SD Command are to be restarted.

S indicates that Spools held by the SD Command are to be restarted.

COMMENTS

- If a job or a file was held with a command other than SD, the SU Command will not remove the hold.
- If *B* or *S* is not given, both jobs and spools are restarted.

??

Purpose:

To request an error explanation.

Format

↑?? [,*n*] [,*lu*]

Where

n is the error in question. If *n* is not specified, the last error printed is expanded. If *n* = 99, all possible error messages are expanded.

lu is the logical unit number of the device on which it is desired to to have the report printed.

COMMENTS

- The second parameter, *lu*, may be used only when *n* = 99.
- See Section IX for spool error codes.

OUTSPOOL FILE STATES

An outspool file can be in four possible states:

- A Active (in process of dumping to outspool device)
- AH Active-hold (in process of dump but being held until restarted)
- W Waiting for outspooling
- H Hold in wait state

Figure 8-1 indicates how the commands CS (change status) and RS (respool) affect the state of outspool files.

From wait state, a file may be moved to hold state with a CS command, or to active state when it is selected as the next file to be outspooled.

From hold state, it is only possible to return to wait state.

From active state, a file may go to wait state or to active-hold state.

From active-hold state, a file may move to wait or to active state.

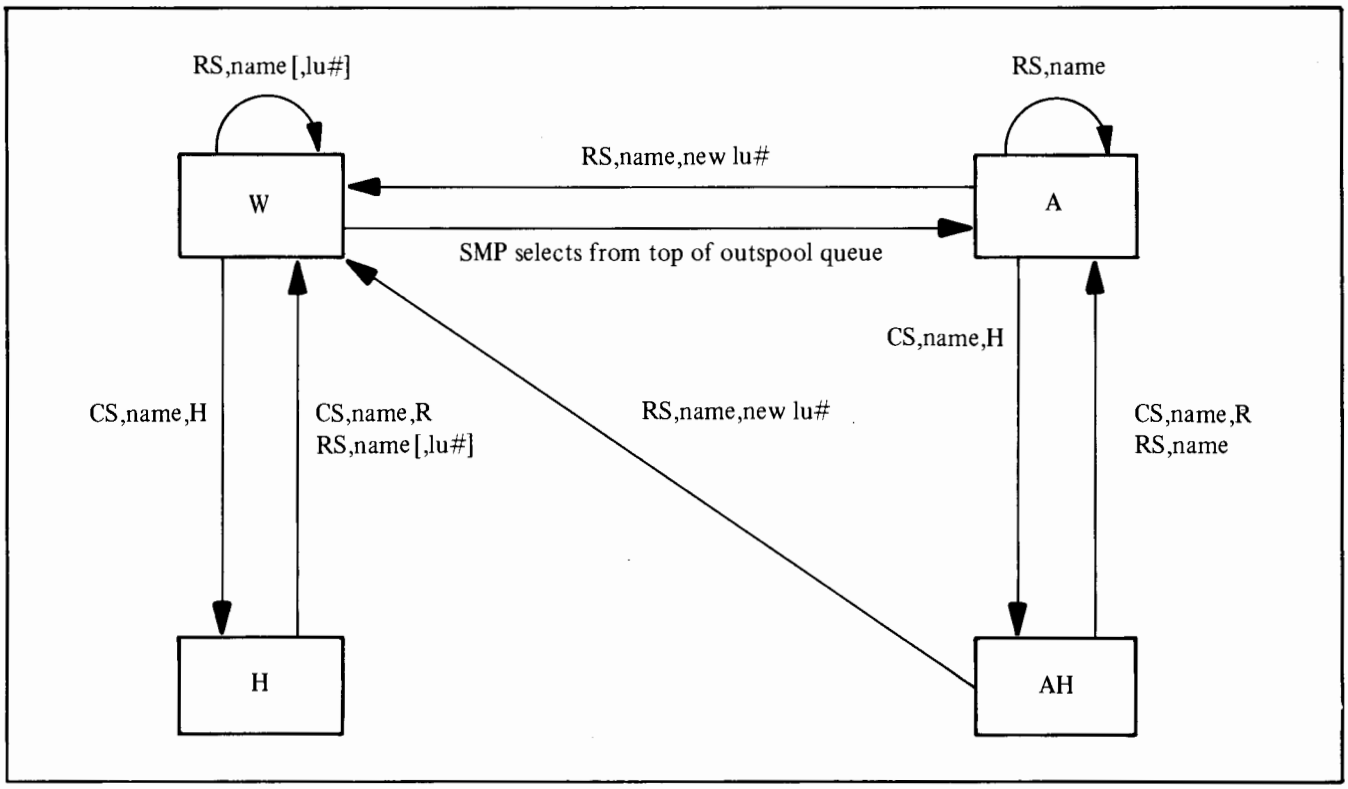


Figure 8-1. Changing Outspool File States

SECTION IX
SPOOL ERROR CODES

Table 9-1. GASP Error Codes

Code	Meaning
ϕ	No error
−1	Disc error
−2	Duplicate file name
−4	More than 32,767 records in a type 2 file
−6	CR or file not found or no room
−7	Bad file security code
−8	File open or lock rejected
−12	EOF or SOF error
−13	Disc locked
−14	Directory full
1	Disc error
2	Number out of range
3	Bad job number
4	Illegal status
5	Illegal command
6	Not found
55	Missing parameter
56	Bad parameter

CHAPTER III
INSTALLATION

SECTION X
FMP SYSTEM INSTALLATION

INTRODUCTION

This section describes configuration of the File Management Package (FMP) into the computer system. For configuration of the complete Batch Spool Monitor, also refer to Section XI, Spool System Installation.

Configuration of the FMP consists of two parts. The first part involves incorporating the package into the RTE System. The second part involves initializing the system and auxiliary discs with FMGR.

The FMGR initialization process is identical whether the operating system is RTE-II or RTE-III. Installation differs only in that the page requirements of RTE-III force a background memory size of 7K when current page linking is used. This includes 6K for the program and 1K for the base page.

FMP INSTALLATION

The FMP must be configured into either RTE System during the Program Input Phase of system generation - no provisions exist for on-line loading. (Refer to "Program Input Phase" in the RTE Manual.) The various modules of the FMP are numbered according to the order of their loading; other distinctions are as follows.

FMGR

The operator interface, FMGR, has a priority of 90 and is segmented into nine parts. When current page linking is not used, FMGR requires slightly less than 5K words of background disc-resident area. When current page linking is used in the background, slightly more than 5K is required for RTE-II, and 6K for RTE-III plus 1K for the base page.

NOTE

FMGR is not a heavy user of links, and therefore it should load with no problems without using current page linking.

D.RTR

Subroutine D.RTR has a priority of one, requires a few words over 1K of area, and is supplied as a foreground disc-resident type 2 program. However, if space permits, it is recommended that it be made core resident for greater speed. This change can be made during the parameter input phase of system generation. The only constraint is that D.RTR have a higher priority than any program using the FMP. When D.RTR is installed as a core resident program, the boundary may precede a page by 200 (octal) words.

FMGR INITIALIZATION

Each time the RTE System is loaded from the disc the FMGR program is scheduled. The first time that FMGR is scheduled it detects that the FMP has not been initialized to the system and auxiliary discs. Once this initialization procedure (described in the following steps) takes place, initialization is no longer necessary.

When FMGR is scheduled, it obtains all available tracks on the system and auxiliary discs, and assigns the tracks to itself. FMGR then prints on the system teleprinter (TTY):

```
FMGR 002
.
.
```

This is a request for the user to initialize the system disc (LU2) using the IN Command.

The following command initializes the system disc (logical unit 2) using the master security code AG and cartridge ID label, ADELE. The first track is 100, a number at least 8 tracks greater than the last system track used for generation.

```
:IN,AG,-2,2,ADELE,100
```

NOTE

The security code entered at this time will be the system master security code. The code can be blank (no security) or any two characters (except a colon, comma, or leading blank). The two characters need not be printing characters. Remember, the code may not be obtained with any FMGR Command once it is set, so be sure and remember it.

After a successful initialization of the system disc, FMGR checks to see if there is an auxiliary disc. If so, FMGR prints on the system TTY.

```
FMGR 003
•
•
```

This is a request for the user to initialize the auxiliary disc (LU3) using the IN Command. If no tracks are to be assigned for the auxiliary disc, its label should be specified as zero. For example

```
:IN,JB, 3,0
```

When a successful initialization is completed, FMGR assigns the tracks as per the IN Command parameters. FMGR then terminates (no message is printed) and returns control to the RTE-II System.

NOTE

The IN Command requires that the first file track on the system disc be at least 8 more than the highest system track.

SECTION XI
SPOOL SYSTEM INSTALLATION

INTRODUCTION

This section describes configuration of the Batch-Spool Monitor into the computer system. If it should be desired to use only the File Management Package (FMP) without the Spool Monitor (SM), refer to Section X, FMP System Installation.

Initialization of the Spool Monitor is the same whether the operating system is RTE-II or RTE-III. The parameter phase of Spool Monitor installation differs for the two systems because of the different program types used in RTE-II and RTE-III. Refer to the discussion of program type codes in the Parameter Input Phase of system generation in either Operating System manual

SM INSTALLATION

The Spool Monitor must be configured into the RTE-II System during the Program Input Phase of system generation no provisions exist for on-line loading. (Refer to the "Program Input Phase" section of the HP Real Time Executive Software System Programming and Operating Manual.)

INPUT PHASE

Load the Spool Program relocatable tape (refer to Table 1-1 for part number identification) during the program input phase.

PARAMETER PHASE

Name	Priority	Program Type	
		RTE-II	RTE-III
JOB	30	2	2
GASP	30	3	3
SMP	30	2	18
EXTND	10	1	17
SPOUT	10	1	17
DVS43		0	0
SP.CL		-	30*

*This program is not used by RTE-II.

Optimal performance is provided by these system supplied program types. For some programs, the type code may be changed during installation if the following rules are observed.

- JOB For RTE-II must be either core resident (type 1 or 4) or foreground disc resident (type 2) to avoid competing for memory with FMGR.

  For RTE-III may be any program type as long as JOB does not compete for the same partition with FMGR If both JOB and FMGR are background disc resident there should be enough partitions to avoid competition

- GASP For both RTE-II and RTE-III, GASP should be in background disc memory because GASP runs only under operator control

- SMP For RTE-II should be foreground disc resident (type 2), although it may be memory resident, there is no good reason for this allocation.

  For RTE-III should be real-time disc resident (type 18) and may be memory resident (type 17) but must be able to access SSGA.

- EXTND For RTE-II must be foreground core resident (type 1). *Do not change program type.*

  – For RTE-III must be memory resident (type 17) with access to SSGA.

- SPOUT – For RTE-II should be foreground core resident (type 1); it may be foreground disc resident (type 2) but this slows the system severely.

  – For RTE-III should be memory resident (type 17); it may be disc resident (types 18 or 19) but this could slow the system.

  – For either system, SPOUT and D.RTR must *not* be in the same disc area.

- DVS43 – For both RTE-II and RTE-III must be system resident (type 0). *Do not change program type.*

- SP.CL – For RTE-III only, provides a spool communication area that must be in SSGA module (type 30). *Do not change the program type.*

SYSTEM RESOURCES REQUIRED

The Spool Monitor uses the following resources:

4 Resource numbers (RN) (one for the logical unit locking during outspooling, two for file locking (JOBFIL and SPLCON), and one for resource waiting)

2 Class numbers (one for outspooling, and one for communication with SMP)

2+n Batch Logical Unit Switch Table entries (one for inspool and one for outspool)

where:

n is the maximum number of LU Commands expected in a job.

I/O TABLES REQUIRED

There should be several (at least six) EQT, LU, and Interrupt Table entries set up for spooling.

The format of the EQT entry is:

XX,DVS43,X = 18

where:

XX is a unique unused select code, in the range 10 (octal) through 77 (octal).

The format of the LU entry is:

n

where:

n is the EQT number.

The format of the Interrupt Table entry is:

XX,EQT,n

where:

n is the EQT number.

XX is the unique unused select code in EQT,n.

SYSTEM MEMORY REQUIRED

SPOUT attempts to keep four requests in System Available Memory for each device to which it is outspooling. For this reason enough System Available Memory must be provided to handle at least four of the longest expected outspool records. This space can be calculated on the basis of a maximum record size of 67 words plus a 10 word header for each record. Thus, 77 words times 4 records = the minimum System Available Memory required by SPOUT. Note that additional System Available Memory may be needed by other programs.

GASP INITIALIZATION

The GASP initialization function is entered when GASP is scheduled and JOBFIL does not exist. Its function is to set up and initialize JOBFIL, SPLCON, and the spool pool files. This is done by requesting from the user the following information:

MAX NUMBER OF JOBS, JOB FILE DISC?

Enter maximum number of jobs on system at one time. (= 1 + number of jobs that can be waiting on the queue). Each 8 jobs requires one block in JOBFIL.

NUMBER OF SPOOL FILES (5 TO 80)?

Enter number of files for system input/output spooling — these are the spool pool files mentioned in Section VIII, Spool Commands, under the heading "Spool File Conventions." Maximum is 80.

SIZE OF SPOOL FILES (IN BLOCKS)?

Enter number of blocks in each spool file.

NUMBER, LOCATION OF SPOOL FILES?

Enter number of spool files and disc label. The question will be asked again until E (for "end") is entered.

MAXIMUM NUMBER ACTIVE AND PENDING OUTSPOOLS?

The number entered should be the maximum number of pending outspools and other active input spools desired. This number should be at least as large as the number of spool pool files.

ENTER OUTSPOOL LU

Enter an outspool LU#. The command will be repeated until "E" is entered to terminate the initialization. The LU numbers entered here correspond to the LU's of the actual spool devices used to dump outspool files. The spool monitor will only dump files to one of these specified LU's. It is most efficient to put the list LU (6) first. After a successful initialization, GASP terminates.

If the Batch-Spool Monitor is shut down by the GASP shut-down command (SD), jobs remaining in the job queue and spools still in the outspool queue will be processed when the next GASP start-up command (SU) is issued.

If the system goes down for any other reason, jobs in the job queue are processed when:

:JO processes an :XE or :EO command, or

FMGR processes an :EX command or otherwise terminates normally, or

FMGR is run with -1 as the first parameter (e.g., RU,FMGR,-1...).

Spools left in the outspool queue when the system shuts down other than by the SD command will be outspooled when:

A job is run, or

program JOB is run.

APPENDIX A
TABLES

This Appendix contains the following tables.

Table A-1. Cartridge Directory Format

The Cartridge Directory is located on the first two sectors of the last track on the system cartridge (LU2). It contains key pointers to all mounted cartridges within the FMP realm.

Sectors	Word	Contents
0/1	0	LU first cartridge
	1	Last track for FMP
	2	Label word
	3	Lock word
	4	LU second cartridge etc. • • • 0 – end of cartridge specs (up to 31 cartridges)
	124	0
	125	Set up (initialized) code word
	126	Set up security code
	127	Spare

Word 125 is the sum of the words in locations 1650 through 1657, and 1742 through 1764.

This order of cartridges can be changed. Refer to the DC Command.

Lock word is either 0, not locked, or ID segment address of locking program. Only FMGR locks the cartridge in supported software. Locked cartridges are available only to the locker to other callers; they are considered not mounted.

Table A-2. File Directory Format

The first entry in each File Directory is the specification entry for the cartridge itself. Each entry is 16 words long. The directory will start on the last FMP track of each cartridge in sector 0 for all but the system cartridge (LU2), on which it starts in the next logical directory block.

Word	Contents
0 – 2	Six-character information pack label
3	Label word (positive integer)
4	First available track
5	Next available sector
6	Number of sectors per track
7	Last available track for files + 1 (i.e., lowest directory track)
8	Negative of number of tracks in directory
9	Next available track
10	First bad track (or zero)
	⋮
15	Sixth bad track (or zero)

Each cartridge has a short label (one word) and a long label. The short label is used for addressing the cartridge, the long label (six ASCII characters) is never used by the FMP and is only used for further ID when listing directories.

The sign bit will be set on word 1.

The directory sector address is obtained from the block address by the following formula:

Sector address = (block *14) mod #S/T where #S/T is the number of sectors per track. Directory blocks are 128 words long.

After the specification entry comes the entries for each file. Each entry is 16 words long. Note that for type 0 file entries, words 3 through 7 are different.

Word	Contents
0	NAME 1,2
1	NAME 3,4
2	NAME 5,6
3	file type
4	Track
5	Extent/sector
6	Number of sectors in file
7	Record length (type 2 files only)
8	Security code

Table A-2. File Directory Format (Continued)

9 10 11 12 13 14 15	Open Flags

For type 0 files, entries 3 through 7 are:

3	0 (type)
4	Logical unit number
5	End-of-file number
Specified at Creation	01XX for MT 10XX for paper tape 11XX for line printer

NOTE

XX = logical unit number

6	Spacing legal code Bit 15 = 1 backspace legal Bit 0 = 1 forward space legal
7	READ/WRITE Code Bit 15 = 1 input legal Bit 0 = 1 output legal

If word 0 = 0 then the end of directory.

If word 0 = −1 then the entry was purged.

Up to seven PGM'S may have the same file open at any given time. The open flags are either zero (not open), or are the ID address of the program the file is opened to. An exclusive open is indicated by the sign being set on the only open flag.

Each time a file is opened, the ID tables for each program that already has the file open are checked to see if the program is dormant. If dormant, i.e., if the point of suspension is zero, the open flag will be set to zero. This does not close the DCB nor post a possibly unwritten record therein.

Table A-3. Data Control Block Format

The Data Control Block (DCB) is a 144-word array provided by the user program with one DCB required for each file opened. The DCB is used to:

1. Prevent unnecessary directory accesses (usually an access is not needed once a file is opened).
2. Keep track of current position within file.
3. Provide a packing/unpacking buffer

Word	Contents
0	Track/LU — Address of directory entry for the file
1	Offset Sector — Address of directory entry for the file
2	File type (may be overridden at open)
3	Track
4	Sector
5	Number of sectors in file
6	Record length (type 2 files only)
7	Security code, open mode, and buffer size
	Bit 15 = 1: Codes agree
	Bit 15 = 0: Codes do not agree
	Bit 0 = 1: Update open
	Bit 0 = 0: Standard open
	Bit 14 = 1: DCB buffer size
8	Number of sectors per track
9	Open – Close flag
	open = ID segment address
	close ≠ ID segment address
10	track — Current location in file
11	Sector — Current location in file
12	Location of next word — Current location in file
13	In buffer and write flag:
	Bit 15 = 0: Not in core
	Bit 15 = 1: In core
	Bit 0 = 1: Written on
	Bit 0 = 0: Not written on
	Bit 14 = 1: If EOF read
	Bit 14 = 0: If EOF not read

Table A-3. Data Control Block Format (Continued)

Word	Contents
14	Record count
15	Extent number
16	Buffer 128 words
:	
143	

For type 0 files, entries 2 through 6 are:

2	0 (type)
3	Logical unit number
4	End-of-file code
Specified at Creation	{ 01XX for MT 10XX for paper tape 11XX for line printer

NOTE

XX = logical unit number

5	Spacing legal code Bit 15 = 1 backspace legal Bit 0 = 1 forward space legal
6	READ/WRITE Code Bit 15 = 1 input legal Bit 0 = 1 output legal

The DCB is free for other use after a PURGE, CLOSE, and NAMF call. Once a file is open, the DCB is used to reference the file, the name no longer being needed or used.

Table A-4. Record Format For Disc Files

FIXED RECORD LENGTH FILES

Type 1 and 2 files will be written as presented. They will be packed and may cross sector and track boundaries.

RANDOM LENGTH FILES

Type 3 and above records will be preceded and followed by a length word which contains the length of the record exclusive of the two length words. A zero length record will consist of two zero words. An end of file will be indicated by a –1 for the length.

SAVE PROGRAM FILES

Save program files are created by the SP command as type 6 files; however, they will always be accessed as type 1.

The first two sectors of the file will be used to record ID information on the program as follows:

Word	Contents
0	–1 (EOF if not forced to type 1)
1 – 6	Not used
7	Priority
8	Primary entry point
9 – 14	Not used
15	Program type
16 – 17	Not used
18 – 20	Time parameters
21 – 22	Not used
23	Low main address
24	High main address
25	Low base-page address
26	High base-page address
27 – 28	Not used
29	Checksum of words 0 through 27.
30	System setup code word (same as disc directory word 125)
31 –127	Not used

The rest of the file will be an exact copy of the original program tracks.

APENDIX B
SUMMARY OF FMP ERROR PRINTOUTS

FMGR ERROR CODES

ERROR	MEANING
-17	ILLEGAL READ/WRITE ON TYPE 0 FILE
-16	ILLEGAL TYPE OR SIZE=0
-15	ILLEGAL NAME
-14	DIRECTORY FULL
-13	DISC LOCKED
-12	EOF OR SOF ERROR
-11	OCB NOT OPEN
-10	NOT ENOUGH PARAMETERS
-09	ATTEMPT TO USE APOSN OR FORCE TO 1 A TYPE 0 FILE
-08	FILE OPEN OR LOCK REJECTED
-07	BAD FILE SECURITY CODE
-06	CR OR FILE NOT FOUND OR NO ROOM
-05	RECORD LENGTH ILLEGAL
-04	MORE THAN 32786 RECORDS IN A TYPE 2 FILE
-03	BACKSPACE ILLEGAL
-02	DUPLICATE FILE NAME
-01	DISC ERROR
000	BREAK
001	DISC ERROR-LU REPORTED
002	INITILIZE LU 2!
003	INITILIZE LU 3!
004	ILLEGAL RESPONSE TO 002 OR 003
005	REQUIRED TRACK NOT AVAILABLE - RELATIVE TAT POSITION REPORTED
006	FMGR SUSPENDED
007	CHECKSUM ERROR
008	D.RTR NOT LOADED
009	ID-SEGMENT NOT FOUND
010	INPUT ERROR
011	DO OF,XXXXX,8 ON NAMED PROGRAMS
012	DUPLICATE DISC LABEL OR LU
013	TR STACK OVERFLOW
014	REQUIRED ID-SEGMENT NOT FOUND
015	LS TRACK REPORT
016	FILE MUST BE AND IS NOT ON LU 2 OR 3
017	ID SEGMENT NOT SET UP BY RP
018	PROGRAM NOT DORMANT
019	FILE NOT SET UP BY SP ON CURRENT SYSTEM
020	ILLEGAL TYPE 0 LU
021	ILLEGAL DISC SPECIFIED
022	COPY TERMINATED
023	DUPLICATE PROGRAM NAME.
047	SPOOL SETUP FAILED
048	GLOBAL SET OUT OF RANGE
049	CAN'T RUN RP'ED PROG.
050	NOT ENOUGH PARAMETERS
051	ILLEGAL MASTER SECURITY CODE
052	ILLEGAL LU IN RESPONSE TO 002 OR 3
053	ILLEGAL LABEL OR ILABEL
054	DISC NOT MOUNTED
055	MISSING PARAMETER
056	BAD PARAMETER
057	BAD TRACK NOT IN FILE AREA
058	LG AREA EMPTY
059	REPORTED TRACK UNAVAILABLE
060	DO YOU REALLY WANT TO PURGE THIS DISC? (YES OR NO).

APPENDIX C
SUMMARY OF FMP CALLS

ASSEMBLY LANGUAGE FORMAT


```
        EXT     name    Declaration of subroutine.
         .
         .
        JSB     name    Transfer control to name.

        DEF     RTN     Return address.

        DEF     IDCB    DCB buffer address.

        DEF     IERR    Error code buffer address.

        DEF     pl      Define addresses
         .               of parameters.
         .
        DEF     pn

RTN     return point
         .
         .
IDCB    BSS 144 + n     DCB buffer.

IERR    BSS 1           Error code returned here.

pl      ---             Actual parameter
                        values.
pn      ---
```

For each FMP call (except FSTAT and RWNDF), this appendix includes only the parameters *pl* through *pn*.

FORTRAN/FORTRAN IV FORMAT

```
DIMENSION IDCB (144 + n)
CALL name (IDCB, p2 . . .,pn)
```

Where:

IDCB is the Data Control Block, and *p2* through *pn* are either integer values or integer variables defined elsewhere in the program. The underlined parameters are optional.

APOSN

Purpose:

This routine sets the address of the next record for any file except type 0.

Assembly:

IREC	DEC	Record number of next record
IRB	DEC	Relative block address of next record (optional)
IOFF	DEC	Block offset of next record (optional)

FORTRAN:

DIMENSION IDCB (144 + n)
IREC = *a* Record number of next record
IRB = *b* Relative block address of next record (optional)
IOFF = *c* Block offset of next record (optional)
CALL APOSN (IDCB, IERR, IREC, IRB, IOFF)
Error code is returned in IERR.

CLOSE

Purpose:

This routine closes the DCB and makes the file available to other callers. CLOSE also optionally truncates the file size.

Assembly:

ITRUN DEC

+*n* = number of blocks to be deleted from the end of the file when it is closed (optional)

–*n* = retain main file, delete extents

n = 0 = standard close

FORTRAN:

DIMENSION IDCB (144 +n)
ITRUN = *m*

+*m* = number of blocks to be deleted from the end of the file when it is closed (optional)

–*m* = retain main file, delete extents

m = 0 = standard close

CALL CLOSE (IDCB, IERR, ITRUN)
Error code is returned in IERR.

CREAT

Purpose:

This routine closes the DCB (if open), and then creates the named file on the specified disc with the specified number of blocks. The file is left open exclusively to the caller on successful completion of the call. This call will not create a type 0 file.

Assembly:

NAME	ASC	File name
ISIZE	DEC	Number of blocks in the file. If negative, use all of disc
	DEC	Record length (used for type 2 file only)
ITYPE	DEC	File type.
ISECU	DEC	Security code (optional)
ICR	DEC	Cartridge Label (optional)
IDCBS	DEC	Number of words in IDCB array (optional)

FORTRAN:

DIMENSION IDCB (144 + n), NAME(3), ISIZE(2)

NAME(1)	=*xxxxx*B	First two characters
NAME(2)	=*xxxxx*B	Second two
NAME(3)	=*xxxxx*B	Last two characters

ISIZE(1)	=*a*	Number of blocks in the file. If negative, use all of disc.
ISIZE(2)	=*b*	Record length (used for type 2 file only)
ITYPE	=*c*	File type
ISECU	=*d*	Security code (optional)
ICR	=*e*	Cartridge Label (optional)
IDCBS	=*f*	Number of words in IDCB array (optional)

CALL CREAT (IDCB, IERR, NAME, ISIZE, ITYPE, ISECU, ICR, IDCBS)
Error code, or number of sectors, is returned in IERR.

FCONT

Purpose:

This routine sends the standard RTE I/O Control request to type 0 (non-disc) files. It has no effect on other files.

Assembly:

ICON1	See Section II for control information
ICON2	Function code – see Section II (optional)

FORTRAN:

```
DIMENSION IDCB (144 + n)
ICON1 = aB          Control word
ICON2 = b           Control word (optional)
CALL FCONT (IDCB, IERR, ICON1, ICON2)
Error code is returned in IERR.
```

FSTAT

Purpose:

This routine returns information on all cartridge labels in the system.

Assembly:

```
ISTAT  BSS  125     Buffer of 125 words
```

FORTRAN:

```
DIMENSION ISTAT (125)     Status buffer
CALL FSTAT (ISTAT)
```

IDCBS

Purpose:

This routine determines the size of the DCB buffer area actually used.

Assembly:

IDCB BSS *n*

FORTRAN:

ISIZE = IDCBS (IDCB)

LOCF

Purpose:

This routine formats and returns location and status information from the DCB.

Assembly:

IREC	BSS	Next record number
IRB	BSS	Relative block of next read (optional)
IOFF	BSS	Block offset of next record (optional)
JSEC	BSS	Number of sectors in the file (optional)
JLU	BSS	File logical unit (optional)
JTY	BSS	File type (optional)
JREC	BSS	Record size (optional)

FORTRAN:

DIMENSION IDCB (144 + n)
CALL LOCF (IDCB, IERR, IREC, IRB, IOFF, JSEC, JLU, JTY, JREC)
Error code returned in IERR
Next record number returned in IREC
Relative block of next read returned in IRB
Block offset of next record returned in IOFF
Number of sectors in the file returned in JSEC
File logical unit returned in JLU
File type returned in JTY
Record size returned in JREC

NAMF

Purpose:

This routine closes the DCB (if open) and then renames the specified file.

Assembly:

NAME	ASC	File's present name
NNAME	ASC	File's new name
ISECU	DEC	Security code (optional)
ICR	DEC	Cartridge label (optional)

FORTRAN:

DIMENSION IDCB (144 + n), NAME(3), NNAME(3)

NAME(1)	=*xxxxx*B	Present name, first two characters
NAME(2)	=*xxxxx*B	Second two
NAME(3)	=*xxxxx*B	Last two characters
NNAME(1)	=*xxxxx*B	New name, first two characters
NNAME(2)	=*xxxxx*B	Second two
NNAME(3)	=*xxxxx*B	Last two characters
ISECU	=*a*	Security code (optional)
ICR	=*b*	Cartridge label (optional)

CALL NAMF (IDCB, IERR, NAME, NNAME, ISECU, ICR)

Error code is returned in IERR

OPEN

Purpose:

This routine closes the DCB (if open), and then opens the named file.

Assembly:

NAME	ASC	File name
IOPTN	OCT	Open options (see Section II)
ISECU	DEC	Security code (optional)
ICR	DEC	Cartridge label (optional)
IDCBS	DEC	Number of words available in DCB

FORTRAN:

DIMENSION IDCB (144 + n), NAME(3)

NAME(1)	=*xxxxx*B	First two characters
NAME(2)	=*xxxxx*B	Second two
NAME(3)	=*xxxxx*B	Last two characters
IOPTN	=*a*	Open control word
ISECU	=*b*	Security code (optional)
ICR	=*c*	Cartridge label (optional)
IDCBS	=*d*	Number of words available in DCB (optional)

CALL OPEN (IDCB, IERR, NAME, IOPTN, ISECU, ICR, IDCBS)
Error code is returned in IERR

POSNT

Purpose:

This routine causes the next read or write to access the given record in any file type.

Assembly:

```
NUR     DEC     New record number

IR      DEC     Absolute vs. relative control parameter for NUR (optional)
```

FORTRAN:

```
DIMENSION IDCB (144 + n)
```

NUR = *a*	NUR>	0	Forward positioning, skip NUR records.
	NUR<	0	Backward positioning, skip NUR records.
	NUR =	0	No operation.
IR = *b* (optional)	IR =	0	The new record indicated by NUR is relative to the present position in the file.
	or not present		
	IR ≠	0	The new record indicated by NUR is the absolute number starting from the first record in the file (record number 1). Note that NUR must be > 0.

```
CALL POSNT (IDCB, IERR, NUR, IR)
```
Error code is returned in IERR

POST

Purpose:

This routine posts the DCB buffer on the disc (if needed) and clears flags indicating data in core.

Assembly:

```
IDCB    BSS  144 + n
IERR    BSS  1          (optional)
```

FORTRAN:

```
DIMENSION IDCB (144 + n)
CALL POST (IDCB, IERR)
```

PURGE

Purpose:

This routine closes the DCB (if open), and then deletes the named file and all of its extents.

Assembly:

NAME	ASC	File name
ISECU	DEC	Security code (optional)
ICR	DEC	Cartridge label (optional)

FORTRAN:

DIMENSION IDCB (144), NAME(3)

NAME(1)	=*xxxxx*B	First two characters
NAME(2)	=*xxxxx*B	Second two
NAME(3)	=*xxxxx*B	Last two characters
ISECU	=*a*	Security code (optional)
ICR	=*b*	Cartridge label (optional)

CALL PURGE (IDCB, IERR, NAME, ISECU, ICR)
Error code is returned in IERR

READF

Purpose:

This routine reads a record from the file currently open to the DCB, to the user buffer.

Assembly:

IBUF	BSS	Data buffer
IL	DEC	Read request length (optional)
LEN	BSS	Actual read length returned here (optional)
NUM	DEC	Record number to be read (optional)

FORTRAN:

DIMENSION IDCB (144 + n), IBUF(*n*)

IL	= *m*	Length of IBUF (optional)
NUM	=*x*	Record number to be read (optional)

CALL READF (IDCB, IERR, IBUF, IL, LEN, NUM)
Error code is returned in IERR
Actual read length is returned in LEN (optional)

RWNDF

Purpose:

This routine rewinds type 0 files, and sets disc files so that the next record is the first record in the file.

Assembly:

```
IDCB  BSS  144 + n
```

FORTRAN:

```
DIMENSION IDCB (144 + n)
Call RWNDF (IDCB, IERR)
```

Error code is returned in IERR

WRITF

Purpose:

This routine writes a record on the file currently open to the DCB.

Assembly:

```
IBUF  BSS        Data buffer
IL    DEC        Write request length (optional)
NUM   DEC        Record number to be written (optional)
```

FORTRAN:

```
DIMENSION IDCB (144 + n), IBUF(m)
IL  =m           Length of IBUF (optional)
NUM =x           Record number to be written (optional)
CALL WRITF (IDCB, IERR, IBUF, IL, NUM)
```

Error code is returned in IERR.

APPENDIX D
SUMMARY OF COMMANDS

FMGR COMMANDS

namr = file name [:*security code* [:*label* [:*file type*
[:*file size* [:*record size*]]]]]

ON, FMGR [,*input* [,*log* [,*list* [,*severity code*]]]]
Schedule operator interface FMGR.

AB
Abort a job and do normal end-of-job cleanup.

AN, *message*
Write a message on the job list file.

CA, #, operand1, operation, operand 2, operation, operand 3 . . .
Calculate and set a value into a global parameter.

CL
List active cartridge labels.

CO, *label1*, *label2*
Copy files from *label1* to *label2*

CN [,*namr* [,*function* [,*subfunction*]]]
Issue specified control request to the indicated device.

CR,*namr*
Create file name.

CR, *namr*, *lu* [,*REad* | ,*WRite* | ,*BOth*] [,*BSpace* | ,*FSpace* | ,*BOth*] [,*EOf* | ,*LEader* | ,*PAge* | ,*numeric*] [,*BInary* | ,*AScii* | ,*numeric*]
Create type 0 file.

CS,*lu*, *RWind* | *PUrge* | *SAve* | *PAss* | *ENd* | *BUffer* | *NBuffer* | *NPass* [,*lu* #] [,*priority*]
Modify or change spool options chosen at setup.

DC, *label*
Dismount cartridge.

DL [, *label* [, *master security code*]]
List contents of file directory.

DP, [*P1* [, *P2* . . .] .]]	Display current parameters on the system console.
DU,*namr1*,*namr2* [,*record format, EOF control* [,*file #* [,*#files*]]]	
or	Transfer contents of one file to another file (does not create *namr2*).
DU,*namr1*,*namr2* [,*record format* [,*file #* [,*#files*]]] [,*EOF control*]	
EO	End a job.
IF, *p1*, *NE*, *p2* [,#] (*EQ*, *LT*, *GT*, *GE*, *LE*)	Do conditional branching in the command stream.
IN, [*master security code*] ,*label1, label2, id* [,*1st trk* [,*#dir trks* [,*#sec/trk* [,*bad tracks*]]]]	Initialize cartridge parameters.
IN,*master security code* *new security code*	Change master security code.
JO [, *name* [: *hr* : *min* : *sec*] [, *job priority* [, *spool priority* [, *NSpool*]]]]	Job control.
LG [,*#tracks*]	Allocate or release disc tracks for load-and-go.
LI,*namr* [, *Source* / , *Binary* / , *Directory*]	Print file contents.
LL,*namr*	Change assignment of list file
LO,*lu*	Change assignment of log device.
LS [,*lu*,*track*]	Set up base page word.
LU,*lu* [,*namr*]	Spool setup or logical unit switch (input only).
LU,*lu* [,*namr*[,*attribute*[,*lu #*[,*priority*]]]]	Spool setup or logical unit switch (output and output/input).
MC,*lu* [,*last track*]	Mount cartridge.
MR,*namr*	Transfer relocatable file to load-and-go area.
MS,*namr* [,*prog name* [,*IH*]]	Transfer source file to EDIT or prog name
OF,*prog*	Same as RTE system command OF,8.

PK [,*label*]	Pack a cartridge to recover purged areas.
PA[,*lu* [,*message*]]	Transfer control to specified device and point text on log device.
PU,*namr*	Purge a file and its extents from system.
RN,*namr*,*nuname*	Change the name of a file.
RP,*namr*	Restore program saved with SP Command.
RP,*namr*,*program*	Assign program's ID segment to *namr*.
RP,,*program*	Release program's ID segment.
RT,*name*	Release all disc tracks assigned to a program.
RU,*name* [,*parameters*]	Seek out and execute a specified program or transfer files.
SP,*namr*	Save a disc resident program.
SA, LS,*namr*	Save logical source area.
SA, LG,*namr*	Save load-and-go area.
SE [,*p1* [,*p2* . . . [,*p9*]] . . .]	Set global parameters 1G through 9G.
ST,*namr1*, *namr2* [,*record format*, *EOF control* [,*file #* [,*#files*]]]	
– or –	Store contents of one file to another file (creates *namr2*).
ST,*namr1*, *namr2* [,*record format* [,*file* [,*#files*]]] [,*EOF control*]	
SV,*number*	Change system log severity code.
TE,*message*	Send operator message to TTY.
TL,*hr* : *min* : *sec*	Set a run-time limit for RUn executed programs.
TR [,*namr* | ,*–integer*] [,*parameters*]]	Transfer control.
?? [,*number*]	Expand error message.
EX	Terminate FMGR.

SPOOL COMMANDS

ON, GASP [,*lu*]	Initiate GASP.
AB,##	Abort a job before it runs.
CJ,## ,*job priority* / ,*H* / ,*R*	Change job status.
CS,*name* ,*spool priority* / ,*H* / ,*R*	Change spool status.
DA KILL SPOOLING? YES	Deallocate all spool files.
DJ [,# / ,*name*]	Display job status.
DS [,*LU*]	Display spool status.
EX	Terminate GASP.
KS,*number* / ,*name*	Purge a spool from the output queue.
RS,*name* [*lu*]	Restart an outspool.
SD [,*B* / ,*S*]	Shut down the Batch-Spool Monitor.
SU [,*B* / ,*S*]	Start up a system that has been shut down.
?? [,*n*] [,*lu*]	Request an error explanation.

APPENDIX E
HP CHARACTER SET

Table E-1. ASCII/Octal Table

This table is presented in ascending order according to the octal equivalents of the ASCII characters.

ASCII Character	First Character Octal Equivalent	Second Character Octal Equivalent
NULL	000000	000000
SUM	000400	000001
EOA	001000	000002
EOM	001400	000003
EOT	002000	000004
WRU	002400	000005
RU	003000	000006
BELL	003400	000007
FE_0	004000	000010
HT/SK	004400	000011
LF	005000	000012
V_{TAB}	005400	000013
FF	006000	000014
CR	006400	000015
SO	007000	000016
SI	007400	000017
DC_0	010000	000020
DC_1	010400	000021
DC_2	011000	000022
DC_3	011400	000023
DC_4	012000	000024
ERR	012400	000025
SYNC	013000	000026
LEM	013400	000027
S_0	014000	000030
S_1	014400	000031
S_2	015000	000032
S_3	015400	000033
S_4	016000	000034
S_5	016400	000035
S_6	017000	000036
S_7	017400	000037
space	020000	000040
!	020400	000041
"	021000	000042
#	021400	000043
$	022000	000044
%	022400	000045
&	023000	000046
'	023400	000047
(	024000	000050
)	024400	000051
*	025000	000052
+	025400	000053
,	026000	000054
–	026400	000055
.	027000	000056
/	027400	000057

ASCII Character	First Character Octal Equivalent	Second Character Octal Equivalent
0	030000	000060
1	030400	000061
2	031000	000062
3	031400	000063
4	032000	000064
5	032400	000065
6	033000	000066
7	033400	000067
8	034000	000070
9	034400	000071
:	035000	000072
;	035400	000073
<	036000	000074
=	036400	000075
>	037000	000076
?	037400	000077
@	040000	000100
A	040400	000101
B	041000	000102
C	041400	000103
D	042000	000104
E	042400	000105
F	043000	000106
G	043400	000107
H	044000	000110
I	044400	000111
J	045000	000112
K	045400	000113
L	046000	000114
M	046400	000115
N	047000	000116
O	047400	000117
P	050000	000120
Q	050400	000121
R	051000	000122
S	051400	000123
T	052000	000124
U	052400	000125
V	053000	000126
W	053400	000127
X	054000	000130
Y	054400	000131
Z	055000	000132
[	055400	000133
\	056000	000134
]	056400	000135
↑ (Λ)	057000	000136
– (←)	057400	000137
ACK	076000	000174
ⓘ	076400	000175
ESC	077000	000176
DEL	077400	000177

BITS b_7 b_6 b_5					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4 ↓	b_3 ↓	b_2 ↓	b_1 ↓	COLUMN → / ROW ↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	¦
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	↑ (Λ)	n	~
1	1	1	1	15	SI	US	/	?	O	←	o	DEL

Figure E-1. ASCII Characters and Binary Codes

NOTES:

Standard 7-bit set code positional order and notation are shown below with b_7 the high-order and b_1 the low-order, bit position.

Example: The code for "R" is:

b_7	b_6	b_5	b_4	b_3	b_2	b_1
1	0	1	0	0	1	0

On some devices, the underscore (_) prints as a left arrow (←) and the up-arrow (↑) prints as a caret (Λ).

RTE-II SPECIAL CHARACTERS:

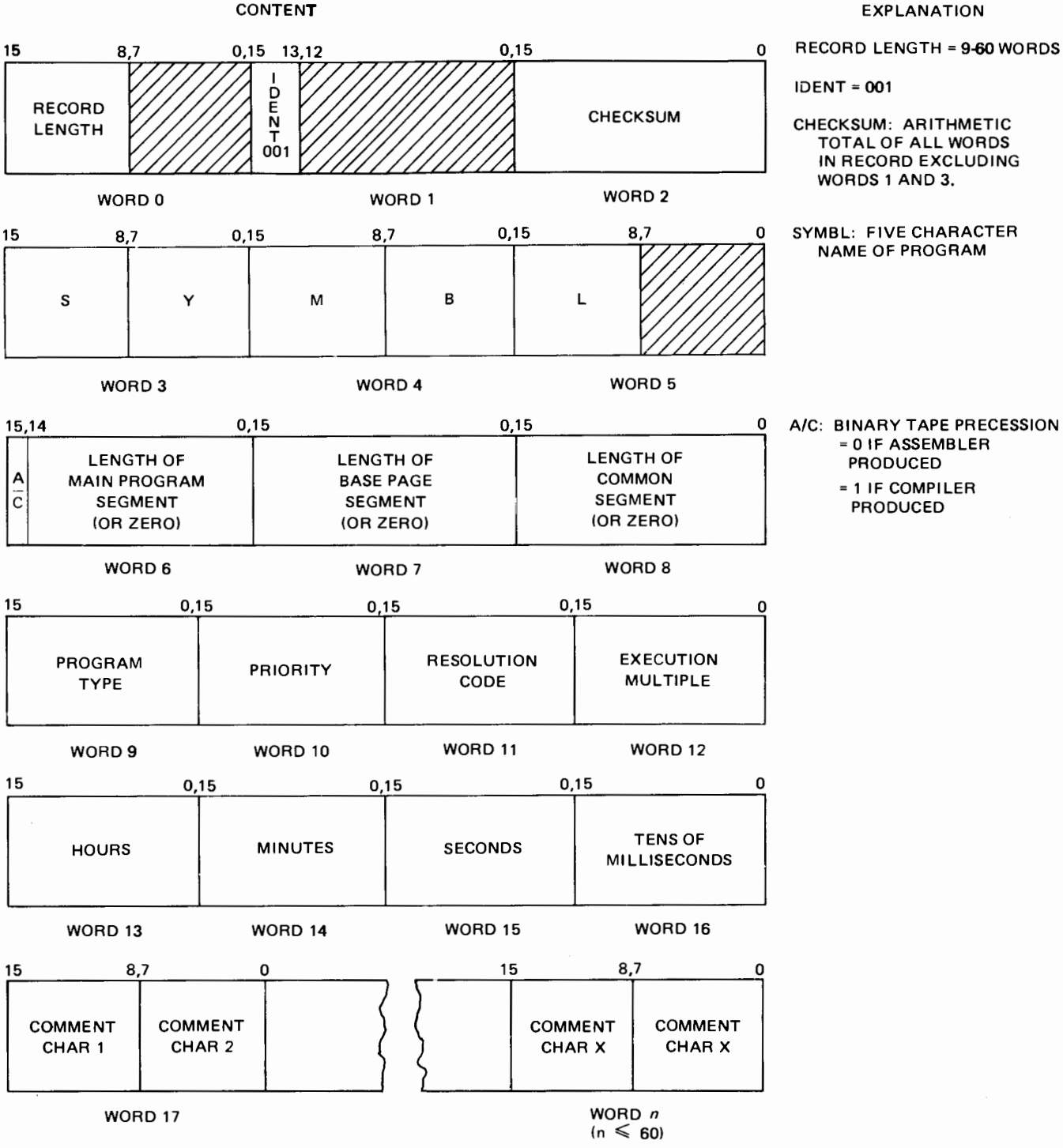
MNEMONIC	Value	Use
SOM (Control A)	1	Backspace
S1 (2600 Backspace) (Control Y)	31	Backspace
FE_O (Control H)	10	Backspace

Table E-2. Legend for Figure E-1

NUL	Null	DC1	Device Control 1
SOH	Start of Heading	DC2	Device Control 2
STX	Start of Text	DC3	Device Control 3
ETX	End of Text	DC4	Device Control 4
EOT	End of Transmission	NAK	Negative Acknowledgement
ENQ	Enquiry	SYN	Synchronous Idle
ACK	Positive Acknowledgement	ETB	End of Transmission Block
BEL	Bell (Audible signal)	CAN	Cancel
BS	Backspace	EM	End of Medium
HT	Horizontal Tabulation	SUB	Substitute
LF	Line Feed	ESC	Escape
VT	Vertical Tabulation	FS	File Separator
FF	Form Feed	GS	Group Separator
CR	Carriage Return	RS	Record Separator
SO	Shift Out	US	Unit Separator
SI	Shift In	SP	Space
DLE	Data Link Escape	DEL	Delete

APPENDIX F
RELOCATABLE TAPE FORMAT

NAM RECORD

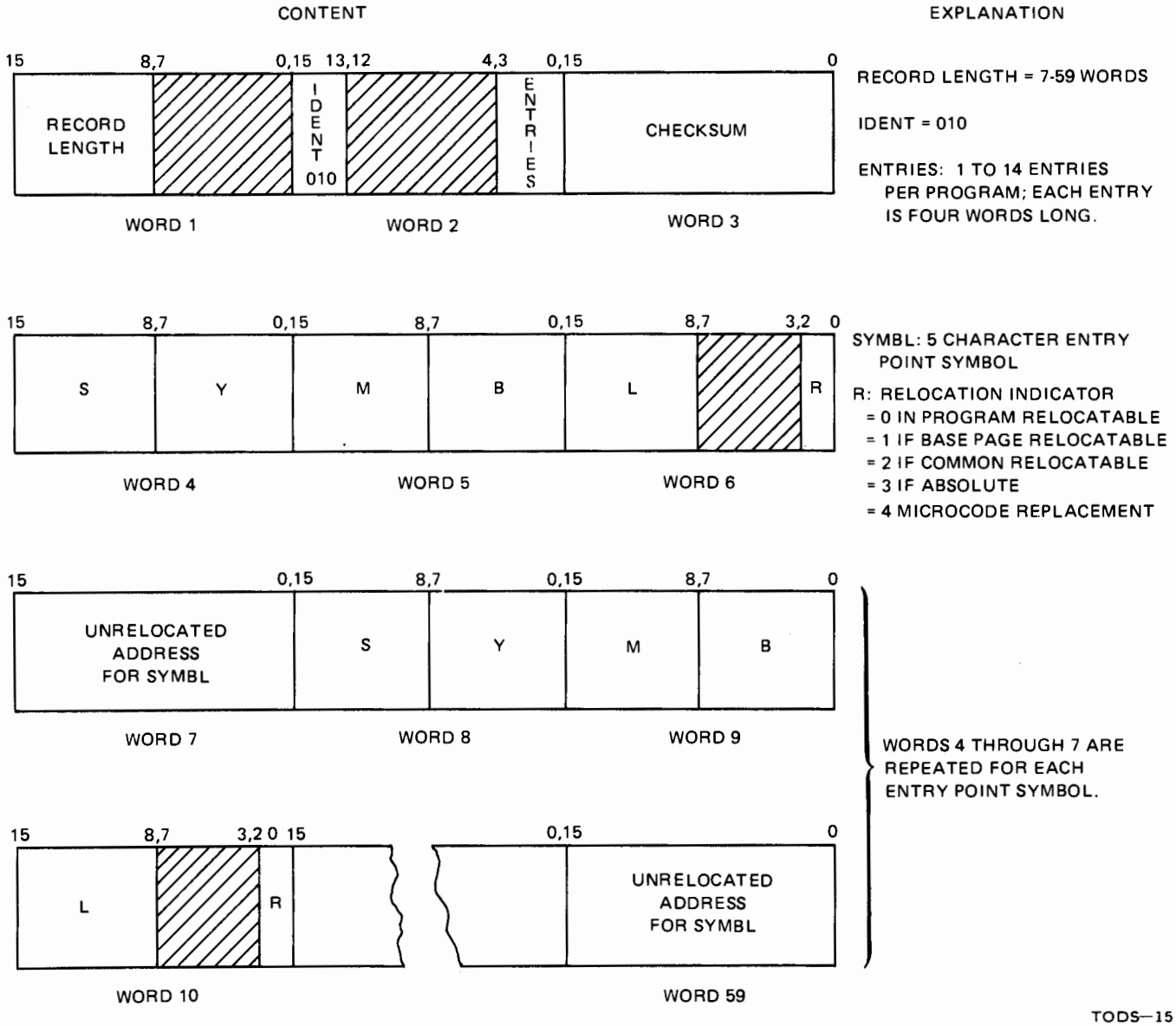


AMD CURRENTLY USES THE FIRST 15 CHARACTERS OF COMMENTS FOR PART NUMBER AND REVISION CODE. THIS FEATURE IS SUPPORTED BY AN AMD ASSEMBLER.

HATCH-MARKED AREAS SHOULD BE ZERO-FILLED WHEN THE RECORDS ARE GENERATED

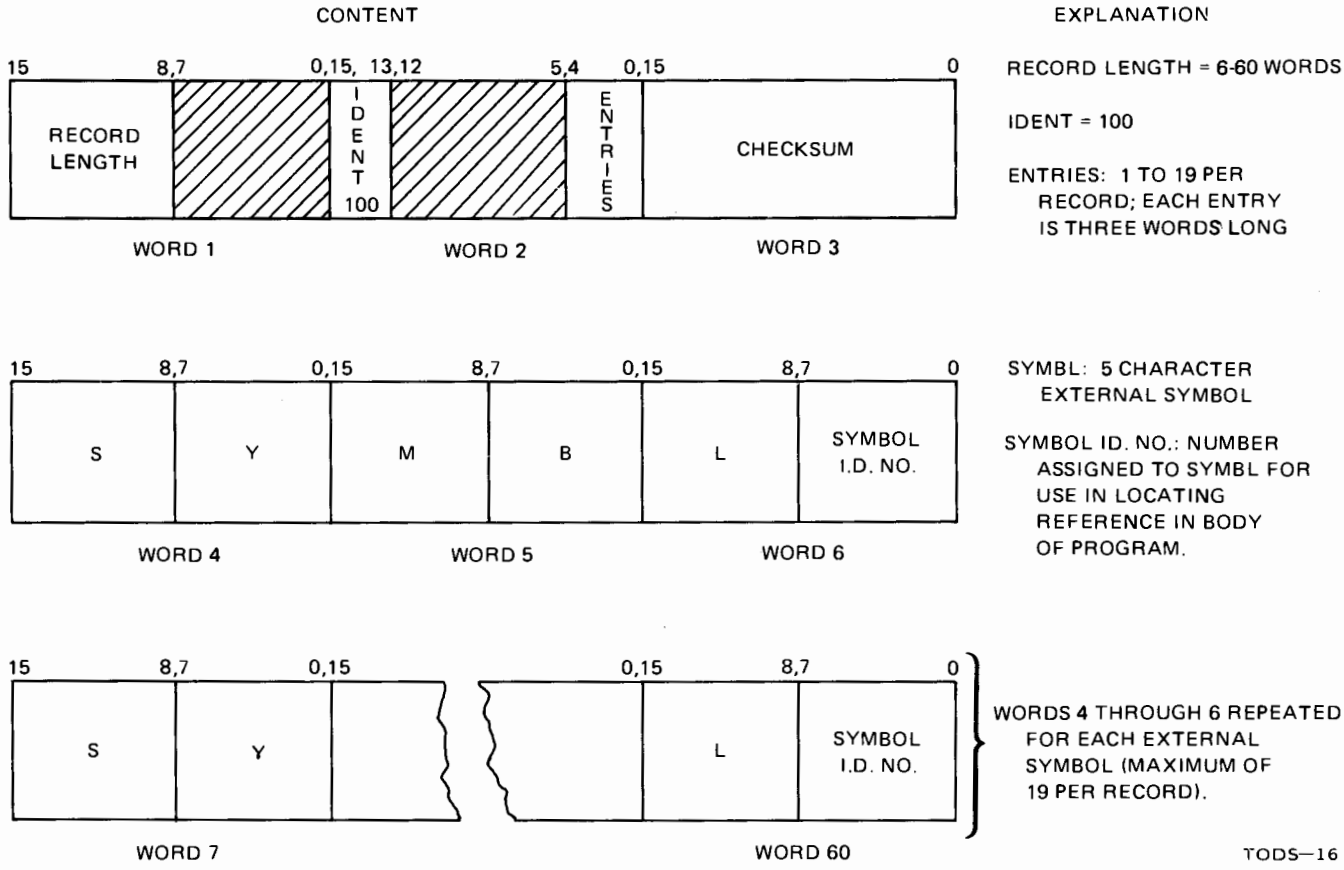
TODS—14

ENT RECORD



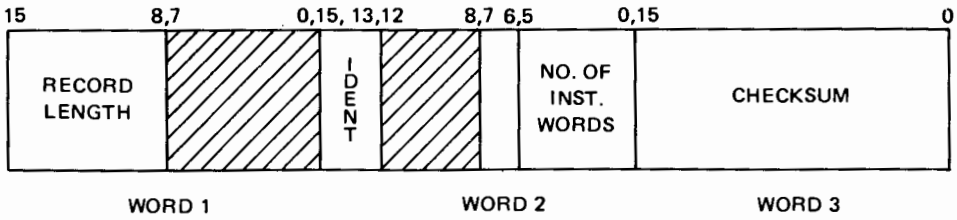
TODS—15

EXT RECORD


CONTENT
EXPLANATION
15
8,7
0,15, 13,12
5,4
0,15
0
RECORD LENGTH
IDENT 100
ENTRIES
CHECKSUM
WORD 1
WORD 2
WORD 3
RECORD LENGTH = 6-60 WORDS
IDENT = 100
ENTRIES: 1 TO 19 PER RECORD; EACH ENTRY IS THREE WORDS LONG
15
8,7
0,15
8,7
0,15
8,7
0
S
Y
M
B
L
SYMBOL I.D. NO.
WORD 4
WORD 5
WORD 6
SYMBL: 5 CHARACTER EXTERNAL SYMBOL
SYMBOL ID. NO.: NUMBER ASSIGNED TO SYMBL FOR USE IN LOCATING REFERENCE IN BODY OF PROGRAM.
15
8,7
0,15
0,15
8,7
0
S
Y
L
SYMBOL I.D. NO.
WORD 7
WORD 60
WORDS 4 THROUGH 6 REPEATED FOR EACH EXTERNAL SYMBOL (MAXIMUM OF 19 PER RECORD).
TODS—16

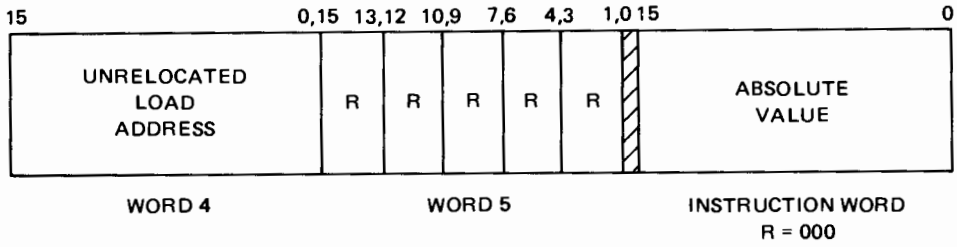
DBL RECORD

CONTENT



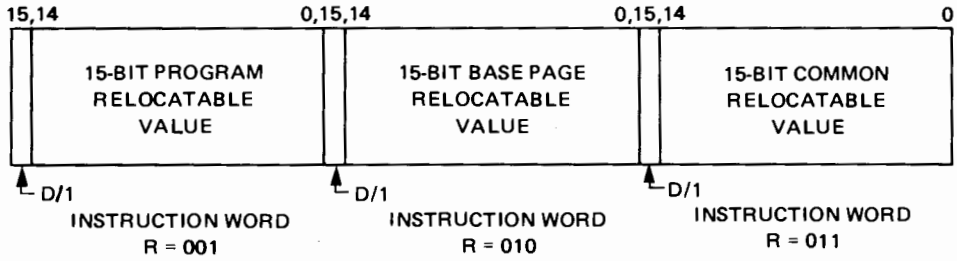
EXPLANATION

RECORD LENGTH = 6-60 WORDS
IDENT = 011
Z/C: RELOCATION OF LOAD ADDRESS
= 0 FOR BASE PAGE
= 1 FOR PROGRAM
= 2 FOR ABSOLUTE
= 3 FOR COMMON
NO. OF INST. WORDS: 1 TO 45 LOADABLE INSTRUCTION WORDS PER RECORD

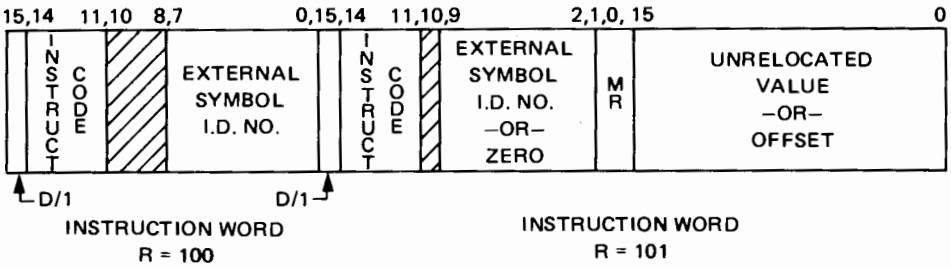


RELOCATABLE LOAD ADDRESS: STARTING ADDRESS FOR LOADING THE INSTRUCTIONS WHICH FOLLOW;

R's: RELOCATION INDICATORS:
000 = ABSOLUTE
001 = 15-BIT PROGRAM RELOCATABLE
010 = 15-BIT BASE PAGE RELOCATABLE
011 = 15-BIT COMMON RELOCATABLE
100 = EXTERNAL REFERENCE
101 = MEMORY REFERENCE



R_1 IS RELOCATION INDICATOR FOR INSTRUCTION $WORD_1$; R_2, FOR INSTRUCTION $WORD_2$; ETC.

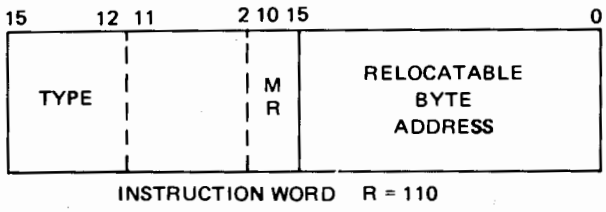


D/I: INDIRECT ADDRESSING

0 = DIRECT
1 = INDIRECT

MEMORY REFERENCE INSTRUCTIONS USE TWO WORDS, WITHIN THE TWO-WORD GROUP, "MR" INDICATES RELOCATABILITY OF OPERAND SPECIFIED IN SECOND WORDS:

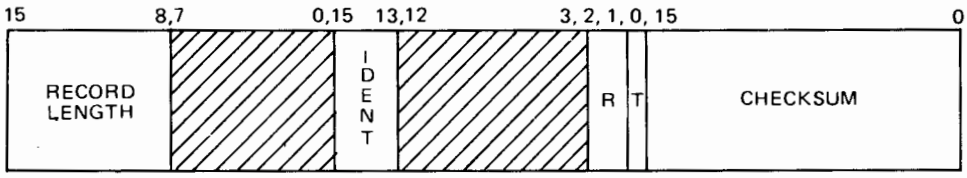
00 = PROGRAM RELOCATABLE
01 = BASE PAGE RELOCATABLE
10 = COMMON RELOCATABLE
11 = ABSOLUTE

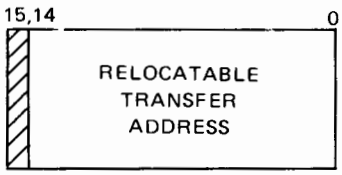


TODS—17

END RECORD

CONTENT





EXPLANATION

RECORD LENGTH = 4 WORDS
IDENT = 101

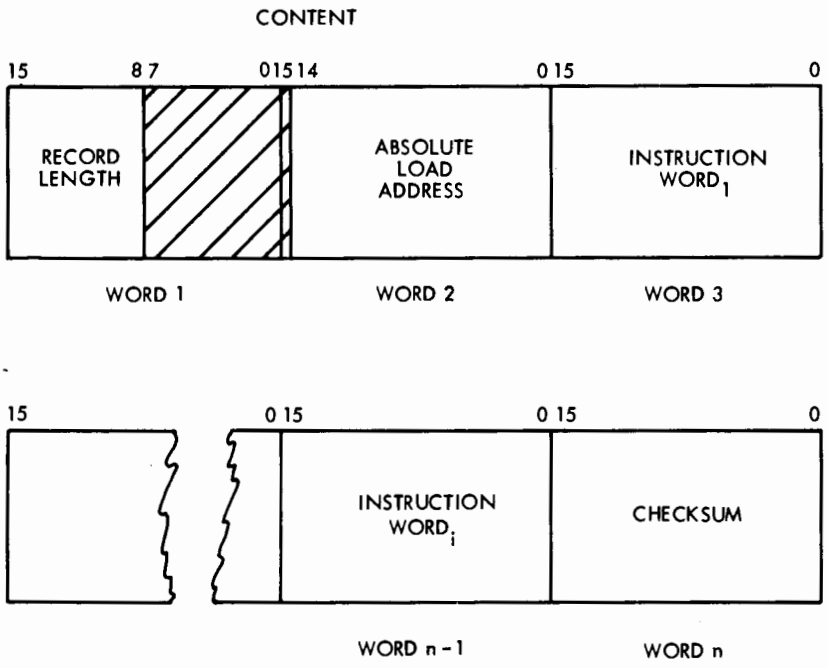
R: RELOCATION INDICATOR FOR TRANSFER ADDRESS

= 0 IF PROGRAM RELOCATABLE
= 1 IF BASE PAGE RELOCATABLE
= 2 IF COMMON RELOCATABLE
= 3 IF ABSOLUTE

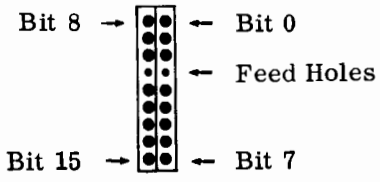
T: TRANSFER ADDRESS INDICATOR

= 0 IF NO TRANSFER ADDRESS IN RECORD
= 1 IF TRANSFER ADDRESS PRESENT

APPENDIX G
ABSOLUTE TAPE FORMAT



†Each word represents two frames arranged as follows:



APPENDIX H
THE BASIC PRINCIPLES OF INDEXING

INTRODUCTION

This appendix provides the user with some ideas on how to keep track of all the programs he has stored on the disc.

BASIC PRINCIPLES

There are six basic principles which apply whether one is dealing with a collection of complex programs, or thousands of technical articles reduced to abstract form.

1. Balance input and output effort.
2. Evaluate single entry and multiple entry files.
3. Describe items fully.
4. Control the vocabulary.
5. Know the subject.
6. Select appropriate index scheme.

Consider each point and see how they tie together to produce an effective index and retrieval system, but first, here are some necessary definitions.

DEFINITIONS

Data are numeric or quantitative notations. Data concepts are mutually exclusive. Data are easily manipulated by machines. The boiling point of lead is 1620°C. That is data.

Information is knowledge concerned principlaly with qualitative concepts or ideas. Information concepts are not mutually exclusive; the concepts interact and overlap. Information is not easily manipulated by machines as data. Information includes data. A qualitative discussion of ablation of plastics in heat shields of space vehicles would be information. Data are relatively easy to handle; information is more difficult.

Records are reports, abstracts, books, articles and catalogs.

Terms are used to represent concepts. In various systems they may be called descriptors, key words, uniterms, or subject headings. Terms identify the elements of information present in an item. Terms are stored as references to information.

There are three distinct levels of retrieval.

Reference Retrieval results when a search provides a list of item numbers or titles as the result. The inquirer gets the items and finds the answers.

Record Retrieval results when a search provides the documents or articles themselves as the results. The inquirer looks through the records for the answers.

Information Retrieval results when a search provides information as the direct answer to the question. The inquirer reviews the material to see if it satisfies his request. The same would be true for data retrieval.

Although there is much talk about information retrieval, in most cases it is reference or item retrieval.

BALANCE THE INPUT AND OUTPUT EFFORT

The usual approach is to spend a minimum of effort on organizing an index. When items are needed, the work begins. Long, frustrating, sequential searches are made through the index looking for specific items to fill an immediate need. Study the economics of the situation. Look at both input and output. Pause for a minute and analyze how large your index is and how much it is expected to grow each year. Assume you have 1000 items which can be filed in chronological order. Each time material is needed it might be necessary to search these 1000 documents by going through the index serially from the first to the last. If many searches are made, a comprehensive index to the collection may be justified. However, if a search is made only once a year, or less, the cost of installing a detailed system could probably not be

justified. Therefore, consider how many items are in the file, how many more will be coming into the collection, and how often these will be searched.

EVALUATE SINGLE-ENTRY AND MULTIPLE-ENTRY FILES

Before you store, consider how you will search. An insurance company keeps its records by account. All records on Charles Walker are kept in one file. The request usually is, "Bring me the file on Charles Walker." Since only a single-entry to the file is required, it can be kept in an ordered arrangement by a unique name. The same is true in a store that keeps its credit accounts by customers. Here files are kept in single-entry arrangement since this is the way the file will be used.

A single-entry system can be used in chemistry for a file on physical properties of chemical compounds, if the only questions asked of the file are to supply the physical properties of specific compounds. In this case, the most logical filing order is by compound name. In science and technology, however, this last situation does not usually occur. After a period of time of filing physical properties under the names of chemical compounds, along comes the problem in which someone needs to know the names of all chemical compounds with a certain physical property value. Now the file must be approached from a new viewpoint; there is a need to know both the physical properties of a given compound, and all compounds with a given physical property. The need for multiple entry to collections is common in science and technology and most other areas. A body of knowledge is usually approached from more than one direction. When an engineer requires the test results from a single instrument for a certain event, retrieval may be difficult if the results were filed only under the event name.

Early in your work decide whether your index can be managed with a single-entry approach or whether it requires multiple-entry search flexibility.

DESCRIBE ITEMS FULLY

If you are going to have a single-entry system, pick the proper entry characteristic and set up your index. More likely you will need multiple entry and more effort is required.

Traditional systems describe an item by perhaps one to four terms. This provides a limited number of ways of finding an item. More often it is desirable and necessary to index an item by perhaps ten, fifteen, or more terms in order to provide the proper degree of flexibility in finding what you want quickly and economically.

Full description of the item is needed because most collections require multiple entry. If you are going to be able to answer a variety of questions, then you need to provide for this search flexibility. This is done at input by describing each item fully, by indexing the item in anticipation of all the probable questions. The key questions to ask yourself are these: "In what ways am I apt to want this item in the future? What questions am I apt to ask for which I would like this returned?" Thinking about future use will help you describe the item properly so that you can find it easily later.

CONTROL THE VOCABULARY

The language for an information system is one of the most important features, yet it is often overlooked. Terms must be selected which bridge the communications gap from the language used by the originator of the item, to the person who stores it away, and then later, to the terms used in searching the index. The terms must be organized to take care of cross references and to provide consistency of input and search. However, the language must be flexible enough so that modifications and extensions can be handled easily. A modest amount of vocabulary control can assure that information is not lost because ideas are referenced in one way and searched for in another. Before setting up a system think about the concepts which cover the area of interest and reflect on the kinds of questions that are apt to be asked of the collection. Some thought at this point and some setting down of these terms will be a major assist in organizing the vocabulary for efficient operation.

KNOW THE SUBJECT

The best qualified person to describe a collection is one who is intimately familiar with the area covered. The organization of a system is done best by the person who will be using it. Finally, the best system will be the one installed and operated by someone knowledgeable in the subject area covered. Most efficient information systems in chemistry, for example, are operated by chemists. Often a system is turned over for indexing to a secretary or a clerk who may not have sufficient subject competence to describe an item well enough so that questions can be handled effectively.

SELECT APPROPRIATE INDEX SCHEME

Some type of index scheme is needed to provide the multiple entry. The FMP has two directories of its own; the main directory (disc directory) which contains information

about all the area available to the FMP and the local directory (file directory) which contains information on each file located on the disc or "file drawer." A supplementary index should be formed as a master list for all files and records, their security codes, and for cross referencing purposes.

Remember,

> "Those who cannot remember the past are commdemned to repeat it." –*Santayana.*

APPENDIX I
JOBFIL ORGANIZATION

JOBFIL contains information on:

- Lock flags
- Spool files
- Pending jobs

The file is a type 2 file with 16-word records. Records 1 through 16 contain a list of pending jobs and their priorities. The format is as follows:

Word 0 is the RN lock flag.

Word 1 is the number of entries.

Word 2 is the priority of pending job described in Record 19.

Word 3 is the priority of pending job described in Record 20.

. . . Priorities . . .

Word 255

A priority of 0 indicates no pending job exists in the record number indicated.

Records 17 and 18 hold information on size of JOBFIL, spool file control, and location flags for spool files. Records 19 and up are descriptor blocks for pending and active jobs.

Record 17

Word 0: RN lock flag

Word 1: # records in JOBFIL

Word 2: # of spool files

Word 3: Size of spool files

Words 4-8: In use flags for spool files – 1 bit per file, total of 80 bits. Bits are associated with the file by position – bit 0 of word 4 being the lowest number and bit 15 of word 8 the highest.

Words 9-12: Not currently used.

Word 13: JOBFIL record number of the job currently being input.

Word 14: Wait – RN.

Word 15: Shut-down flag.

Record 18

Words 0-16: Location flags for the spool files as follows:

A. Number of files/first file
B. Disc label

Two words are required per entry, total of 8 possible devices.

Record 19 and up

Each record is used for one JOB.

Word 0: Job priority (–1 => NO JOB)

Word 1: Job number (system assignment)

Word 2: Job status

Word 3-5: Job location – 3 word file name or LU.

Word 6: Disc ID of JOb location

Words 7-9: Job name

Word 10: Spool priority

Word 11-15: Spool usage flags – a bit set indicates job owns the spool.

APPENDIX J
SPLCON ORGANIZATION

SPLCON is a type 2 file with 16-word records. It contains information on all active spool files and pending outspools, outspool LUs, and available spool LUs.

Record 1

Word 0	RN lock flag
Word 1	# of spool control records
Word 2	# of outspool LU's
Word 3	record # of first spool control record
Word 4	maximum # pending spools
Word 5	class ID used by outspool program
Word 6-15	outspool LU list

Record 2

Used to keep track of the availability of spool control records in SPLCON. Each bit in each word corresponds to a record in SPLCON which can be used for spool file description.

Records 3-8

Word 0	Shutdown flag
Word 1	Wait – RN

Records 9-16 (one sector)

This is a queue on one outspool LU. Unused positions (entries) contain a zero.

Word 0	outspool LU#	
Word 1	# of entries on the queue	
Word 2	SPLCON record # of file	} pairs entries.
Word 3	spool priority	
.	.	
.	.	
.	.	
.	.	
.	.	
	.	
	.	
	.	
Word 127		

Additional sets of 8 records each are used for each outspool LU assigned by the user during GASP initialization.

Spool Control Records (immediately follow the outspool queue)

These records are used to describe the spool files currently in use, and the format corresponds to the setup buffer for SMP.

NOTE

Possible values for spool status (word 10) are:

A	file is being outspooled
W	file is waiting for the device
H	file is being held (by operator)
AH	file is active, but it is being held

Word 1 will be used to hold the value of the spool LU#.

APPENDIX K
SPOOL EQT

EQT1-EQT8	same as standard
EQT9	temporary
EQT10	temporary
EQT11	disposition flags
	bit 0: control word read/written
	bit 1: request initiated
	bit 2: request length in characters sw
	bit 3: spool pool file
	bit 4: standard file if set
	bit 5: not currently used
	bit 6: not currently used
	bits 8-7: 00 write and read 01 read only 10 write only
	bits 9, 10, 11: transfer vector for EXTND and time-out returns ϕ – standard read/write loop 1 – wait for EXTND to start SPOUT 2 – wait for backspace resource 3 – wait in backspace routine 4 – wait for write EXTND 5 – wait for read EXTND 6 – wait in backspace routine
	bit 12: holding I/O on this LU
	bit 13: batch input check failed once or EOF condition occurred
	bit 14: inside record length read/written (header)
	bit 15: control word read/written (header)
EQT12	# of words in EQT extension, BSREC entry point save
EQT13	pointer to EQT extension
EQT14-EQT15	same as standard
EQT16	backspace temporary
EQT17	backspace temporary
EQT18	current track/sector
EQT19	current offset
EQT20	extension #
EQT21	beginning track/sector in the file
EQT22	temporary (counter)
EQT23	temporary (packing buffer address)
EQT24	temporary (users buffer pointer)
EQT25	beginning track/sector in this extent
EQT26	number of sectors in each file extent
EQT27	TR/LU } directory address of master entry
EQT28	offset/sector } directory address of master entry
EQT29	ID segment of program requesting batch checking, or file counter for SPOUT, or 0
EQT30	number of sectors/track
EQT31	record count
EQT32	saved class word 1 } for use by SPOUT
EQT33	saved class word 2 } for use by SPOUT

APPENDIX L
BATCH PROCESSING PROCEDURES

INTRODUCTION

Batch processing (as described in Section I) can be accomplished either with or without spooling. Operation in the Non-Spool Mode uses the file management capabilities of the File Manager (FMGR) to read in commands, source language programs, and data comprising the job stream and to execute them. Operation in the Spooling Mode uses the input spooling capabilities of the Batch-Spool Monitor (FMP plus SM) to read the job stream into an input job file. As soon as at least one job has been read in, the spooling system automatically turns on FMGR to execute the job stream, using output spooling to keep throughput at a high level.

RUNNING BATCH JOBS WITHOUT SPOOLING

Job batching may be done without spooling. Although this does not increase throughput, it still offers the advantages of non-interactive program development and use of pre-defined job procedures.

In subsequent discussions, examples will be provided that use job control commands. These commands will be described as required; however, no attempt is made to give the complete definition of each command. Complete command descriptions are provided in Section III, FMGR Commands.

The job control (JOb) command, which has the general form,

:JO [,*name* [*:hr:min:sec*] [,*job priority*
[,*spool priority*, [,*NSpool*]]]]

is used to identify the programmer, length of time the job is to run, job priority, spool priority and whether or not automatic spooling is desired.

The :EOjob command terminates the job.

The run command which has the general form,

:RU, *namr* [,*parameters*]

is used to cause scheduling and passing of parameters to the named program. If execution of the command does not result in finding an existing ID segment for the program, a file by that name will be searched for. If a file is found and if it is a valid type 6 file, the program will be provided an ID segment and run. An interesting point to note is that *namr* can be a transfer file, in which case :RU acts as a :TR command.

As an example, a user wishes to assemble, load, and execute a program. Program source and test data are on the same device as the job control commands. The card reader is the system input device (LU5).

```
:JO,HAL
:LG,1
:RU,ASMB, 99
```

```
SOURCE CODE
FOR PROGRAM
PROGX
```

```
:RU,LOADR, 99
:RU,PROGX
```

```
DATA
```

```
:EO
```

The situation becomes more complex if the job must reference devices other than standard system units or must load subroutines from other than the relocatable library.

In the following discussion, the :LU command is used. Fundamentally it is used to make a logical unit logically equivalent to a file and to equate logical units. In non-spooled operation, the :LU command would most likely be used in the following form:

:LU,*lu*,*namr*

Lu is a logical unit reference used in the program and *namr* is an actual logical unit known to the system.

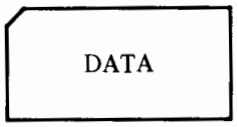
Assume that the user must compile a program whose source is in a disc file named SFILE. This program in turn calls a subroutine which is stored on a file named SUBRF. It also calls another subroutine named SUBX whose relocatable is to be submitted on paper tape to the system input unit. Additionally, the program reads data from a magnetic tape unit that it refers to as LU 10. Unfortunately, the system is configured so that the magnetic tape drive is LU 8.

The job would be set up as follows:

```
:JO,HAL
:LG,1
:LU,10,8
:MS,SFILE
:RU,ASMB,2,99
:MR,SUBRF
:MR,5
```

RELOCATABLE TAPE OF SUBX

```
:RU,LOADR,99
:RU,PROGY
```




```
:EO
```

It should be apparent that many jobs are basically similar in command structure; that is they use the same control commands in much the same order. Only the names of the modules, files and I/O devices change. Consider how convenient it would be if a user could request a standard compile and load-and-go job by simply requesting that control be transferred to a file that contains all the commands required for such a procedure. This procedure could be termed a "macro" procedure. Of course the user must pass the actual names of modules, files and I/O devices used to the macro procedure. The macro procedure must be written to reference generalized and undefined parameters instead of actual files, programs or I/O devices. Such parameters are termed "global parameters". Their real values will be provided when control is transferred to the procedure.

The means of transferring control to the procedure and passing parameters to it requires a special form of the transfer command:

:TR,*namr*,*P*1,*P*2,*P*3. . . . *P*9

The parameters *P*1 through *P*9 are used to define global parameters. Global parameters have specific names as follows:

1G, 2G. . . .9G

These are user defined according to the actual value of P1 through P9. Obviously there is a one-to-one correspondence between the position of an actual parameter in the parameter list of the :TR command and the ordinal in the name of the global parameter. The command

:TR,*namr*,*P*1,*P*2,*P*3

sets globals 1G, 2G, 3G to the values specified by *P1*,*P2* and *P3*.

For example, suppose the programmer wishes to transfer control to a job command file that will perform a simple assembly with input from a file named INFIL. The transfer command would be:

:TR,ASMBF,INFIL

The complete setup would be:

Job Deck:

```
:JO,HAL
:TR,ASMBF,INFIL
:EOJ
<NEXT JOB>
```

Contents of ASMBF:

```
:MS,1G
:RU,ASMB,2
```

It is very important to note the use of the MS command to place the file represented by 1G in logical source tracks. The RU command cannot reference the global 1G directly, as it represents a three-word ASCII array containing the file name. The RU command is implemented by the ON,ASMB command, which requires an integer logical unit number for the input device.

Nine global parameters have specific definitions. Two of them are discussed here:

0G This always implies the file from which the job commands were read.

10G Essentially this global is set to contain the name of the last program loaded by the RTE relocating loader. It can then be referenced by a subsequent :RU command. If the program cannot be loaded, 10G will be ASCII null.

To understand the use of the global 10G in relation to the RTE loader, consider the problem of building a generalized procedure to assemble, load and run a program from load-and-go. The :RU command requires the name of the program to execute:

:RU,PROGN

It is not desirable to hard code the name of a user program into a generalized procedure. However, it is possible to reference a global parameter in the run command.

Job Deck:	Contents of FORTLG:
:JO,HAL	:LG,1
:TR,FORTLG,PROGN	:MS,1G
:EOJ	:RU,FTN4,2,99
<NEXT JOB>	:RU,LOADR,99
	:RU,10G

However, the global, 10G, permits even more generalization than using a user-defined global. Remember that 10G contains either the name of the program loaded or an ASCII null, the user can perform error checking as shown in the following example.

Three new commands are used in the example:

:IF,P1,	EQ ,P2, n NE LT GT GE LE	If the relationship between P1 and P2 is true, skip n following job commands.
:AN, message		The message is annotated on the job list output.
:AB		Aborts the job.

The IF command obviously causes conditional execution of job control statements. If 10G had been set to ASCII null by the loader, indicating an unsuccessful load, the user aborts the job before attempting program execution. The relational operator "GT" was used because an ASCII program name has greater numerical value than "!!!!!!".

Job Deck:	Contents of FORTLG:
:JO,HAL	:LG,1
:TR,FORTLG,PROGN	:MS,1G
:EO	:RU,FTN4,2,99
<NEXT JOB>	:RU,LOADR,99
	:IF,10G,GT,!!!!!!,2
	:AN, ** NO LOAD **
	:AB
	:RU,10G

Another problem could arise that the user could recover from within his job control commands. What if the program we are trying to load has the same name as one already in the system. Remember that the RTE loader renames the program it is trying to load by changing the first two characters of the name to "..".

It will be necessary to check to see if the name of the program was changed by the loader. If so, the user may wish to change the PROGRAM or NAM statement in his program at a later date.

In the next example, we consider the possibility of duplicate program names. Also, let us assume that program input can come from either the card reader or photoreader and that the job structure must determine which.

Two new commands will be used in the example:

:SE [,[P1] [,[P2] . . .[,P9]] . . .]

Sets the globals 1G through 9G to the values defined by P1 through P9. Notice that the :SE command is similar to the :TR command with global setting capability. Actually it is the set function that is used by the :TR command processor.

:CA, global, operand, + − / * Or Xor And ,operand

Performs the indicated operation on the two operands and places the resultant value in the global specified.

The job command structure would be:

Input from Job input device:

```
:JO,TEST
:TR,FORTLG,input,plist      input = location (LU) of source (not a file name)
                            plist = up to five parameters to be passed to program
:EO
```

Contents of FORTLG:

```
:LG,1

:CA,7,0G                    Assume source is on job input unit
:IF,0G,EQ,1G,1              Then check if that is so
:CA,7,1G                    No? Then set 7G to source input unit
:RU,FTN4,7G,99              Schedule compiler
:RU,LOADR,99                Schedule loader
:CA,7,10G,AND,-1            Mask off last three characters of name
:CA,1,. . ,AND,-1           Put MASKED . . in 1G
:IF,1G,NE,7G,1              Check if loader renamed program
:AN,**DUP NAME**            Message to printer
:IF,10G,GT,!!!!!!,2         Did it load?
:AN,**NO LOAD**             No, send message to operator
:AB                         Abort job
:RU,10G,2G,3G,4G,5G,6G      Yes, run program and print plist
```

At this point, five other globals may be of interest to the RTE programmer; these are 1P through 5P. Unlike the globals 0G through 10G, which can contain a three-word ASCII parameter, these represent a one-word numeric value. They are used to hold up to five parameters which can be passed back to the Batch Monitor through a program's use of the subroutine PRTN.

The previously discussed global 10G, is actually used to reference parameters 1P, 2P and 3P as if they were a single three-word ASCII parameter. The loader passes back to the FMP the name of a successfully loaded program and other FMGR commands may access this name by referencing 10G.

To demonstrate the use of the "P" globals, consider the case of a job with one program that returns five values in P1 through P5 that must be passed as schedule parameters to a second program:

```
:JO,TEST
:RU,PROG1
   < PR0G1 calls PRTN to return values in P1 through P5 >
:RU,PROG2,P1,P2,P3,P4,P5
:EO
```

RUNNING BATCH JOBS WITH SPOOLING

If the user wishes to activate the spool system he must schedule a module named JOB (the input spooler). Only one parameter is specified, the logical unit number of the device on which the batch jobs reside:

```
*ON,JOB,5
```

would direct JOB to the system input unit for the job stack. Jobs would then be read off that device and placed in spool files. Each job will have its own spool file. Jobs will then be run on a priority basis. If any job requires output spools they will be provided by the spool system, or the user may set up his own spools. Output spooling is automatically provided to the system list device (usually a line printer); however, the user may define his own spool file for printable output. Additionally, he could define output spool files for other slow devices. The user can, at his option, disable list device spooling for a job by using the "NS" parameter in the :JO statement.

All of the jobs discussed in the section entitled "Running Batch Jobs without Spooling" will run with the spool system active without modification to the command statements.

Consider the following job:

```
:JO,HAL
:LG,1
:RU,FTN4,99
```

SOURCE	Compiler expects source on LU5 and outputs list on LU6.

```
:RU,LOADR,99
:RU,10G
```

DATA	Program reads data from LU5.

```
:EO
```

Notice that something peculiar appears to take place. Although the job references logical unit 5 for input data and logical unit 6 for printable output, which are still known to the system as the card reader and the printer, the job actually reads data from a disc file and outputs data to a disc file. It would appear that logical units 5 and 6 have a different meaning for the program than they do for the system. This is essentially true because the system has internally transformed all program references to logical units 5 and 6 to references to logical unit numbers associated with system managed spool files.

The transformation is made within the system and no alteration is made in the program. Also the system remembers that the output spool is destined for the system list device when the file is unspooled.

Thus, if the job is spooled to a spool file known to the system as LU15, all compiler reads from LU5 are transformed to reads from LU15, which has become the job input "device". The logical unit numbers associated with the system-managed spool files are established at system generation. These logical units are represented by entries in the RTE Device Reference Table (DRT) and therefore have a corresponding Equipment Table (EQT) entry. These entries are managed by a program known as the Spool Monitor Driver. As the Spool Monitor Driver (DVS43) manages I/O tables in much the same manner as an actual device driver, it is placed in the RTE driver area at system generation even though actual data transmission to or from a disc file by DVS43 is done through a system I/O request.

Generally speaking, if the programmer is to batch jobs using spooling, he must:

1. Request that spools (other than standard input and list) be made available to him;
2. Define his own spool files if he wants to retain the spools as permanent, named files;
3. Tell the operating system what spool files the logical unit references in his program are associated with; and
4. Tell the operating system how to dispose of his output spools.

Consider the problem of a programmer that wishes to compile a program and retain the relocatable as a permanent file as well as obtaining a punched tape. At first glance it would appear that either the programmer must actually punch paper tape and later store it into a file using the ST command, or he must compile a load-and-go and then save the relocatable with a SA,LG,namr command.

There is an alternative. The programmer could spool his punch output to a file that he defines. To accomplish this he must declare to the system that, for this compilation, all output to logical unit 4 is to go to his file, and that the file is to be saved. This is done with the :LU command, whose complete generalized form is:

:LU,*lu1*,*namr*,*attribute*,*lu2*,*priority*

where

lu1 = logical unit referenced in program

namr = name of spool file

attribute = <discussed below>

lu2 = logical unit of actual output device

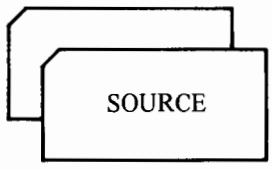
priority = determines relative order in which spool files are dumped. Has values 0 – 9999.

For the purposes of this example, the :LU command would be:

:LU,4,%RELO,WRSAST,104B

This command states that output to logical unit 4 is to be spooled to a file named %RELO and that it is a write only (WR) file in standard (ST) format that is to be saved (SA). The last parameter indicates that the relocatable is to be passed to the system outspool program and punched on the system punch device (LU4; or in binary, 104B). The "WR" attribute must be specified because the system defaults to read-only. Similarly, the "SA" must be specified because the system defaults to releasing the file at the end of the job – even though it was created by the user. The job structure of this example is:

```
:JO,HAL
:LU,4,%RELO,WRSAST,104B
:RU,RTN4,5,6,4
```




```
:EO
```

The "ST" must be specified since the default format contains record headers needed to propagate spacing control requests on a line printer or necessary commands to other devices. Headers are not needed here.

List spools contain special header words in front of records that contain information of interest to the system output spool program only. When a user defines an output list file in an :LU command, he must indicate with an attribute whether or not the file will contain these special records (default is headers).

To indicate that a file is to be dumped to an external device, an outspool LU is supplied in the command. For example,

:LU,6,&LIST,WRSA,6

specifies that &LIST is to actually be spooled out to the system list device, and it contains header records. However, the command

:LU,6,&ASCII,WRSAST

specifies that &ASCII is not to be spooled out, but is saved as a standard RTE file.

From the discussion in the last paragraph it becomes apparent that logical unit to file equivalences need not result in the actual outspooling of the file to a punch or printer. Technically speaking, outspooling is the process of taking the punch or print spool file and outputting it on the actual punch or print device. However, it is possible to create and save spool files which, in reality, are not actually spooled out.

Another non-spool aspect of the :LU command is that a programmer can reference arbitrary logical unit numbers in his program and later define them at run time with the :LU command. This affords not only the device independence that is characteristic of logical unit references, but it allows for a certain measure of independence from logical unit numbers that have specific system definitions (i.e., the list device in RTE is always LU6).

A programmer must develop application programs long before a system is delivered and actual logical unit number assignments are made. He might decide to arbitrarily refer to a magnetic tape drive as LU17:

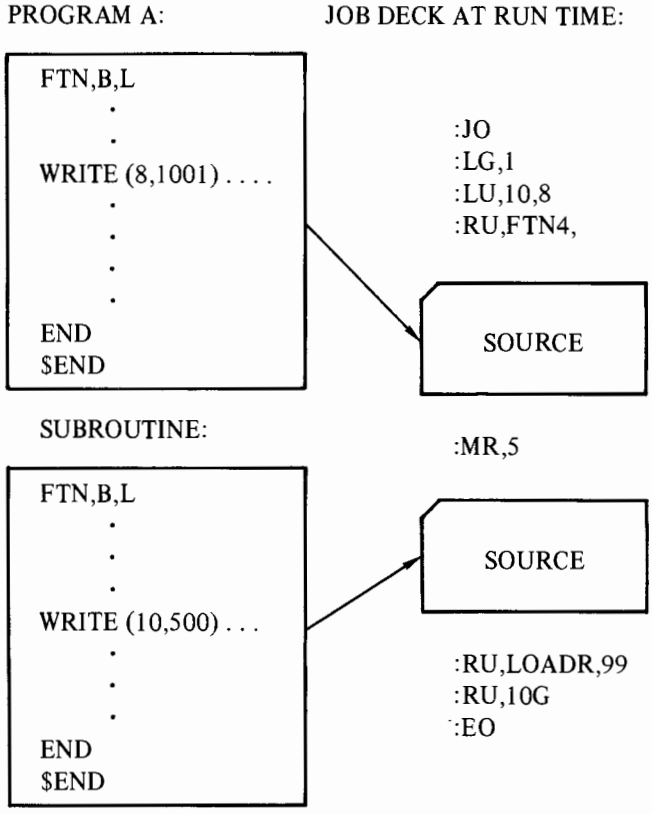
WRITE(17,1ØØ1) . . . . . .

When it comes time to actually run the program, the system might have been generated with the magnetic tape drive as LU10. The programmer would then insert the following :LU command in his job deck:

:LU,17,1Ø

Also, two or more programmers can refer to the same phsycial unit or file by different logical unit numbers; the equivalences being made at run time. Thus, one programmer does not have to coordinate his logical unit references with another programmer to insure use of the same number. Old, but still usable, software would not have to have logical unit numbers changed, nor would the writer of a program destined to run with the old software be constrained to use the same logical unit references appearing in the old code.

For example, program A below refers to the tape drive as LU8, but its subroutine, which was written for a previous configuration refers to the tape drive as LU10. The problem is solved by running the program with an :LU command set up to equate LU8 to LU1Ø:



Macro procedures, which are discussed on page L-2 can effectively be used with spooling. These procedures will require the programmer to utilize globals in his :LU command as well as in the control statements already discussed. Of particular use in this regard will be the SEt

command, which was described on page L-3. Additionally two globals "OS: and 1S may be used. They have the following definitions.

0S This contains the logical unit number of the spool used in the last spool set-up. At job start this is usually the list spool.

1S This contains the actual name of the last spool file setup.

The following examples demonstrate the use of globals and macro procedures involving spooling.

Example 1. This example demonstrates the use of a macro-procedure to perform standard compilations and execution.

Transfer command definition:

```
· FORTLG, in, bout, lib, plist
```

Abbreviated form of TR command.

Name of macro procedure file.

File name or logical unit number of the source program input unit; is automatically equated to global parameter 1G.

Logical unit number of destination device for punch output; is automatically equated to global parameter 2G (optional).

File name or logical unit number of device on which a set of needed library subroutines resides; is automatically equated to global parameter 3G (optional).

List of up to five parameters, separated by commas, that can be passed to the program for use when it is run; the parameters are automatically equated to global parameters 4G through 8G, in the listed sequence (optional).

The setup of the FORTLG macro procedures file is shown below. The first two commands get the Batch-Spool Monitor's attention and tell it to set up the FORTLG file.

```
*ON,FMGR          Operator turns on access to file manager commands for entry of FORTLG file
:ST,1,FORTLG,AS   Stores keyboard input from LU1 into FORTLG file, in ASCII format
```

The remaining commands, below, make up the FORTLG macro procedure file.

```
:LG,1             Requests one load-and-go track for use by FORTLG operation
:CA,9,ØG          Same current input location
:IF,1G,EQ,ØG,2    If input from job file, skip
:LU,15,1G         Else set up input to FTN 1
:CA,9,15          Set 9G for RUN command
:LU,4,2G,WRSAST   Sets up write-only file, which will be saved
:RU,FTN4,9G,6,4,99  Runs compiler
:IF,3G,EQ         If no library, skips next command
:MR,3G            Moves library to load-and-go track
:RU,LOADR,99      Runs loader to load compiled version of program along with needed library routines
:IF,1ØG,GT,!!!!!!,3  Skips next three commands if load has been successful (10G contains a file name)
:AN, ***LOADR ABORTED***
:AN, ****JOB FLUSHED****   Outputs message to job listing and aborts the job currently using FORTLG if load has been unsuccessful
:AB
:RU,1ØG,4G,5G,6G,7G,8G   Runs program
:OF,1ØG           Flush program from system (name is contained in global parameter 10G)
:TR               Terminates FORTLG and returns to job stream
```

FORTLG would be used as follows:

```
*ON,JOB,5         Operator turns on job entry, input via card reader (LU5)
:JO,PROGEX        Identifies job by name (PROGEX)
::FORTRLG,PROG,4,LIB   Transfers control to FORTLG, acting upon PROG, outputting to punch (LU4)
RTN,L
----------------
----------------
----------------   Body of program; blank card is translated as end of file
:EOJ,PROGEX       Terminates job
```

Example 2

```
ON,JOB,5 ---------- Operator turns on job entry, input via card reader (LU5)
:JO,JOBNAM ---------- Identifies job by name (JOBNAM)
:RU,EDITR,5 ---------- Runs editor to correct program in file FILNAM, edit input via card reader
----------
----------  ---------- Edit commands
----------
----------
:LG,1 ---------- Requests one load-and-go track for compilation
:LU,9,FILNAM ---------- Equates logical unit number 9 to source of input (FILNAM)
:LU,4,,WRST,4 ---------- Sets up file, which will not be saved for punch output (illegal command)
:RU,RTN4,9,6,4,99 ---------- Runs compiler, with punching of relocatable output tape
:MR,LIB ---------- Moves library to load-and-go track
:RU,LOADR,99 ---------- Runs loader to load compiled version of program along with needed library routines
:RU,1ØG,4G,5G,6G,7G,8G ---------- Runs program
:OF,1ØG ---------- Turns off program
:EOJ,JOBNAM ---------- Terminates job
```

Note that the program is corrected or updated by running the editor as part of the batch job. Also note that a user generated library has been moved to the load-and-go tracks so that it may automatically be scanned when loader searches for subroutines to satisfy main program external references.

Example 3

Card-to-Mag Tape Dump

In preparation for job processing, it is often helpful in backlogging a large volume of work to transfer input from mark-sense or punched cards, which are a low-density storage medium, to magnetic tape, which gives far higher storage density. It would be useful to maintain a command file that can perform this function whenever required. Consider a command file named "UTILITY" constructed as follows:

```
:CN ———————————————— Rewinds tape on magnetic tape unit
:DU,5,8,AS,4 ———————— Dumps input from card reader (LU5) to mag tape (LU8) in ASCII format to file number 4
:CN ———————————————— Rewinds tape on magnetic tape unit
:EX ———————————————— Exits file manager
```

This procedure may be executed under spooling in the following manner.
Turn on JOB and enter the following command:

:XEQ, NAME [,*p*<]

where NAMR is the name of the file containing the job and pr can be used to specify a job priority. In this case, UTILITY is NAMR and its contents will be placed in a spool file and logged as a job by the system. If this procedure is executed as described :JO and :60 cards must be added to the file UTILITY.

APPENDIX M
CARTRIDGE FORMATTING

During initial moving head system generation, the operator is given the option of formatting all subchannels. The following procedure may be used to format new packs, or packs that may not have been mounted during the initial generation. (Refer to the RTE manual for more details on cartridge formatting.)

1. Load the moving head RTGEN using the BBL.
2. Load and configure a TTY SIO driver.
3. Start RTGEN at octal address 100; clear the switch register and push RUN.
4. Answer the moving-head disc cartridge channel question as in a normal generation.
5. a. Define the areas on the respective subchannels (number of tracks and starting track number) which are to be initialized.
   b. Define a subchannel higher than any defined in "a" that is not mounted (i.e., either the unit is not present or it is unloaded). For example, if the system has one drive (i.e., subchannels 0 and 1), then define some tracks on subchannel 2. Then enter the /E.

CAUTION

Do not define any cartridge areas in step 5.a that have existing systems or data that is to be preserved on them. Steps 6 through 9 are important to prevent RTGEN from automatically formatting a cartridge area containing data.

6. Assign the dummy subchannel as the system subchannel.
7. Assign the dummy subchannel as the scratch data.
8. Do not assign an "AUX DISC" (i.e., answer "NO").
9. Set scratch origin to 0.
10. The number of 128-word sectors per track is 48 for the HP 7900/7901 and 24 for the HP 2870.
11. Answer the rest of the questions as per a normal generation.
12. When RTGEN asks:

    INITIALIZE SUBCHNL:
    X?

    answer YES for those channels to be initialized.
13. After the last subchannel is initialized, RTGEN will try to initialize the dummy subchannel (because it is the system subchannel) and will find it down. This causes message:

    READY DISC AND PRESS RUN

At this time the cartridges are formatted and the procedure is complete.

INDEX

A

B

C

D

D (Continued)

E

INDEX (Continued)

INDEX (Continued)

INDEX (Continued)

S

S (Continued)

T

U

W

X

HEWLETT hp PACKARD

Manual Part No. 92002-93001
Microfiche Part No. 92002-93002

Printed: FEB 1975
Printed in U.S.A.