HEWLETT PACKARD

RTE-6/VM

CI User's Manual

Data Systems Division
11000 Wolfe Road
Cupertino, CA 95014-9974

Part No. 92084-90036
E0185

Printed in U.S.A. January 1985

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Printing History

The Printing History below identifies the Edition of this Manual and any Updates that are included. Periodically, Update packages are distributed which contain replacement pages to be merged into the manual, including an updated copy of this Printing History page. Also, the update may contain write-in instructions.

Each reprinting of this manual will incorporate all past Updates, however, no new information will be added. Thus, the reprinted copy will be identical in content to prior printings of the same edition with its user-inserted update information. New editions of this manual will contain new information, as well as all Updates.

To determine what manual edition and update is compatible with your current software revision code, refer to the appropriate Software Numbering Catalog, Software Product Catalog, or Diagnostic Configurator Manual.

First Edition . . . . . Dec 1983. . . . . .
Second Edition. . . . . Jan 1985. . . . . .

Preface

This manual tells you how to use the RTE-6/VM Command Interpreter (CI) and the CI file system. It also describes the FMP subroutines and certain utilities used in the CI file system environment. Descriptions of other utilities and subroutines that can be used in the CI environment are given in manuals shown below.

EDIT/1000 User's Manual	92074-90001
Symbolic Debug/1000 User's Manual	92860-90001
Macro/1000 Reference Manual	92059-90001
RTE-6/VM LINK User's Manual	92084-90038
RTE-A & RTE-6/VM Relocatable Libraries Manual	92077-90037
RTE-6/VM Utilities Manual (TF)	92084-90007

There are four sections in this manual: an introduction section giving an overview of the CI file system environment, a section providing information for the display terminal users (Chapters 2 through 5), a section describing the FMP calls for programmers using the CI files, and a section describing the CI file system utilities. Error messages and error codes that may be encountered in the CI file system are given in Appendix A. Appendix B contains descriptions of exception conditions in the CI file environment.

This manual is for the first time users of RTE-6/VM and the CI file system. If you are such a user, go through the manual in the order in which the information is presented. However, if you are familiar with the FMGR file system and with the RTE-6/VM Operating System, skip to the chapter of interest.

Chapter 1 Introduces the CI file system and CI.

Chapter 2 Shows how to use the CI system commands.

Chapter 3 Shows how to use the CI file manipulating commands.

Chapter 4 Shows how to use the CI program control commands.

Chapter 5 Contains the CI command descriptions in alphabetical order.

Chapter 6 Describes the CI FMP subroutines.

Chapter 7 Describes the CI file system utilities.

Appendix A Contains the error codes and error messages.

Appendix B Explains the unusual conditions that may occur.

Appendix C Describes the FMP call conversion from the FMGR to the CI file system.

Table of Contents

Chapter 4
Controlling Programs

Chapter 5
CI Command Descriptions

Chapter 6
FMP Calls

Appendix A
Error Messages and Codes

Appendix B
Exception Condition Handling

Appendix C
Converting FMGR File Calls

Illustrations

Tables

Chapter 1
CI File System Introduction

This manual is the primary reference source for display terminal users and programmers in a file system environment managed by the Command Interpreter (CI) program. The CI file system allows efficient and more logical use of disc space. It also facilitates organization of files with a hierarchical directory structure.

This chapter provides a brief comparison between the CI and FMGR file systems. It describes the features of the CI program and introduces the basic characteristics of CI files. Also included in this chapter are conventions and terms used throughout this manual.

The RTE-6/VM File Systems

Normal interface to the operating system is through one of two programs, FMGR or CI. Each of these programs provides two basic capabilities:

1. Interface to the operating system. This includes obtaining system status, modifying system parameters, running other programs, and controlling input/output devices.

2. Interface to the file system. This includes creating, purging, and transferring files and obtaining status information.

In system interfacing, the FMGR and CI programs are basically identical except for a few differences in command syntax or function. For example, the FMGR system break mode commands can be entered in CI without the SY prefix. File system interface is different between the two programs. An overview of the CI file system is given in this chapter. Refer to the RTE-6/VM Terminal User's Reference for a full description of the FMGR file system.

The FMGR file system divides a disc into fixed-size cartridges that are identified with either negative LU numbers or positive cartridge reference numbers (CRN). The CRN can also be a two-character string. Each cartridge has a cartridge directory containing pertinent information on all files stored on that cartridge.

The CI file system divides the disc into large areas of free blocks. These areas are identified by LU numbers and are called disc volumes. Files in each disc volume are managed by directories and subdirectories which maintain information on what files exist and where they are located on the disc. There is one directory in each disc volume that contains the names of all unique directories in the disc volume. This directory is called the root directory and the directories included in the root directory are called global directories. Directories that are included in other directories are called subdirectories.

Comparison of the characteristics of the two file systems is given in Table 1-1.

During user accounts installation, either FMGR or CI can be specified as the primary program to be used for accessing all relevant system functions. The primary program is defined as the program scheduled at log on. Each program can also be selected to run automatically at log on. The primary program setup procedure is described in the RTE-6/VM System Manager's Reference Manual.

Table 1-1. RTE-6/VM File Systems Comparison

	FMGR File System	CI File System
File name	1-6 characters	1-16 characters
Cartridge/ directory	1-2 characters or numeric cartridge names	1-16 characters in directory names
File Security	Security code used for file protection	Protection based on dir. (and file) ownership
File Types	Defines the structure of the files	File type extensions describe the contents of the files
File Mask	None	Mask qualifier and special characters in file name
File Size	Extendible (except type 6)	Extendible (except type 6)
Time Stamps	None	Create, access, update times handled by the file system
Sub-directories	None	Subdirectories within directories and other subdirectories
File Recovery	None	Operator recoverable immediately after purge
Spooling	Can be done interactively and programmatically	Through FMGR
Incremental Backup	None	Done in conjunction with TF utility

The Command Interpreter

The Command Interpreter (CI) is a friendly user interface program. It may be scheduled at log on or from FMGR or other user programs in the session environment. When executed, CI displays the prompt

```
CI.nn>
```

and is ready to accept commands. The actual prompt displayed is based on the name of the CI clone being run. Normally, nn is the session LU number. The commands and the required parameters can be entered in the command string in either uppercase or lowercase letters. Blank characters and commas can be used as the command string delimiters. Throughout this manual, the blank is used as the command string delimiter. However, if there are parameters omitted from the middle of a string, commas must be used as place holders. Following are examples of CI command entries. Note that the user entries are underlined.

```
CI.65> edit                    (runs the Editor program)

CI.65> WH                      (displays system status)

CI.65> co report.txt data      (copies REPORT.TXT into new file DATA)

CI.65> dl /jones/              (displays all files in directory JONES)

CI.65> bl,,700                 (modify upper buffer limit)
```

Interactive operations available through CI include the following functions:

- System Status Check
- System Control
- File Manipulation
- Program Control

The Command Interpreter provides commands that start and stop programs and commands that change the way a program executes. Commands are available for creation of directories and subdirectories that are used in file management. Files can be created, copied, stored in a directory or subdirectory, purged, and renamed. You can control access to files and manipulate a group of files with a single command using the file mask feature. Time stamps are maintained for all files to keep track of date of creation, access, and updates. Various system information can be displayed on the terminal screen, e.g., program status, input/output device status, etc. System behavior can be controlled through the system commands. These features are described in Chapters 2 through 5.

Spooling is not available in CI. If you must use the spooling system, store your files on an FMGR disc cartridge (following the FMGR namr convention) and then run FMGR to use the spooling system. An alternative is to use a combination of CI and FMGR commands, described in the Spool Files section in Chapter 3.

CI provides another level of commands for the System Manager. In addition to all the interactive capabilities, the System Manager is allowed the following:

A. Set up the session to run in the CI environment.

B. Change the properties of any program. Applies to all program control commands.

C. Override file protection. Applies to all file manipulating commands.

D. Modify system attributes such as the system clock. Applies to all system control commands.

E. Re-initialize discs.

Certain commands are subject to the RTE-6/VM user capability level restrictions. If you do not have the proper capability level, entering any of these commands will result in a capability error. To change your capability level, see your System Manager. For more information on command capability levels, refer to the RTE-6/VM System Manager's Reference and the Terminal User's Reference manuals.

CI also provides special features designed for the convenience of the users: command stack, command files, program execution control, multiple commands entry, and quoting (string passing).

The command stack allows users to repeat commands without retyping or to modify commands before repeating them. A command stack file is used to save the stack contents for subsequent sessions. This method enables a truly distinct working session environment where a user can pick up where he left off in the prior session.

The command file (also known as the transfer file) is used to minimize user interface; it allows a series of commands to be entered from a file instead of being entered one by one at a terminal. Associated with the transfer file command are positional variables and program/command execution control features. Positional variables are used in transfer files and CI command strings for value passing. Conditional branching commands (IF-THEN-ELSE-FI and WHILE-DO-DONE) are also allowed in transfer files for use in controlling program or command execution. These features are particularly useful in program development.

CI allows multiple commands in a single entry and the use of quotes for value passing. These and the features mentioned above are further described in Chapter 2.

Introduction to CI Files

Files are identified by file names that can have up to 16 characters. Additional information is added to the file name to describe the associated properties. The file name includes a file type extension that further describes the type of information contained in the file. The file name and the associated attributes that identify a file are considered a file descriptor. Colons, dots, and slashes are delimiters in file descriptors. Following are the various forms of file specification.

```
proga                                (file name with blank file type extension)
prog.rel                             (file name)
userManualchap11.text                (file name)
progb.rel::directory                 (file descriptor)
/directory/progb.rel                 (file descriptor)
/directory/subdirectory/chapter6:::4:609   (file descriptor)
```

In the CI file system, files can be grouped under unique directories or subdirectories. Subdirectories can be nested within other subdirectories. Thus a hierarchical file structure can be established.

File protection is based on directory ownership. You become the owner of the directories or subdirectories created by you. Ownership can be changed by either the owner or the System Manager. The owner of a directory or subdirectory can restrict file access on that directory (or subdirectory) for other general users.

Time stamping of all files is automatic in the CI environment. The CI file system maintains three time stamps, time of creation, time of last access, and time of last update. The time stamps can be used to search, to access, and to purge files using the file mask feature.

CI files can be specified as a group, determined by a file mask. The file mask includes a field appended to the filename parameter in the file descriptor. This field is called the mask qualifier that determines the grouping of files. For example, you can use the mask qualifier in a file listing command (DL) to tell the file system to search everywhere in the system and not follow the hierarchical directory structure or to search for files created at a certain date.

If your system has the optional DS/1000 Distributed System, you can access files on other systems in the network using the CI DS transparency feature. Remote file access is described in Chapter 3.

The file name, file type extension, file descriptor, file type, directory, file mask, and other file properties are described in detail in Chapter 3.

Manual Conventions

The command syntax and other conventions used throughout this manual are described in the following paragraphs. Sample terminal displays include both user inputs and program prompts and messages. User inputs are underlined. Comments are given in parentheses. For example,

```
CI.65> dl /derick/casey/@.@      (List all files in subdirectory
                                  CASEY under directory DERICK)
```

The command syntax conventions are as follows.

Uppercase characters — Capital letters indicate the exact characters required. However, CI will accept lowercase input. For example, the command syntax for the ON command is:

```
ON[ prog[ NO parm*5]]
```

and the actual sample entry can be:

```
CI.65> on testprogram no a b c e g
```

[] — Square brackets are used to show optional parameters. If additional parameters follow an omitted parameter, commas must be used as place holders.

/ — Alternate choices are separated by a slash. For example, "dest file/lu" indicates that either a destination file or an LU number can be entered. Note that the slash is also used in the command string to designate directories and subdirectories.

, or blank — Delimiters between commands and parameters are commas or blanks. Blank spaces are used throughout this manual in all syntax strings.

lowercase and <> — Lowercase letters represent variables supplied by the user. In case of a long descriptive phrase, the variable may be enclosed in angle brackets for clarity, e.g., <directory name>.

There are certain terms used in all syntax strings that have standard meanings throughout this manual. The most common terms are described below.

prog — Program name; up to five characters can be used. Examples of program names:

```
A
PROGA
ADVEN
TIMER
```

lu — Logical unit number in the range of 0 to 255. It refers to a physical input/output (I/O) device. LU 1 is usually the user terminal, exceptions are noted throughout this manual, e.g., in the TO command description in Chapter 2. LU 0 is the "bit-bucket", a non-existent device to which unwanted data can be dumped.

file — File descriptor which includes parameters that describe various properties of the file, e.g., search path, file type, size, and record length. It can be accepted in any of the following formats:

```
Standard:  /dir/subdir/filename.typex.qual:::type:size:rlen

Combined:  subdir/filename.typex.qual::dir:type:size:rlen

FMGR:      filename:sc:crn:type:size:rlen
```

Refer to Chapters 3 and 6 for a full description of the file descriptor.

filename — A file descriptor parameter. In the CI file system, it can have up to 16 characters followed by the file type extension (typex) and the qualifier parameter (qual) with a period as the delimiter.

mask — Mask field. May be the wildcard characters in the filename parameter (- and @) or a mask qualifier appended to the typex parameter. Refer to Chapter 3 for details.

file/lu — Either a file descriptor or a logical unit number may be specified. A mask may be used in the file descriptor.

prog/file — Either a program name or a file descriptor may be specified.

parm — One parameter is allowed.

parm*2 : parm*n — Two to n parameters are allowed. Unspecified parameters may default to zero or zero-length strings depending on the application.

Chapter 2
System Commands

This chapter discusses system commands that you may need in the CI environment. These are system commands provided to display system status and help information, and to control certain system operations. A summary of these commands are given in Table 2-1. Commands that affect system processes and control system operations are reserved for the System Manager. These commands are listed in Table 2-2.

This chapter shows you how to use the CI system commands in obtaining system status information and controlling system operations. Usage of file manipulating and program control commands are given in Chapters 3 and 4, respectively. If you are familiar with FMGR and the RTE-6/VM Operating System, you need only to learn those commands specific to CI, for example, ECHO, RETURN, SET, etc. Then you can skip to Chapter 3 for the CI file system information.

Table 2-1. General User System Commands

Command	Task
? (or HE)	Display help summary
? command or HE command	Display command description
/[n]	Display command stack
/n/	Set command stack display size
BL	Display buffer limits
CN lu[function[parm*4]]	Control I/O device
ECHO[parm]	Display command parameters
EQ	Display EQT information
EX	Exit CI
OF prog[ID]	Terminate program
PR prog[priority]	Display/change program priority
RETURN[,return1-5[,return_s]]	Return value(s) and/or string

Table 2-1. General User System Commands (Continued)

Command	Task
SET[variable = string]	Define/display variable
SL	Display LU information
ST	Display program status
SZ	Display/modify program size
TI	Display system time
TM	Display formatted system time
TO	Display device time-out
TR file[parm*9]	Transfer to command file
UL label	Unlock shareable EMA partition
UNSET variable	Delete a variable
UP eqt	Up a device
UR	Release reserved partition
VS	Display VMA size
WH[parm] AL PA SM PR PL	System status report All programs Memory partitions
WHOSD DIRECTORY/lu	Display directory/volume status
WS	Display working set size

Table 2-2. System Manager Commands

Command	Task
AG[number/OF]	Modify partition aging
BL[lower[upper]]	Modify buffer limits
TM year date hour min sec	Modify system time
TO eqt[interval]	Sets device time-out
UL label	Unlock any shareable EMA partition
UP eqt	Up any device
UR partition	Release any reserved partition
VS prog[lastpg]	Modify VMA size
WS prog[wrksz]	Modify any working set size

Getting Help

The CI program provides an on-line help summary and a quick reference guide. The help summary, or a brief explanation of any command or item listed in the summary, can be displayed with the HELP command.

The help command can be entered as ?, ??, or HELP. This gives a list of the commands by their command mnemonics and other useful items such as a description of file mask, file descriptor, etc. For example, following is the response from a ? command:

```
CI.65> ?
Commands: (use ? <command> for help on <command>)
directory ::HELP
??          AG          AS          BL          BR
CI          CL          CN          CO          CR
CRDIR       CU          DC          DL          DN
ECHO        EQ          ERROR       EX          FOWN
FPACK       FREES       FVERI       GO          HE
IF          IN          IS          IT          LI
LINDX       LINK        LU          MACRO       MASK
MC          MERGE       MO          OF          ON
OWNER       PATH        PR          PRINT       PROT
PU          QU          RN          RP          RU
SET         SL          SS          ST          SZ
TI          TM          TO          TR          UL
UNPU        UNSET       UP          UR          VS
WD          WH          WHILE       WHOSD       WS
XQ

CI.65>
```

To get information about a command, enter the command name after the help command. For example, to get information about the OF command, entering "? of" will display a brief explanation of the command and command syntax.

```
CI.65> ? of
OF -- "Off" a program and optionally remove its ID segment

Usage: OF[ programName[ ID]]

 program Name has up to 5 characters; its session identifier is optional.
 ID is an optional keyword used to release a previously RP'd program's
 ID segment.
    :
    :
 More...('a' to abort)
```

In addition to commands and explanations, other information is available with the help command. This includes items such as file descriptor, file mask, and file type extension.

It is possible to add items to this list. The help command works by listing a file contained in a directory called HELP. You select the file it lists when you ask for help on a particular command. By entering the help command without any parameter, you are displaying the contents of a directory called HELP. By adding files to this directory, you can increase the number of items listed in the help summary.

Using the Command Stack

As command lines are entered at the terminal keyboard, they are saved in a stack for reference or reuse. The number of command entries in the stack varies depending on the length of the entries. A minimum of 25 entries will be saved; however, the average will be approximately 100 commands.

If the stack is full, the oldest commands in the stack are removed to make room for new commands. Duplicate commands are not saved in the stack. Commands entered from a command file are not saved in the stack. Command lines in the stack can be edited and reentered, or simply reentered without retyping.

Commands in the stack can be saved in a file. By default, a file called CI.STK on the working directory is used. (CI uses CI.STK on the working directory when you log on.) Another file can be created or selected to hold the command stack. Refer to the command stack and the WD command descriptions in Chapter 5 for details on manipulating the command stack file.

If you do not want to save your command stack in a file or do not want the file updated, set the predefined variable $SAVE_STACK to FALSE. Refer to the Predefined Variables section for details.

To display the command stack, enter a slash:

```
CI> /                          (A screenful of commands, defaulting
---Commands---                  to 20, is displayed. Refer to the
wd /mine/myprograms             /n/ command description in Chapter 5
dl                              for changing default.)
     .
     .
     .
li syslog
pu syslog
ru print report.txt
```

```
co 8 report1.sale
co 8 report2.sale
co 8 report3.sale
co 8 report4.sale
co 8 report5.sale
co report5.sale 4
prot report5.sale rw/
_
```

Note that the cursor is at the bottom of the stack. Pressing the return key will return to CI. A new command can be entered. The cursor can be moved to any line using the terminal cursor control keys. The line can be edited using the local editing keys of the terminal. When the carriage return key is pressed, the line is entered as if it were typed from the terminal keyboard.

You can recall just the last command with the cursor positioned on the command line. This is done with two slashes. For example:

```
CI> //
---Commands---
prot report5.sale rw/
```

In this example, pressing the carriage return key will repeat the last command. To display the stack with the cursor positioned on the second to the last line, enter three slashes. The number of lines backward from the last line can be specified with the corresponding number of slashes after the command stack command (the first slash).

A slash followed by a number can be entered. In this case, a screenful of commands is displayed beginning with the line number specified (backward from the last line). The cursor is positioned on the line specified.

At this point, either enter the command or use the terminal editing keys to change the entry. Pressing the carriage return key will enter the command line. If you do not wish to enter this line, you can enter a slash to repeat the whole command stack or move the cursor to a blank line and press the return key to return to the CI prompt.

You can change the command stack display size by entering a slash followed by the display size desired followed by another slash. For example, the following command changes the command stack display size to 15 lines:

```
CI> /15/
```

Obtaining System Status

The following system status can be displayed on your terminal screen with the appropriate CI commands. How to use these commands to get the desired information is explained in this section. Note that these are the most common commands. A full description of the status command WH is given in Chapter 5.

Status of your programs
Status of all programs
Status of system memory usage
Status of system devices

Displaying Program Status

One of the most useful display is a listing of your programs' status. To display the status of your programs, use the WH command as shown in the following example.

```
CI.65> wh

10:49:44:420
-------------------------------------------------------------------------
  PRGRM T PRIOR PT  SZ DO.SC.IO.WT.ME.DS.OP.     .PRG CNTR.  .NEXT TIME.
-------------------------------------------------------------------------
**PROG2 6 00090  5  18 * * * *  3,CI.65 * * * * * P:45031
  CI.65 6 00051 17  32 . . . .  3,WHZAT . . . . . P:21171
  WHZAT 1 00002  0   . .  1,  . . . . . . . . . . P:61647
.
-------------------------------------------------------------------------
ALL LU'S OK
ALL EQT'S OK
LOCKED LU'S (PROG NAME)  67(EDI67), 73(EDI73), 76(EDI76),
-------------------------------------------------------------------------
10:49:45: 60
```

In this example, there are three programs in memory. The display shows the status of the three programs. Their names are listed in the left-most column. Note that one of them is the WHZAT program which is run when the WH command is entered. This program is often abbreviated to WH. The other two are the command interpreter CI, and a program called PROG2. CI is waiting for WH to finish. PROG2 is running at priority 90, but since WH is running at a higher priority, WH preempts PROG2 while it is printing its information. The status column shows what the programs are doing. The common conditions are shown in the following example. The information in the other columns is explained fully in the WH command description in the RTE-6/VM Utility Programs Reference Manual.

To display the status of all programs, enter:

```
CI.65> wh al
```

A sample display is shown below.

```
10:51: 5:760
-------------------------------------------------------------------------
  PRGRM T PRIOR PT  SZ DO.SC.IO.WT.ME.DS.OP.     .PRG CNTR.  .NEXT TIME.
-------------------------------------------------------------------------
**FMG65 3 00052  5  18 * * * *  3,CI.65 * * * * * P:45031
  CI.65 6 00051 17  32 . . . .  3,WHZAT . . . . . P:21171
  WHZAT 1 00002  0   . .  1,  . . . . . . . . . . P:61647
.
**FMG54 3 00052  8  18 * * * *  3,EDI54 * * * * * P:45031
  EDI54 6 00051 10  32 . . . 2,EQ: 13,AV:2,ST:002 P:23532
**FMG76 3 00052 20  18 * * * *  3,EDI76 * * * * * P:45031
  EDI76 6 00051 16  32 . . . 2,EQ: 35,AV:2,ST:000 P:23532
**FMG73 3 00052 18  18 * * * *  3,RUN73 * * * * * P:45031
  RUN73 4 00075 19  18 . . . 2,EQ: 32,AV:2,ST:002 P:44573
**FMG67 3 00052 11  18 * * * *  3,EDI67 * * * * * P:45031
  EDI67 6 00051 12  32 . . . 2,EQ: 26,AV:2,ST:000 P:23532
.
  SPOUT 1 00011  0   . . . . .  3,CL  046 . . . . P:41706
  UPLIN 1 00003  0   . 0, . . . . . . . . . . . . P:00000   10:51: 8: 80
  GRPM  1 00004  0   . . . . .  3,CL  060 . . . . P:51050
  RTRY  1 00020  0   . . . . .  3,CL  059 . . . . P:55325
  LUMAP 2 00030  4   6 . . . .  3,CL  045 . . . . P:36045
  LOGON 3 00049 23  12 . . . .  3,CL  062 . . . . P:40535
  LGOFF 3 00053 14  10 . . . .  3,CL  063 . . . . P:37227
  R$PN$ 3 00005 26   4 . . . .  3,CL  061 . . . . P:36032
  RSM   3 00020 22   4 . . . .  3,CL  057 . . . . P:42025
  RFAM  3 00030 25  10 . . . .  3,CL  049 . . . . P:55153SWP
  EXECM 3 00030  7   4 . . . .  3,CL  052 . . . . P:41525
  OPERM 3 00030  6   3 . . . .  3,CL  054 . . . . P:37464
  PTOPM 3 00030  9   8 . . . .  3,CL  050 . . . . P:37601
  EXECW 3 00030 15   9 . . . .  3,CL  051 . . . . P:37706
  DLIST 3 00030  6   4 . . . .  3,CL  053 . . . . P:42053SWP
  QCLM  3 00028  5   3 . . . .  3,CL  065 . . . . P:36026SWP
  INCNV 3 00020  3   3 . . . .  3,CL  056 . . . . P:37303SWP
  OTCNV 3 00020  4   3 . . . .  3,CL  055 . . . . P:37247SWP
  CI.71 6 00051 13  32 . . . 2,EQ: 30,AV:2,ST:002 P:42235
-------------------------------------------------------------------------
ALL LU'S OK
ALL EQT'S OK
LOCKED LU'S (PROG NAME)  67(EDI67), 76(EDI76),
-------------------------------------------------------------------------
10:51: 8:370
```

This display is similar to the previous example, except that it includes other programs, either system programs or programs belonging to other users. Programs are grouped by owner; those at the bottom of the list are system programs or subroutines.

The status column in this example includes several categories. These are briefly explained below.

Status	Meaning
scheduled	The program is scheduled to run or executing.
waiting for WH	The program (CI) is waiting for WH to finish.
dormant	The program is not running.
dormant - saving resources	The program is still in memory but not running, waiting for directives.
class susp on class #xx	The program is suspended waiting on a class number, usually waiting for I/O.

Displaying Memory Usage

How the operating system uses memory is indicated by the partition status. Use the WH command to display what programs are in which partitions. You can use this information to tell if you have enough memory in your system for all of the programs that you want to run. To display the partition status, enter the following:

```
CI.65> wh pa
10:51:48:550
-------------------------------------------------------------------------
PTN#  SIZE    PAGES  BG/RT SHR/LBL #ACT L PRGRM PTN-PRIOR
-------------------------------------------------------------------------
 1 R   32    65-  96 BG                   D.RTR       1
 2M   200    97- 296 BG                   <NONE>
 3S    32    97- 128 BG                   <NONE>
 4S    32   129- 160 BG                   LUMAP     294
 5S    32   161- 192 BG                   FMG65     170
 6S    32   193- 224 BG                   OPERM     294
 7S    32   225- 256 BG                   EXECM   26192
 8S    20   257- 276 BG                   FMG54    2854
 9S    20   277- 296 BG                   PTOPM    1578
10     32   297- 328 BG                   EDI54      51
11     32   329- 360 BG                   FMG67    1374
12     32   361- 392 BG                   EDI67      51
13     32   393- 424 BG                   CI.71      51
14     32   425- 456 BG                   LGOFF     137
15     32   457- 488 BG                   EXECW     292
16     32   489- 520 BG                   <NONE>
17     32   521- 552 BG                   CI.65      51
18     32   553- 654 BG                   FMG73      76
19     32   585- 616 BG                   RUN73      75
20     50   617- 666 BG                   FMG76      52
21M    50   667- 716 BG                   <NONE>
22S    25   667- 691 BG                   RSM       282
23S    25   692- 716 BG                   LOGON     129
24M    51   717- 767 BG                   <NONE>
25S    26   717- 742 BG                   LUQUE      25
26S    25   743- 767 BG                   R$PN$      85
27-50  <UNDEFINED>
-------------------------------------------------------------------------
MAXIMUM PARTITION SIZE AVAILABLE
RT    50 PAGES, BG    50 PAGES, MOTHER   200 PAGES
MAX. PART. SIZE GUARANTEED AVAILABLE - DUE TO SHAREABLE EMA
RT    50 PAGES, BG    50 PAGES, MOTHER   200 PAGES
-------------------------------------------------------------------------
10:51:50:810

CI.65> _
```

Displaying I/O Configuration

Systems differ widely in the number and types of peripherals, such as discs, printers and tape drives. The input/output (I/O) configuration of a system is the way that these devices are connected to the system and identified. The operating system identifies each device by a logical unit (LU) number or an Equipment Table entry (EQT) number. The LU or EQT number for a particular device can be used to specify that device. The I/O configuration information can be displayed with the LUPRN utility or the SL command. The SL command displays an abbreviated status of all session devices or seleted devices. The LUPRN utility displays a more detailed status of the system devices.

The LUPRN utility displays, on your terminal screen, a listing of what devices are on your system. For each LU, LUPRN shows the type of device attached to that LU, a select code identifying what I/O card the device is attached to, the subchannels associated with that device, and the device status. Refer to the RTE-6/VM Utility Programs Reference Manual for more detail about LUPRN.

Following are examples using the SL command to display device information.

To display information about LU 6.

```
CI.65> sl 6
SLU  #  6=LU  #  6 = E   6
```

The display shows the session and system LU numbers, the EQT number, and device status if the device was down (inoperative or off-line).

To display all devices:

```
CI.65> sl
SLU   1=LU  # 65 = E  17
SLU   2=LU  #  2 = E   1
SLU   3=LU  #  3 = E   2
SLU   4=LU  #145 = E  17 S 1
SLU   6=LU  #  6 = E   6
SLU   8=LU  #  8 = E   8
SLU  20=LU  # 20 = E   2 S 1
SLU  23=LU  # 23 = E   2 S 4
SLU  26=LU  # 26 = E   2 S 7
SLU  36=LU  # 36 = E   2 S17
SLU  47=LU  # 47 = E   2 S28
CI.65>
```

To display I/O information using the LUPRN utility:

```
CI.65> luprn
                    RTE-6 System Device Configuration
                RTE-6 System rev = 2440  LUPRN's rev = 2440
             9:54 AM FRI., 1 MAR., 1985...Sorted by Session LU
  Time Base (11B)  Priv. Fence SC (none)  Partitions (35)  Memory size (1024K)

 SLU LU  EQT,sc SCD Flags AV T.out  Stats  Driver DP Device Name      LU  SLU
 --- --- ------ --- ----- -- ------ -----  ------ -- --------------- --- ---
   1  73  13    21B  BPS   2 327.67    2B  DVM05  47 8-CH MUX (DDV05) 73   1
   2   2   1    12B D                120B  DVR32   4 7905/6/20/25 DSK  2   2
   3   3  2,10  13B D                100B  DVP32  50 7905/6/20/25 DSK  3   3
   4 108  13,1  21B  BPS   2 327.67    2B  DVM05  47 Left CTU @ LU 80 108  4
   5 109  13,2  21B  BPS   2 327.67    2B  DVM05  47 Right CTU@ LU 80 109  5
   6   6   6    20B  B                131B  DVB12  44 2608A Printer     6   6
   7   7  7,1   30B   PS       .02         DVA65  53 DS1000 to 1000    7   7
   8   8   8    15B  B S     5.00    1B  DVR23  56 9TK Mag Tape #0     8   8
   9   9   9    25B  B S     5.00    1B  DVR23  56 9TK Mag Tape #0     9   9
  10  10  1,4   12B D                120B  DVR32   4 7905/6/20/25 DSK 10  10
  11  11  1,5   12B D                120B  DVR32   4 7905/6/20/25 DSK 11  11
  12  12  1,6   12B D                120B  DVR32   4 7905/6/20/25 DSK 12  12
  15  15  1,9   12B D                120B  DVR32   4 7905/6/20/25 DSK 15  15
  17  17  1,10  12B D                120B  DVR32   4 7905/6/20/25 DSK 17  17
  24  24  10    24B D  S     60.00   40B  DVM33  44 CS-80 Tape Drive 24  24
  30  30  2,2   13B D                100B  DVP32  50 7905/6/20/25 DSK 30  30
  48  48  1,13  12B D                120B  DVR32   4 7905/6/20/25 DSK 48  48
  61   4   4    17B   PS                 4B  DVA66  53 HDLC/BiSync card  4  61
  62   5   5    17B   PS                     DVA66  53 HDLC/BiSync card  5  62
  65   3  2,10  13B D                100B  DVP32  50 7905/6/20/25 DSK  3  65
 --- --- ------ --- ----- -- ------ -----  ------ -- --------------- --- ---
     DP=Driver Partition page ($=SDA), SLU=Session LU
     (T.out is in seconds)                EQT Flags:
     LU # with a      EQT availability:     D=DCPC, B=Buffered, T=Timed-out
     D means the       1=down, 2=busy,      P=Driver handles Powerfail
     LU is down.       3=waiting DCPC       S=Driver handles Timeout
```

Controlling Devices

There may be times when you need to control the operations of an I/O device from the terminal, such as rewind tape, eject paper, etc. The system performs these operations in response to the device control requests. To control devices interactively, the CN command is used to send the control requests. This is done by entering the CN command with the proper command parameter. The common functions and the command parameters required are listed below.

Function	Command Parameter
Reset device	0 (zero)
Top-of-Form (paper feed)	TO
Rewind tape	RW
Write End-of-File	EO
Forward one file	FF
Backward one file	BF
Forward one record	FR
Backward one record	BR

The following examples illustrate some typical device control requests. In these examples, the printer is LU 6 and the magnetic tape drive is LU 8. For some common devices such as tape drives and printers, the parameters may be omitted for the most common operations. For example, if CI recognizes an LU as a tape drive or printer, it will assume that the command without any control parameter is rewinding tape or ejecting paper, respectively.

CI.65> cn 6	Eject paper on printer
CI.65> cn 8	Rewind tape drive on LU 8
CI.65> cn 8 ff	Advance tape to the next file
CI.65> cn 8 bf	Rewind tape to the previous file
CI.65> cn 8 fr	Advance tape to the next record
CI.65> cn 8 br	Rewind tape to the previous record

In each RTE computer system, there are many different types of devices that are each controlled by a software interface module called a device driver. There may be additional parameters needed for controlling a peripheral device. Refer to the appropriate driver reference manual for details.

Various other I/O control requests can be issued with the CN command. For example, the following example sets up multiplexer ports.

```
CI.65> cn 1 30b 152331b
```

In this case, the numbers are octal parameters required for the multiplexer, specified in the multiplexer documentation. This entry sets up LU 1 as a 9600 baud terminal on port 1 with the standard options. Other control requests are described in the CN command description in the RTE-6/VM Terminal User's Reference Manual.

Changing I/O Device Attributes

The CI program provides several commands that modify I/O operations. The operating system maintains a set of attributes for each device. The common attributes include the operational status of the device and the waiting period to complete an I/O request. Most of the attributes are set up when the system is created; details are contained in the RTE-6/VM System Manager's Reference Manual. Some of the attributes can be modified with CI commands if necessary. Modifications made with CI commands remain in effect until the system is rebooted or another modification is made to the same device. These CI commands are described in the following subsections.

In addition to the attributes mentioned above, there are other attributes which can be changed in special situations. These attributes are the device HP-IB address, device priority (not to be confused with program priority), and the driver parameters specified at generation time. These can only be done by the System Manager.

How to Up a Device

One of the most important attributes of a device is whether it is working or not. The operating system maintains the device status, whether a device is "up" (working) or "down" (not working). All devices are initially assumed to be working; when the operating system finds out that a device is not working, it suspends I/O operations to the device until the situation has been corrected. The command to notify the system that a particular device has been fixed is the UP command. For example, to notify the system that the magnetic tape unit (LU 8, EQT 8) is operational, enter:

```
CI.65> up 8    (The EQT number, not the LU number, is used.)
```

This allows I/O requests to go to the device. If the original problem reoccurs, that device will go down again. This happens for various reasons. A device may be inadvertently taken off-line, effectively disconnected from the system. Tape drives go off-line because most tape save/restore utilities put the tape drive off-line when they are finished, to allow removal of the tape and to prevent another user from using the tape drive. Printers are taken off-line for manual form feeds. Whenever a device is off-line and you need to access it, you must place it on-line and up the device before using it. Otherwise the device will not be able to complete the control request and the operating system will mark the device as down.

When the operating system detects a downed device, a message is displayed:

```
CI.65> cn 4
IONR L*  4 E  14 S ***
```

This indicates that the device will be unavailable until the problem is fixed. Note that only the first request to a down device gets this message. Subsequently, all I/O control requests to that device will be placed on hold. It can be mysterious to have programs waiting to access a down device, because the programs will seem to be waiting for no reason. Either the WH or LU command can be used to find out if there are any down devices in the system. To bring up your terminal, use the EQT number for that terminal and not LU 1.

How to Change Time-Out Values

In most cases there is something wrong if an I/O operation takes too long. A disc or printer should always respond within 1 second; any disc I/O operation that has not been completed in 5 seconds usually means that a problem has occurred. The operating system detects when this condition occurs through a mechanism called a time-out.

Each device has a time-out value associated with it. This is a device attribute that tells the system how long it should wait for a response from the device. When the system starts an I/O operation, it also starts a timer. If this timer goes off before the operation completes, then some appropriate action is taken. This action varies from device to device; it is determined by the driver. Usually the device is noted as being down, awaiting user intervention.

Time-out values can be specified either during system generation or with the TO command. They are specified in units of 1/100th of a second. A time-out value of 100 means one second. This unit is chosen to match the resolution of the time base generator. The associated EQT number of an LU is required to specify the time-out value.

To set the time-out on LU 8 (EQT 8) to 10 seconds, enter the following:

```
CI.65> to 8 1000
```

This sets the device with an EQT number of 8 (by normal convention a magnetic tape unit) time-out value to 10 seconds.

There are two other useful forms of the TO command. Specifying a time-out of zero really requests an infinite time-out. This is useful for devices where there is no limit to how long it might take I/O to complete. The most common example is a terminal; there is no particular time limit for entering commands, so it is reasonable to set terminal time-out values to zero.

Giving the TO command with an EQT number but without any time-out parameter will display the time-out value currently in effect for that device. To display the time-out value for your terminal, use the system EQT number for the terminal and not LU 1.

Displaying System Time

The current system time can be displayed with the TM command. Although this command is also used to reset the system time, it is typically used only by the System Manager or installer for this purpose.

To display the current system time, enter:

```
CI.65> tm
Thu Nov 17, 1983 10:55:34 am
CI.65>
```

It is important to maintain the correct system time. Otherwise, features such as time scheduling programs and time stamping files cannot be used effectively.

Unlocking a Shareable EMA Partition

The UL command is a useful tool for situations that can occur when a shareable EMA program aborts or ends without unlocking the shareable EMA partition. This partition left in memory remains locked until released by the UL command. For example, to unlock a shareable EMA partition labeled D12, enter the following:

```
CI.65> ul d12
```

You should use the UL command only when you know that no other program requires the shareable EMA partition.

Executing a Command File

To execute a series of commands without operator intervention, a command file can be created, using the EDIT program. This file contains all the commands to be executed in the desired sequence. To execute commands contained in this file, the TR command is entered specifying the command file desired. At the end of the command sequence, the system returns to the interactive mode.

The following is the content of a sample command file called REPORT.CMD on the working directory.

```
co report01::src datafile1::data
co report02::src datafile2::data
    :
co report30::src datafile30::data
pu report01::src ok
pu report02::src ok
   :
pu report30::src ok
pu /src ok
```

To execute these commands, enter:

```
CI.65> tr,report.cmd
co report01::src datafile1::data
Copying REPORT01::SRC to DATAFILE1::DATA ... [ok]
 :
pu report01::src ok
Purging REPORT01::SRC
 :
CI.65>
```

Positional Variables

Positional variables are defined in the CI runstring or in the TR command. The variable names are $1 through $9, where the number following the dollar sign indicates the position of the variable in the CI or TR command string. For example, to set up the positional variables:

```
CI.65> ru,ci,manual.cmd,um1,um2,um3,um4,um5,um6,A,B,C
```

or

```
CI.65> tr,manual.cmd,um1,um2,um3,um4,um5,um6,A,B,C
```

Positional variables can be separated by blanks or commas; however, commas must be used if you specify positional variables that are not consecutive.

For example, if you transfer to a command file and want to specify values for only $1 and $4, the runstring would be as follows:

```
CI.65> tr myfile.cmd prog1,,,prog4
```

The three commas are required to ensure that the value of positional variable $4 is prog4.

You can specify any string (for example, a number or a valid file descriptor) for the positional variables; unspecified positional variables are set to null. If you specify more than 9 variables, only the first 9 values are used and the extra values are ignored. The command string containing the positional variables can be a maximum of 256 characters, including delimiters. Positional variable cannot be deleted.

The values of positional variables are local. If they are set in a CI runstring, the values are used until that session of CI terminates. If the positional variables are set in a TR command string, their values are valid until you exit from the command file. Before executing a TR command, CI saves the current values of $1 through $9. While executing the command file, the values specified in the TR command string are used in the variable substitutions. When the command file is exited, the original values of $1 through $9 are restored.

A command file must be specified when you set the positional variables; however, LU 1 (your terminal) can be entered instead of a command file name. For example, you can specify values for the positional variables as follows:

```
CI.65> tr 1 myprog1 myprog2 myprog3 myprog4 myprog5
```

You then can use the positional variables in other CI commands; for example, LI $1 would list file MYPROG1 at the terminal.

Concatenation of variables is allowed. For example;

```
CI.65> tr 1 R T E                    (3 parameters set up)
CI.65> co $1$2$3AnswerFile SpoolA    (Copies file RTEAnswerFile)
```

User-Defined Variables

The SET command allows users to define variables, and the UNSET command to delete the user-defined variables. User-defined variables are global; they can be used in CI command strings and at different levels of nesting command files. A user-defined variable is referenced by preceding the name with a a dollar sign ($). For example, if the value of variable name is set to RTE as shown below,

```
CI.65> SET name = RTE-6/VMPrimarySys
```

then the user-defined variable $name can be used in other CI commands and in any command file to represent the value RTE-6/VM_PrimarySys.

When referencing a user-defined variable, CI determines the end of the variable name to be the first character that is not valid for a variable name (valid characters are letters, digits, and underscores). For example, in the following command, the period indicates the end of the user-defined variable name:

```
CI.65> echo $file.ftn
```

This allows you to define similar variable names such as $FILE, $FILENAME, and $FILENAME1.

concatenation of user-defined variables is allowed. For example:

```
CI.65> set file = program1          (Define file name,
CI.65> set ext = .ftn               file type extension,
CI.65> set dir = ::mydir            and directory.)
CI.65> ftn7x $file$ext$dir          (Compile PROGRAM1.FTN::MYDIR)
```

Another example of concatenation is as follows:

```
CI.65> set dir = /system             (Define a directory.)
CI.65> li $dir/answers               (List file /SYSTEM/ANSWERS.)
```

Note that the slash (/) entered after user-defined variable $DIR is necessary. If you omit the slash, CI determines the variable name to be $DIRANSWERS because the blank after ANSWERS is the first invalid character for a variable name.

To concatenate two words, use the single character quote (backslash, described in the Quoting section below). For example, variable NAME is set up as user,

```
CI.65> set name = user
CI.65> li $name\2          (list file user2)
CI.65> li $name\3          (list file user3)
CI.65> li $name\prog       (list file userprog)
```

Note that the character after the backslash is not changed to uppercase by CI. So if you need to enter a string such as Pref, you must type P instead of p.

You should delete unneeded user-defined variables. CI uses its free space to save variable names and values. If too many variables are defined, CI runs out of space and returns an error. This may effect user-defined and predefined variables. For example, CI may not have enough space to redefine variable $WD if the new working directory name is large, and there is little free space left.

Predefined Variables

There are predefined variables in each CI session. The values are the default values used by CI. However, you can use the SET command to modify the variable values. The SET and ECHO commands can be used to display the values of predefined variables. You cannot delete these variables.

The predefined variables are listed and described as follows:

$AUTO_LOGOFF

Allows for automatic logoff if session is inactive. CI initializes $AUTO_LOGOFF to zero, which means automatic logoff is not in effect. If you set $AUTO_LOGOFF to a nonzero value, CI times out after that many terminal time-outs. If CI is the only active program, after four CI time-outs, an EX command is executed to terminate the session.

$LOG

A flag indicating if commands executed in a command file are logged to the terminal. CI initializes this variable to OF, which means that commands are not displayed at the terminal. To display commands at the terminal, set the value to ON.

$OPSY

The ID number of your operating system.

$PROMPT

The prompt that is displayed when CI is waiting for input. CI initializes this variable based on the name of the CI program and your session number.

$RETURN1 - $RETURN5

Five integer values (ASCII representation) returned from execution of the last command. CI updates the values as commands are executed. These variables can be set to values between -32768 and 32767, inclusive.

$RETURN_S

An 80 character string returned from execution of the last command. CI updates the value as commands are executed.

$RU_FIRST

Flag indicating whether RU or TR is to be assumed if you only enter a file name in response to a CI prompt or as a line in a command file. CI initializes this variable to TRUE, which means CI first attempts to execute a RU command for the specified name. Set this value to FALSE if you want CI to assume that the file name entered is the name of a command file. You should set the variable to FALSE if you will be executing more command files than program files.

$SAVE_STACK

Flag indicating if the command stack is saved when you exit CI or when the command stack file is changed with the WD command. CI initializes this variable to TRUE, which means the stack should be saved when CI exits. Set the value to FALSE if you do not want the stack saved.

$SESSION

Number of your current session. CI initializes this variable to your session number and updates the value after execution of every CI command.

$WD

Name of the current working directory. CI updates this variable after execution of each WD command.

The following example changes the value of $PROMPT:

```
CI.65> set prompt = waiting:
WAITING:
```

The following example displays the value of $OPSY:

```
CI.65> echo $opsy
-17
```

Nesting Command Files

Command files can be nested by using the TR command, implicitly or explicitly, in a command file. Before CI transfers control to a new command file, the positional variables ($1 through $9) are saved. Upon returning from a lower level of command file, these values are restored.

Quoting

There are two methods of quoting available to allow characters to pass unaltered to the destination program, command file, or CI command. A single character is quoted by preceding it with a backslash (\). A string is quoted by enclosing it in grave accents (`).

If you do not quote a character or a string, CI shifts it to uppercase, replaces each contiguous group of blanks with a comma, and performs variable substitution before executing the command.

To include a grave accent in a quoted string, enter a second grave accent with the accent you want passed as part of the string.

Some examples of quoting are as follows:

```
CI.65> echo `Hello.  How are you?` (Displays string unaltered by CI.)
Hello.  How are you?

CI.65> ru,savename,jane\ doe       (Passes a blank in command string.)

CI.65> echo `This is a grave accent (``).` (Passes a grave accent as
This is a grave accent (`).                part of the quoted string.)
```

Multiple Commands Per Line

You can enter more than one CI command per input line by separating the commands with semicolons (;). Blanks immediately before or after a semicolon are ignored. A semicolon enclosed in quotes loses its effect as the command separator.

Examples:

```
CI.65> wh;dl
```

Executes the WH program followed immediately by DL.

```
CI.65> ftn7x test.ftn 0 - ; link test.rel ; test
```

Compiles, links, and runs program TEST.

Return Status

Most commands, programs, and command files can return status to CI to indicate success or failure of execution. CI interprets an internal status returned by commands.

Programs and command files can return five integer values and a string to CI. The first of these integers is used for status. The rest of the values are additional information for the user. A status of zero indicates success; anything else indicates failure. The five integers are then made available to the user in the string variables $RETURN1 through $RETURN5. The returned string is saved in variable $RETURN_S.

Programs pass CI the five integer values through the system routine PRTN, and the string via an EXEC 14 call. Command files return these values using the CI RETURN command. See the RETURN command description for further details.

The return status is used by CI's execution control structures discussed later in this chapter. Note that the control commands IF-THEN-ELSE-FI, WHILE-DO-DONE, and the SET and ECHO commands do not alter the return variables. This is to assure that the user will be able to access these values before they are modified.

The AG, BL, CU, DN, EQ, IT, LU, OF, ON, PR, QU, ST, SZ, TI, TO, UL, UR, VW, and WS commands do not return status to CI; therefore, $RETURN1 will always equal zero after any of these commands are executed.

Execution Control Structures

A powerful feature available in command files is the IF-THEN-ELSE-FI and WHILE-DO-DONE control structures. These provide a means for decision making during execution of the command file. Chapter 5 contains detailed information on these control structures.

The following statements compile TEST. If no errors or warnings occur when TEST is compiled, TEST will be linked. Otherwise, EDIT will be run on TEST.FTN to allow you to fix the errors.

```
IF ftn7x test.ftn 0 -
THEN link test.lod
ELSE edit test.ftn
FI
```

In the following example, the file SOME_FILE will be printed 5 times. The IS command compares the value of $COUNT and zero; as long as $COUNT is greater than 0, the WHILE loop continues executing. CALC is a simple program that accepts two ASCII representations of integers, converts them to integers and performs the specified operation. The result, in ASCII form, is returned to $RETURN_S.

```
set count = 5
WHILE is $count gt 0
   DO calc $count - 1
      set count = $RETURN_S
      print some_file
DONE
```

Time-Out/Logoff Function

To eliminate inactive sessions on a system, CI has the ability to log off a user. The variable $AUTO_LOGOFF can be defined to tell CI how many device time-outs can occur at the user's terminal before CI times out. Each time CI times out a warning message will be displayed on the terminal. After the fourth CI time-out, CI executes an EX command.

The following example begins the CI time-out process after CI waits 15 minutes for input. First, you set the terminal time-out to 30000 (see the TO command) and set the $AUTO_LOGOFF variable to 3. In this example, CI terminates after 60 minutes.

```
CI.65> to 113 30000
CI.65> set auto logoff = 3
CI.65>
Waiting for input...
Going...
Going...
Gone!
Finished
```

Chapter 3
Manipulating Files

This chapter shows how to use the CI commands to manipulate files, directories, and subdirectories. Before using these commands, you should be familiar with the CI file properties and features such as file masking and command stack. This information is important if you want to take full advantage of the file system. The CI file properties and features are also described in this chapter. A summary of file manipulating commands is given in Table 3-1.

File Properties

Each file has certain associated properties: some describe the way that information is organized in the file and others contain such information as location, ownership, protection, and time stamps. These are listed below and described in the following paragraphs.

file name
directory
subdirectory
file type extension
file type
file size
record length
owner
protection
time stamps

File Names

Each file has a name to distinguish it from other files. A file name can have up to 16 characters. File names should contain only letters and numbers, e.g., NOTES or TEST23. Use of punctuation characters should be avoided. However, some files created for use with the FMGR program may contain punctuation characters in the file names. These files can be managed by CI. Capitalization of file names is optional; CI allows entry of either uppercase or lowercase letters. CI always shifts the input to uppercase. File names must not start with a number, because the first letter is used to distinguish between file names and logical unit (LU) numbers that represent I/O devices.

Table 3-1. File Manipulating Commands

Command	Task
CL	List mounted disc volumes
CO <src file> <dest file>[parm]	Copy file
CR file	Create file
CRDIR <directory name>[lu]	Create directory
DC lu	Dismount disc volume
DL mask	Display directory contents
IN lu [blocks[ok]]	Initialize disc volume
LI file	List file
MC lu	Mount disc volume
MO <src file/lu> <dest file/lu>	Move file
OWNER directory[newOwner]	Display/reassign directory owner
PATH[-E]	Display current UDSP information
PATH[-E[-N:n]] udspnum [dirname1[dirname2[...]]	Display/define specified UDSP or UDSP entry
PATH[-E] -F,file/lu	Display/define UDSP using commands from specified file/lu
PROT file	Display/modify file protection
PU file[ok]	Purge file
PU <directory name>	Purge directory
RN file <new name>	Rename file
RN <old dir> <new dir>	Rename directory
UNPU file	Unpurge file
WD[directory name[file/+s]]	Display/set up working directory
WHOSD <directory name>/lu	Display session using directory or disc LU as part of a UDSP

The file name includes a file type extension that shows what type of information is in the file. It is separated from the file name by a period and may contain up to four characters. Thus, the full file name can be up to 21 characters. For example, a file name may be:

```
currentmanualch1.text
```

Blank file type extensions are allowed; the period can be omitted if the file type extension is blank. Standard file extensions should be used when files contain standard information. For example, all executable program files should have file extension RUN, relocatable files should have REL, etc. The standard file type extensions are given later in this chapter. To access a file with a file type extension in the file name, the file type extension must be specified.

File Descriptors

A file descriptor is a term used to specify a file using any or all of the optional parameters, including subdirectories and directory. The file attributes specified by these parameters include file size, types, and record length. Colons are used to separate parameters and slashes are used to separate subdirectories, directory, and file name. The formats of the file descriptor are shown below:

```
    filename::directory:type:size:<record length>
 or
    /directory/subdirectory/filename:::type:size:<record length>
```

Note that the filename parameter includes the file type extension. Default parameters must have a colon as a place holder when followed by another parameter. The maximum length of the file descriptor is 63 characters including delimiters. File descriptors with more than 63 characters are inaccessible. However, it is possible to create files with file descriptors longer than 63 characters by using working directories or by renaming directories. It is recommended that file descriptors be kept within the range of 30 to 40 characters.

The first form of the file descriptor can be used in accessing FMGR files as long as the rules for FMGR file descriptors are observed. Use six-character file names without any illegal characters and substitute a positive CRN or a negative LU for the directory parameter.

The following are some examples of file descriptors. In these examples, the file names in the user entries are shown in uppercase letters for clarity only. Directories and subdirectories in comments are shown in uppercase letters similar to that shown in the text throughout this manual.

MANUAL.TXT::op:4	(type 4, text file on directory OP)
/op/output/OUTLINE.TXT:::4	(type 4, text file on subdirectory OUTPUT in directory OP)
EDIT.RUN::programs	(file in directory PROGRAMS)
PROGRAMERRS:::3:356	(type 3, text file in working directory with a size of 356 blocks)
/new/pascal.dir	(subdirectory PASCAL in directory NEW)
/new	(directory NEW)
/jones/@.@	(all files in directory JONES)

The filename parameter in the file descriptor may contain a mask qualifier that can be used in multiple file access. In addition, two wildcard characters, "@" and "-" can be used in the filename parameter. Refer to the File Masks paragraph in this chapter for details.

Directories

Directories maintain information about files: file names, file type extensions, all the optional properties defined, and where to locate files. Each file must be in a directory. There can be many directories and even subdirectories that are imbedded inside directories. Duplicate file names may exist in different directories. Directories can be specified in the following ways:

```
    ::<directory name>
 or
    /<directory name>
```

Directories are identified by name. Each directory has a name of up to 16 characters, subject to the same rules as file names. A directory name is specified with a file name in order to identify the file (but it can be omitted). In the first form shown above, the directory name is separated from the file name by two colons. This form is generally used with FMGR files and the CI file system may display this form for compatibility with FMGR files. The two colons are used by FMGR files to define an optional file security code, for example, FILE:SC:CRN. The second form is used in the CI file system. Such a file structure may contain directories that have within each directory nested subdirectories. This form of specifying files is used to indicate the search path for the files in the CI file structure.

If the directory name is omitted in a file name, a default directory called the Working Directory (WD) is used. It can be defined or changed with a WD command. The working directory, once defined, will remain in effect until changed by another WD command. You may display the name of the working directory by using the WD command without any parameter.

Certain programs have a special feature that enables users to schedule other programs without the directory name. If the directory is omitted in the program runstring, standard directories set up by the system will be searched. For example, in executing the run command, CI will search a directory named PROGRAMS for programs specified without a directory. The standard default directory search sequence used by CI is described later in this chapter.

Subdirectories

In addition to containing information about files, directories can contain other directories. Directories which are contained in another directory are known as subdirectories. Subdirectories can include other subdirectories and there can be many levels of subdirectories. Unlike directories, subdirectories can have the same name as long as all names within a directory are unique.

Subdirectories have the same properties as directories. Throughout this manual, references to directories also apply to subdirectories unless otherwise indicated.

When specifying files which are in subdirectories, the hierarchical format is used, with the subdirectory in front of the file name, separated by slashes. For example, if a file named MANUAL.TXT is in directory DIR, it is specified as:

```
/dir/manual.txt
```

If this file is moved to the subdirectory SUBDIR, it would be specified as:

```
/dir/subdir/manual.txt
```

The first form is used when there are no subdirectories. The second form is used to specify a search path in a hierarchical file structure where there may be many levels of subdirectories. There is no limit to how many levels of subdirectories can be nested inside other directories. However, there is a limit to the length of the file descriptor, a maximum of 63 characters including delimiters.

In a sample hierarchical file structure, shown in Figure 3-1, to specify a file named DRAWCKTAA.REL in subdirectory SUBROUTINES:

```
/programs/applications/graphics/subroutines/drawcktaa.rel
```

There may also be a file with the same name in subdirectory APPLICATIONS. To specify this file:

```
/programs/applications/drawcktaa.rel
```

The CI file structure provides a search path so that the search time may be minimized and duplicate file names in different directories or subdirectories are possible.

In the CI file specification, it is easy to confuse subdirectories with directories and thus specify the wrong search path. The point to remember is that a leading slash is needed to specify a directory; without the slash, the name is taken to be a subdirectory. For example,

```
/system/archive/file.txt:::3
```

specifies that FILE.TXT is located in subdirectory ARCHIVE of directory SYSTEM while

```
system/archive/file.txt:::3
```

which is the same descriptor less the leading slash has a completely different meaning from the first. It specifies that FILE.TXT is in subdirectory ARCHIVE of subdirectory SYSTEM. Since the directory is not specified, the working directory is assumed, meaning that SYSTEM is a subdirectory of the working directory.

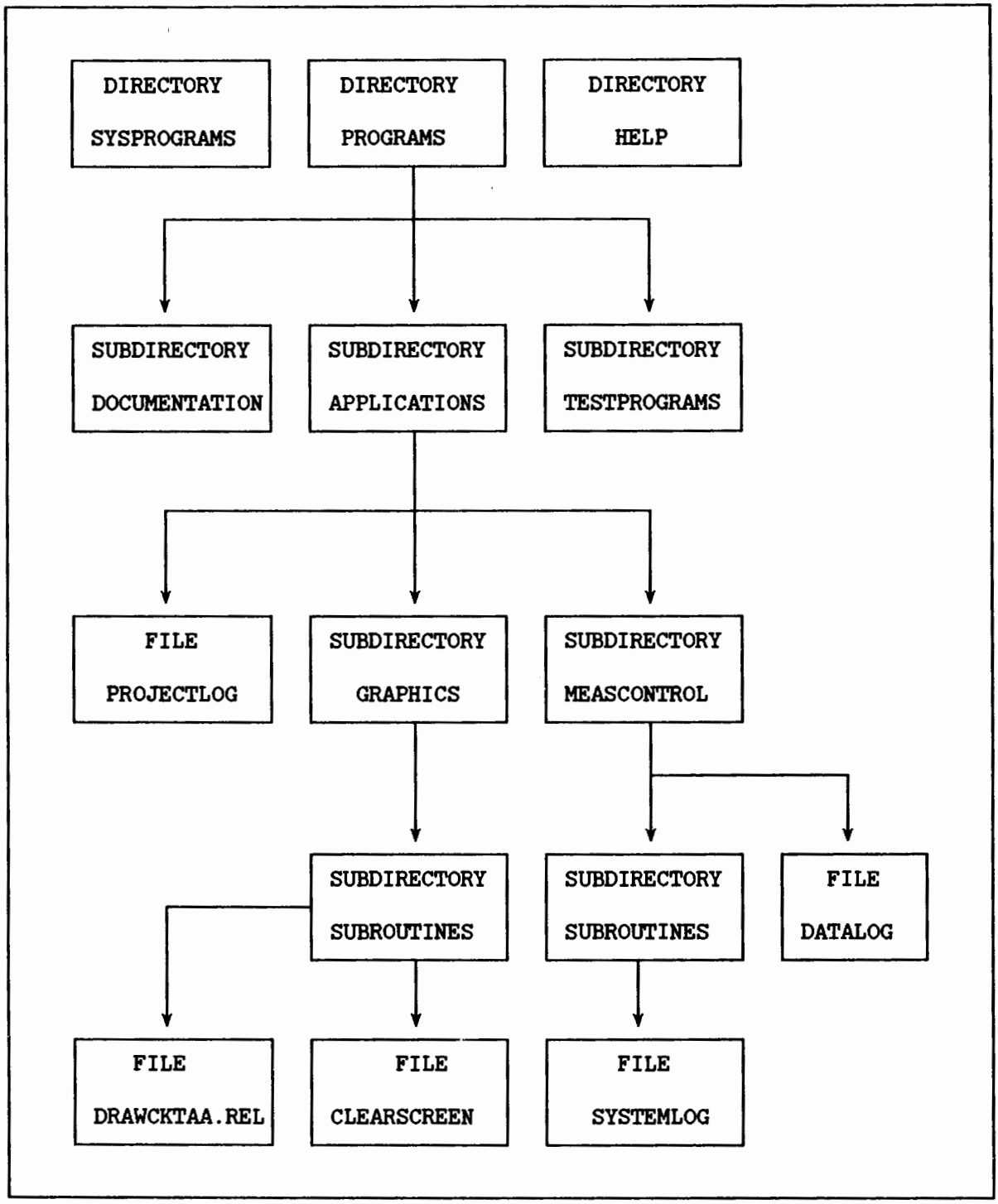


Figure 3-1. Sample CI File Organization

File Type Extension

The file name is supplemented by a secondary name called a file type extension. The file type extension is used to indicate the type of information in the file: text, binary data, etc. It consists of a period and up to four characters appended to the file name. For example, in the filename parameter EDIT.RUN, the file type extension is RUN. Blank type extensions are allowed; the period can be omitted if the type type extension is blank. When specifying a file, the file type extension must be included if one exists. For example, if the file name is RPORT.TXT, you must include the file type extension. If you enter only RPORT, you are implying a blank file type extension that will not match RPORT.TXT. An exception is that various programs and program commands assume different default file type extensions. For example, the CI program RU command uses the default file type extension of RUN for programs scheduled without file type extension. Refer to the respective manuals describing the programs for any default file type extensions (e.g., EDIT, Debug/1000, etc.)

Standard type extensions should be used when files contain standard information. For example, all executable program files should have type extension RUN and all CI transfer files should have file type extension CMD. The standard file type extensions are listed below.

.cmd	CI command file
.dat	data file
.dbg	debug file
.dir	directory or subdirectory entry
.ftn	FORTRAN source file
.lib	library of relocatables
.lod	relocatable loader command file
.lst	listing
.mac	Macro source file
.map	loader map listing
.pas	Pascal source file
.rel	relocatable (binary) file
.run	program file
.snp	system snapshot file
.stk	command stack file
.txt	text file

File Type

Each file descriptor has a file type parameter that indicates how the information in the file is organized. The file type is a number, and is not to be confused with a file type extension. There are standard RTE file types defined with the following characteristics:

Type 0 file is the term used in accessing devices with file calls. There is no disc file or directory entry for a type 0 file, and they do not have the other properties listed in this section. A type 0 file is an I/O device.

Type 1 files are random access files which do not have any structure information in them. These files contain fixed record lengths (128 words). They can be read and written very quickly.

Type 2 files are fixed-length record, random access files. The record length is defined when the file is created. They are usually user created large data files.

Type 3 and above files are variable-length record, sequential files that are suitable for use as text files. There is no difference in the handling of file types 3, 4, and 7. By convention, type 5 is used for the Compiler or Assembler relocatable output files, type 6 is for program files that are memory-images of executable programs, and type 7 is for the Compiler or Assembler absolute binary output files. Type 6 files are treated in the same way as type 1 files. Type 4 is recommended for text files. Type 3 is for general purpose files and can be used for text. This is the default file type when files are created with the CR command. Type 8 and above files are user defined.

File type is important when files are accessed programmatically. Substituting a random-access file for a sequential file or vice versa will cause problems.

File type is specified after the directory name, separated by a colon. For example, you could create a type 1 file with the following:

```
CI.65> cr file.dat::system:1
```

If the subdirectory and directory are specified before the file name, the file type is preceded by three colons. For example:

```
CI.65> cr /system/subdir/file.dat:::1
```

The directory name has been moved to the front in this case, but colons are required as place holders. When creating a file in the working directory, place holders are also required if the file type is specified. For example:

```
CI.65> cr subdir/file.dat:::1
```

If the file type is omitted, as in creating files with the CR command, the default is type 3. However, some programs may have a different file type default. For example, the default file type for the EDIT program is type 4.

File Size

The file size parameter in the file descriptor specifies how many blocks of disc space the file needs. One block is 128 words (256 bytes or characters). One printed page takes about 10 blocks of disc space. You can specify how big a file should be when you create it. If size is unspecified and there is no information about the file, the file system chooses a size of 24 blocks. If the contents of the file are known, such as when creating a file with the CO command, the exact size of the file is used.

The size of a file is specified in the file descriptor after the file type parameter, separated by a colon. You can specify the initial size of a file when you create it. For example, to create a file of 100 blocks, enter the following:

```
CI.65> cr bigger.dat::system::100
```

To specify both size and type, enter the following:

```
CI.65> cr macro.err::system:1:100  (create a type 1 file, 100 blocks in
                                    length, called MACRO.ERR in directory
                                    SYSTEM)
```

In most cases the file system increases the file size to accommodate more data as needed through a process called extending the file. Extents are always created by the file system at least as big as the original file size, because there are performance advantages to having fewer, larger extents. This is true for all except type 6 and some type 1 files which are memory images of programs of the RTE system. Such files are not extended.

Files larger than 16383 blocks are rounded off to multiples of 128 blocks. Files can be as large as any disc available in your system. Files larger than 32767 blocks must be created by specifying the size as a negative number of 128 block "chunks" of the file. For example, a 50000 block file would be specified by a size of -50000/128 = -391, rounding to nearest larger number in absolute value. Large files are usually created by programs and rarely by the user. Be aware that the file size parameter larger than 32767 blocks will be accepted but the desired file size will not be created. For example, the entry "cr file::::36214" creates a file with 13110 blocks.

If the file size is not specified when creating a new file, a default size of 24 blocks is used. Files that increase in size will automatically be extended as required by the file system.

Record Length

Record length is the last parameter in the file descriptor. It is used mostly for fixed-length type 2 files. It is specified in units of words. For example, the following creates a type 2 file of 100 blocks with 64-word records:

```
CI.65> cr file.dat::system:2:100:64
```

This field is also used in type 3 and above files. The file system uses the value of the longest record in the record length field. This value appears in messages displayed by the file system utilities to indicate the longest record. Any other value specified by the user in type 3 and above files will be ignored.

File Ownership

Ownership of files is determined by the owner of the directory containing the files. All files in a directory are considered owned by the directory owner. The owner can change the protection status of files in that directory, which defines the read/write access allowed for the owner and general users.

The owner of the directory (and no other users) can also reassign the directory ownership. The owner of a directory is the user who created it. The same is true for subdirectories. However, the owner of a subdirectory can be different from the owner of the directory that contains the subdirectory. More information about ownership, directories, and subdirectories is given in the paragraphs under the Manipulating Directories section in this chapter.

Protection

File protection is a security measure in the CI file system and is defined when a file is created or copied into a directory. It is specified in terms of read or write access allowed. It can be specified differently for owners and general users. The default protection is to allow the owner both read and write accesses and read access only for other general users.

Each file created assumes the protection status defined for the directory where the file resides. A copied file assumes the protection status defined for the original file. If there is no protection status for a file to be copied, the copied file assumes the protection of the directory where the file is copied into.

Protection can be specified on a file-by-file basis. It is specified by indicating what operations should be allowed for the file owner, and what operations should be allowed for the general users. The available operations are reading and writing the file. You can specify protection in any combination of read and write for the owner and for the general users.

Directories also have protection information. The protection status of a directory applies to any file to be created in that directory. For example, if a directory allows read and write access for the owner and read only for others, all files created in that directory also have the same protection status until changed by the PROT command.

Read and write protection for a directory is slightly different from that for a file. Write protected directories (only read access specified) will prevent the general users from changing information in the directory through CI commands. This means that they cannot create, purge, or rename files in the directory. Read-protected directories will prevent the general users from finding out the contents of those directories.

Time Stamps

Time stamps are maintained for all files in the CI file system. The time stamps include the time of creation, time of last access, and time of last change. Times reflect both time of day and date, with a one-second resolution. Time stamps are not maintained for directories.

Time stamps are changed automatically by the system; users can only examine them with the DL command. The following examples illustrate the use of time stamps.

Examples:

CI.65> dl <file descriptor> a	Examine time last accessed
CI.65> dl <file descriptor> c	Examine time created
CI.65> dl <file descriptor> u	Examine time last updated
CI.65> dl <file descriptor> uac	Display all three time stamps

Creation time is set when the file is created. Update time is set whenever a file is closed after being changed. Closing a file refers to when a program finishes using the file. If a program has a file in use for a long time without closing it, the update time will not necessarily be accurate. Update time is used to tell if the file has been changed. By comparing update times, it is possible to tell whether an output file reflects all changes made to an input file. Access time is set whenever a file is opened; files are opened whenever a program needs to access the information in them. Examining the file protection information does not affect the data in a file, so it does not count as an access. Access time is also changed whenever update time is changed.

When a file is created by copying an existing file, the create and last access times are changed to the time of copying. The last update time, however, will remain the same as the existing file. This is done for preserving the revision history of the file.

File Masks

Access to multiple files can be simplified by a file mask feature. Files can be specified using one or more fields in the file descriptor, ignoring the other fields. The characters in the file name field can also be masked to access a group of related files. Several different files can be specified with a single entry. For example, The daily entries of a system log can be accessed by masking the date code in the file names. All files shown below can be accessed by specifying "syslog------".

```
SYSLOG010181
SYSLOG010281
    :
    :
SYSLOG123081
SYSLOG123181
```

The dash (-) is used to mask one character position, except a blank character. The @ character is used to mask any and all characters. The files shown above can also be accessed with another single entry:

```
SYSLOG@
```

Some file related commands can refer to a number of files using one file descriptor with the aid of a file mask. The file mask feature uses all the fields in the file descriptor plus a special mask qualifier field. The fields used in this manner can be any or all of the following:

file name (including file type extension)
mask qualifier appended to file name
directory
subdirectory
file type
file type extension
file size
file record length
time stamps

The mask characters, "-" and "@" can be used only in the file name and file type extension fields; they have no special meaning in any other fields, including directory and subdirectory. The dash masks a single character position and the @ character masks zero or more characters.

The mask qualifier field is a string of characters appended to the file name after the file type extension. It is separated from the file name by a period. Special characters are used in the qualifier field to facilitate finding the desired files. These characters are:

b match only those files that need to be backed up. (Refer to the TF utility description in the RTE-6/VM Utility Programs Manual).

d a search directive. If any directory matches a mask, then all files in that directory match, regardless of other characteristics.

e a search directive. Search all mounted disc volumes in the system, including the FMGR file system disc cartridges. (This can take a long time, depending on the size, contents, and the number of volumes.)

n do not match directories. Useful mostly for copying. Overrides the d qualifier.

o match only open files.

p match only purged files.

s a search directive. Search from the specified directory down through any subdirectories in that directory, applying the mask to those files throughout the search path.

t match only temporary files.

x match only files with extents (not applied to FMGR files).

Any of the time stamps can be used as the mask qualifier. Time stamps can be specified as an option in commands that use the file mask feature. Time stamps can also be specified as a range. The format is

```
[c/a/u][xxxxxx.xxxxxx][-][xxxxxx.xxxxxx]
```

where the xxxxxx.xxxxxx is a date and time in the format YYMMDD.HHMMSS; 830529.120000 is noon May 29, 1983. Only one choice among c/a/u is allowed (create, access, or update). The default is last update time. The dash character "-" is not a mask character when used in the qualifier field. It is used to specify a range of dates. If "-" is not used, the specification is for files which match that date/time.

For example: c780416.121108-810611.141411

This entry specifies files created between April 16, 1978 12:11:08 and June 11, 1981 14:14:11. The time can be specified with as few digits as desired. Thus a81-83 would mean files last accessed during or after 1981 and before or during 1983.

The time stamps in the file system begin on Jan 1, 1970. Dates specified in years between 00 and 37 (inclusive) will be interpreted as being the year 2000 through 2037. Time stamps only extend through the year 2037.

Appropriate default values for each field are defined. If the name is not specified, all names match. The same is true for type, size, and record length. If the qualifier is not specified, all files qualify except purged files. (Note that if p is specified, then only purged files qualify.) If the file type extension is not specified then only files with blank type extensions match. If the directory and subdirectory are not specified then only the working directory is used. The directory and subdirectory specification has precedence over the e option (if both are specified then only the specified directory is searched).

There are several special cases in specifying directories using file masks. If the mask ends with a / as in /foo/joe/, this is interpreted to be equivalent to /foo/joe/@.@ (default name and file type extension). This mask directs the file system to search for all files with any name and file type extension in subdirectory JOE of global directory FOO. A global directory is one that is unique throughout the disc volume in the CI file system; it cannot be included in any other directory as a subdirectory.

The trailing / is a way of referring to the contents of a directory rather than to the directory itself. To refer to the files in directory FOO, the proper mask is /foo/. Thus to list the files in directory FOO, the command is:

```
CI.65> dl /foo/
```

Note that /FOO will not list the files in directory FOO. Alternate means of listing the files of directory FOO:

```
CI.65> dl ::foo
CI.65> dl @.@::foo
CI.65> dl /foo/@.@
```

If the file name in a mask ends with the wild card character (@) and the file type extension is not specified, then a wild card file type extension is assumed. For example:

```
file1@  is the same as  file1@.@
```

Files with a null file type extension can be specified with a trailing dot as follows:

```
file2@.
```

If the mask ends with DIR as in /foo/joe.dir, this means to match only subdirectory JOE in global directory FOO. The DIR file type extension is needed in all contexts where either a file or directory can be given; it may be omitted where only directories are allowed.

Examples of mask qualifiers:

u82-	(updated during or since 1982)
u82-83	(updated during 1982 or 1983)

Examples using the whole mask:

@	(equivalent to '@.@', specifies all files on the working directory)
/foo/	(equivalent to /foo/@.@, specifies all files on directory FOO)
/foo/@.	(specifies all files on directory FOO with a blank file type extension)
@.ftn.eu82	(search everywhere for all FORTRAN source files last updated during 1982)
/games/backgammon/source/@.@	(all files on subdirectory SOURCE of subdirectory BACKGAMMON on directory GAMES)
@.dir	(all subdirectories on the working directory)
@.@.x:::4	(type 4 files with extents on the working directory)

Other Examples:

dl,@.txt (display files in the working directory with file type extension TXT)

dl,a@.@.c83 (display files in the working directory which start with a, created during 1983)

dl,/joe/@.@.sc80-83 (display files in directory JOE created during the period between 1980 and and 1983. Also, the s qualifier directs a search of any subdirectories of directory JOE for similar files)

Destination Masks

Several commands (COpy, MOve, ReName) may use a destination file mask in addition to a source file mask to give the command a framework for the destination file name. For example, the command "RN @.src @.ftn" will rename all of the files on the current directory which have file type extension SRC to have file type extension FTN. In general, if a name or a file type extension is specified in the destination mask, it will be used for the destination file name or file type extension. If either is defaulted using the @ character, then the name or file type extension of the source file will be used.

The @ character must mask a complete name or file type extension. Thus the command "RN %@.rel @.rel" will NOT remove the "%" from the front of the files with type extension "rel". In the case of the DIR file type extension, it cannot be changed in the destination file descriptor. If the source file type extension is DIR, then the destination type extension will be DIR, regardless of the destination mask type extension.

The destination mask has the same rules as the source mask for implicit "@". Thus /sources/ is equivalent to /sources/@.@. This will result in the default name and file type extension.

For the type and record length fields, the values from the source file will always be used, even if a value was specified in the destination mask. For the security code and file size fields, any value used in the destination mask will override that of the source. The following paragraph describes how the destination directory path is generated.

The destination directory path comes from both the destination mask and the source file directory path. The beginning of the destination directory path comes from the destination mask. Next the source directory path, LESS the directory path in the SOURCE MASK, is appended.

The following examples illustrate destination masks used with the CO command.

To copy all files in subdirectory /PROGRAM/DOCUMENTS into subdirectory /MANUAL/DOCUMENTS:

```
CI.65> co /programs/documents/@ /manual/documents/@
```

The destination subdirectory must exist prior to executing the copy command. This could also have been accomplished using the following command:

```
CI.65> co /program/documents.dir.d /manual/@
```

The ".d" in the source mask specifies all files in the directory /DOCUMENTS. Both forms are identical except that in the second form, the subdirectory will be created if it does not exist.

If these files are to be copied to a subdirectory called /MANUAL/CHAPTERS, changing the subdirectory name at the destination, enter the following:

```
CI.65> co /program/documents/@ /manual/chapters/
```

Subdirectory /MANUAL/CHAPTERS must be an existing subdirectory for the command to work. An alternate form is:

```
CI.65> co /program/documents.dir /manual/chapters.dir
```

The destination subdirectory in this example will be created if it does not exist. More examples are shown below.

```
CI.65> mo main.txt subroutine.ftn   (Move MAIN.TXT into SUBROUTINE.FTN)

CI.65> co main.lst @.temp           (Copy MAIN.LST into MAIN.TMP)

CI.65> rn /program /pgm             (Rename directory PROGRAM to PGM)

CI.65> co /pgm.dir /new/@           (Create subdirectory PGM on directory
                                    NEW with all files and subdirectories
                                    that are in directory PGM)
```

I/O Devices Referenced as Files

In addition to identifying a file, the file name can be a number that identifies an I/O device. This number is a logical unit (LU) number which is assigned at generation time to all devices in the system. The LU numbers for devices such as terminals and printers can be used in most cases where a file name appears. As an example of using LU numbers to indicate I/O devices, try using the CO command to copy a file. Your terminal is always LU 1, so you could display a file to your terminal as follows:

```
CI.65> co welcome.txt::system 1
```

Use of LUs is further described under the heading "Data Transfer To and From Devices" in this chapter.

Directory Listings

One of the most useful file operations is getting a listing of what files are in a directory. This shows the files available to you.

The directory list command is DL. Entering the DL command without any parameters returns a list of file names in your working directory. These will be sorted in alphabetical order.

Example: To list all files in the working directory:

```
CI.65> dl
directory ::SMITH
A.B           D.E           TEMP.FTN      TWENTY.FTN
```

In the sample display shown above, the working directory is SMITH which contains the four files listed. Note that files A and D have uninformative names and non-standard file type extensions. These names are not recommended for important data.

DL can also be used to get a list of the files contained in some other directory simply by specifying the name of the directory. There are several ways to list the contents of a directory and these are shown in the following examples.

To list all files in directory named JONES:

```
CI.65> dl ::jones          (recommended for FMGR files)
CI.65> dl @.@::jones
CI.65> dl /jones/          (recommended for CI files)
CI.65> dl /jones/@.@
```

To list all files in subdirectory SUBDIR which is in global directory JONES:

```
CI.65> dl /jones/subdir/
CI.65> dl /jones/subdir/@.@
```

This gives the names of the files contained in those directories. The trailing slash must be given to the directory or subdirectory of interest to get the desired effect. The use of the slash is a feature of masking file specification previously described in this chapter.

Listing Files

The LI command lists the contents of a file to your terminal for examination. You can use a file mask to list a group of files.

To list file /SYSTEM/WELCOME.CMD:

```
CI.65> li /system/welcome.cmd
```

In this example, the file is displayed on the terminal screen in blocks of lines ending with the following message:

```
More...('a' to abort)
```

You can then select one of the following actions:

Action	Enter
abort the list command	a (no carriage return needed)
display remainder of file	carriage return
display next block of lines	any key other than "a" or carriage return

The abort character "a" can be either uppercase or lowercase. Once it is entered, the listing will stop and the CI.65> prompt is displayed.

Several other commands use this method of display, pausing after a screenful of lines to let you read what has been displayed with the same choices for abort or continuation. If you are using a printing terminal, you may send the command output to a file, then use the CO command to copy the file to the printing terminal without stopping.

If you enter a file mask, you are prompted as follows before each file is listed:

```
Next file: filedescriptor List? (Yes, No, Abort) [Y]?
```

You then can select one of the following actions:

Action	Enter
Don't list this file	n (no carriage return needed)
Abort the LI command	a (no carriage return needed)
Display the next file	any key other than "n" or "a"

The no list character "n" can be either uppercase or lowercase. It causes LI to skip to the next file.

The abort character "a" can be either uppercase or lowercase. After the abort character is entered, the LI command stops and the CI prompt is displayed.

The LI command will list the data in octal if the data cannot be displayed as ASCII characters. It also allows display of a range of lines in the file.

Following are examples of the LI command.

To list lines 15 through 39 of file BIOSIN.YL:

```
CI.65> li biosin.yl,,15 39
```

(note the use of two commas as a place holder)

To display binary information:

```
CI.65> li test.rel b
```

To list all files with type extension .FTN:

```
CI.65> li @.ftn
```

Copying Files

The CO command is a general purpose command for copying files. It can be used to make a copy of any type of file. It also can be used to copy files to or from I/O devices.

To copy FILE1.TXT to NEWFILE1.TXT, enter:

```
CI.65> co FILE1.TXT newFILE1.TXT
```

The source file is given first, followed by the destination file. The source file descriptor can be masked to include a number of files. The destination file must not exist in this case; CO will not overwrite files unless directed by a replace duplicate (D) option.

The CO command creates the destination file with the same attributes as those associated with the source file. Some attributes in the destination file can be specifed in the file descriptor (security code in FMGR files and file size). There is a set of optional command parameters to control the copying process. These are options provided to control the way data will be transferred; these are most useful when transferring data to or from an I/O device. For more information on the CO command options, refer to the CO command description in Chapter 5 of this manual. Following are more examples of file copying entries.

To copy /SYS/REPORT1 to working directory:

```
CI.65> co /sys/report1 report1
```

Note that the destination file uses the default working directory and is not defaulted to the source file directory. To copy a file to an existing file on the working directory:

```
CI.65> co file1 masterfile d    (d is the replace duplicate option; the
                                 current masterfile is to be purged if it
                                 exists.)
```

To copy a file to magnetic tape (LU 8):

```
CI.65> co file 8
```

To copy a file to the terminal screen (LU 1):

```
CI.65> co file 1
```

Renaming Files

The RN command is used to change the name of a file (or files with the use of a file mask). It can also change the file type extension. You must have write access to the directory containing files to be renamed. To change the name FILE1.TXT to NEWFILE1.TXT, enter the following:

```
CI.65> rn file1.txt newfile1.txt
```

In this example, the file FILE1.TXT will no longer exist after this operation. The new file name cannot be an existing file in this case. Refer to the RN command description in Chapter 5 for details.

Moving Files

The MO command is used to move files from one directory to another. For example, to move file FILE1.TXT::SMITH to FILE1.TXT::JONES, you could copy the file to the new destination and then purge the original file. However, if FILE1 were an enormous file this would take a long time, not to mention the fact that there would have to be enough disc space for both copies.

You can use the MO command to move the file provided that the different directories both refer to disc space on the same LU. It is not always easy to tell if two directories are on the same LU; the MO command will find out if you ask it to move the file from one directory to another. The directories must be on the same LU. The "move" goes extremely fast, because the file data is not moved or copied. Only the directory information is changed. This is a useful technique for moving a file into or out of a subdirectory.

```
CI.65> mo /smith/file1.txt /jones/file1.txt
```

If the directories are not on the same LU, an error message is displayed and you must use the CO command to move the file.

Spooling Files

Spooling can be done with the CO or LI command in conjunction with the FMGR spool setup command sequence. Ensure that the spool system has been set up, then enter the following sequence of commands.

```
CI.65> fmgr
:sl,6,,,6
:ex
CI.65> co,<file to be spooled>,6
CI.65> fmgr
:cs,6
:ex
CI.65> _          (The file is now spooled out to LU 6.)
```

Purging Files

The PU command is used to purge a file, removing it from the directory. A group of files can be purged by means of the file mask. You must have write access to the directory containing the files to be purged.

To remove FILE1.TXT, enter the following:

```
CI.65> pu file1.txt
Purging FILE1.TXT ...[ok]
```

This removes FILE1.TXT from the working directory. The disc space that FILE1.TXT occupied is now available for use by another file, but the data is still unaffected. The PU command does not destroy the contents of a file it removes. It leaves enough information so that as long as the disc area occupied by the purged file has not been overwritten, the file can be recovered with the UNPU command. This is very useful for cases when you inadvertently purge the wrong file.

You may purge a number of files using the file mask feature. If the optional OK parameter is not specified, a prompt is displayed for each file to be purged and a Yes response is required to purge the file.

For example:

```
CI.65> pu file-.txt
Purging FILE2.TXT:::4:24 (Yes, No, Abort ? [Y]) Y
Purging FILE3.TXT:::4:24 (Yes, No, Abort ? [Y]) Y
```

If the OK parameter is specified, the prompt is suppressed and the message indicating the file is being purged will be displayed. For example:

```
CI.65> pu fil-.txt ok
Purging FILA.TXT:::4:24 ...[ok]
Purging FILB.TXT:::4:24 ...[ok]
Purging FILC.TXT:::4:24 ...[ok]
```

Be sure that the directories containing important files are write-protected. The PU command only checks the directory protection.

Unpurging Files

The UNPU command is used to restore a purged file (or files), usually immediately after the error occurs. It is effective as long as the purged file has not been overwritten.

To restore file FILE1.TXT that was purged earlier, enter:

```
CI.65> unpu file1.txt
```

There is no particular limit to the length of time that a purged file will remain recoverable. It depends on such random factors as the number of files being created and the position of the file on the disc and in the directory. Unpurging should be used immediately after an erroneous purge command. If the command returns an error message indicating that the file is irrecoverable, the file has been destroyed.

In program development, there may be several purged files with the same name. This can happen through sequences of create and purge operations, but it is relatively uncommon. In this case you can unpurge all of the files by successively unpurging one and renaming it to recover files of the same name.

Creating Empty Files

Empty files can be created with the CR command. However, most files are normally created by EDIT or other programs. The file space specified for these files is filled as soon as each file is created. The CR command cannot be used to overwrite an existing file.

To create a file called FILE1.TXT:

```
CI.65> cr file1.txt
```

You can specify various file attributes: type, size, and record length. The following examples illustrate creating empty files with these attributes.

```
CI.65> cr file.dat::system:1          (create type 1 file)

CI.65> cr /system/subdir/file.txt:::1 (create type 1 file within a
                                       subdirectory)

CI.65> cr file.mnl:::1                (create type 1 file in working
                                       directory)

CI.65> cr /system/bigger.dat:::100    (create file of 100 blocks)
```

Changing File Protection

The protection status of files can be displayed with the PROT command. Protection status of a file can only be changed by the owner of the directory containing the file.

To display the protection status of a file in the working directory:

```
CI.65> prot file.txt
directory ::DOUG
    name    prot

FILE.TXT    rw/r     (file is write protected from general users)
```

The protection status is given in abbreviations, W for write access and R for read access. The owner status is given first, followed by a slash and then the general user status.

Most files are usually assigned read access for general users and read and write access for owners. To reassign the protection status, refer to the following examples.

```
CI.65> prot report rw/          (read and write allowed for owner only)

CI.65> prot receipts rw/r       (read/write for owner;read for others)

CI.65> prot testdata.txt r/r    (read only for owner and others)

CI.65> prot memo rw/rw          (read/write for everyone)
```

To change the protection for all files in a directory, follow the example shown below.

```
CI.65> prot /data/ rw/rw        (read/write access to everyone for all
                                 existing files in directory DATA)
```

In this example, all existing files in directory DATA are allowed both read and write access. Note that the protection for directory DATA has not changed. All files to be created in that directory still follow the directory protection status.

Manipulating Directories

Directories can be thought of as system files that only the operating system is concerned with. Each directory contains information about the files which are in the directory, although the data in the file itself is not in the directory. File data is kept elsewhere on the disc volume. (Volumes are described separately in this chapter.) Directories have an initial size; they are automatically extended to hold more files as necessary. When files are purged, the directories are not truncated; the purged entries are reused when new files are added.

Subdirectories can appear in other directories in much the same way that any other file does. Directory and subdirectory names always have type extension DIR to distinguish them as directories; no other file can have a type extension of DIR. There is usually no need to specify the DIR type extension when dealing with directories because it is implied by the way the name is used. For example, the DIR type extension is not needed in the name /MAIN/SUBDIR/FILE, nor is it needed in the WD (working directory) command. (The entry /MAIN.DIR/SUBDIR.DIR is not valid; the file type extension DIR cannot be used in front of a slash.) Note that the file type extension and the delimiter (.DIR) is appended by the file system to the filename parameter which affects the 63 characters limit of the file descriptor. For example, entering /GLOB/SUB/SUB1/SUB2/FILE.TXT includes the implied ".DIR" for each of the subdirectories and the directory, adding 16 characters to the string.

Operations involving directories include directory creation, changing working directory, listing directories, renaming directories, purging directories, and examining and changing directory owner. These operations are discussed in the following sections.

Creating a Directory

Directories are created with the CRDIR command. To create directory SYSTEM, enter the following:

```
CI.65> crdir /system
```

This entry would create a global directory SYSTEM on the same disc volume as the working directory. If there is no working directory or if you want to place SYSTEM on a different disc volume, enter the following:

```
CI.65> crdir /system 12
```

This would create directory SYSTEM on disc volume LU 12. To find out what disc volumes are available, use the CL command. In this example, since you entered the command, you become the owner of directory SYSTEM. Other users are not allowed to create another directory of the same name. This directory is a global directory with the initial default protection status. Global directories have the following default protection status:

WR/R - write and read allowed for owner and read only for other users.

All subsequent files managed in directory SYSTEM will have the same protection status unless changed by the PROT command, either for the directory or individual files.

Note that the entry "crdir system" does not create a directory; instead, it creates a subdirectory called SYSTEM in the current working directory. Refer to the following paragraph for creation of subdirectories.

Creating a Subdirectory

Creating a subdirectory is similar to creating a directory. To create a subdirectory of directory SYSTEM called SUBDIR, enter the following:

```
CI.65> crdir /system/subdir
```

This would create the subdirectory SUBDIR in global directory SYSTEM. Note that the DIR type extension is not necessary. The subdirectory protection will be set to that of the directory it is being created in. If this is a global directory, then protection will be set to RW/R. The user who creates the subdirectory becomes its owner, even if it is a subdirectory of a directory the user does not own (but has write access).

The difference between specifying subdirectories and directories at the beginning of a file descriptor is that a leading slash is used for a directory while none is used for a subdirectory. For example:

```
/director/sub1/sub2/file.txt

sub3/sub4/file2.txt
```

Displaying/Changing Working Directory

The working directory will be searched first by the file system when searching for files. It is the directory used if a file is specified without any directory name. The working directory can be a subdirectory.

To examine the name of the working directory, use the WD command without any parameter. For example:

```
CI.65> wd
Working directory is ::DOUG
```

To set up a working directory or to reassign another directory as the working directory, enter the WD command with the name of the directory (or subdirectory). For example:

```
CI.65> wd games          (GAMES, a subdirectory in the current working
                          directory, becomes the working directory)

CI.65> wd /games/rules   (subdirectory RULES is a working directory)
```

If there is no need for any working directory, specify 0 as follows:

```
CI.65> wd 0
```

The effect of this command is that the first FMGR disc on the cartridge list will be used if the directory or CRN (in FMGR files) is omitted.

Displaying Directory Owner

The owner of a directory can be displayed with the OWNER command. This can be done by any users of the system. To display the owner of directory named SYSTEM, enter the following:

```
CI.65> owner /system
Owner of /SYSTEM is DOUG
```

Changing Directory Owner

The owner of a directory can be changed with the OWNER command. This can only be done by the current owner. Assuming that you created directory SYSTEM, to change its owner to JONES, enter the following:

```
CI.65> owner /system jones
```

Use this command with caution. Once the ownership is changed, you are no longer the owner and thus may not enjoy the same protection status. You may not be able to write (or read/write) into the directory and you cannot revert the ownership. From this point on, only JONES can change the ownership. The subdirectories within directory SYSTEM are not affected.

Moving Directories

The names of directories can be changed by the MO command which is especially powerful in manipulating directories. It can be used to move all files in one directory to another. For example, to change subdirectory /SYSTEM/SUBDIR into a new global directory NEWDIR, enter the following:

```
CI.65> mo /system/subdir.dir /newdir    (move SUBDIR into the global
                                         directory table and rename it
                                         to NEWDIR)
```

This changes the way you refer to all of the files in the directory as well; they must be preceded by /NEWDIR instead of /SYSTEM/SUBDIR. Directories do not have to be empty to be moved.

Purging Directories

Directories and subdirectories can only be purged by the owner when they are empty. All files must be purged or moved to another directory before purging the directory. Directories cannot be unpurged.

To purge a directory named GAMES:

```
CI.65> pu /games
```

Note that the form ::GAMES cannot be used because this is interpreted by PU as all files in directory GAMES. It will proceed to purge them if there are files in directory GAMES. If not, the message "Directory is empty ::GAMES" is displayed. You must precede the directory specification with a slash.

To purge a subdirectory called SUB.DIR under directory SYSTEM:

```
CI.65> pu /system/sub.dir
```

Displaying/Changing Directory Protection

The protection status of a directory or subdirectory can be displayed with the PROT command. Only the owner can change the protection status of a directory or subdirectory.

Following are examples of displaying protection status.

```
CI.65> prot /system                (for directory SYSTEM)

CI.65> prot /system/               (for all files in directory SYSTEM)

CI.65> prot /system/data.dir       (for subdirectory DATA)

CI.65> prot /system/data/          (for all files in subdirectory DATA)
```

To change the protection for a directory (SYSTEM):

```
CI.65> prot /system rw/rw          (read/write access for everyone)
```

To change the protection for a subdirectory (DATA):

```
CI.65> prot /system/data.dir rw/   (read/write access for owner only;
                                    read/write protected from others)
```

Finding a File

When you enter a file-referencing CI command, CI checks if a directory was specified. If you supply directory information, CI searches that directory for the file and returns an error if the file is not found.

If you do not supply directory information, CI attempts to locate the file. For all file-referencing commands except RU and TR, CI searches your current working directory or all mounted FMGR cartridges if you do not have a working directory. An error is returned if the file is not found.

When searching for files specified in the RU and TR commands, CI follows special default search sequences. By defining User-Definable Directory Search Paths (UDSP) #1 and #2, you can change the default search sequences for the RU and TR commands, respectively.

Default Search Sequence

If you do not include directory information with a RU or TR command (implied or explicit), the followingsearch sequence is used to locate the file:

- The current working directory is searched. If the file is not found, a default type extension of .RUN or .CMD is assumed and the working directory is searched again.

- If you do not have a working directory, all mounted FMGR cartridges are searched.

- If the file is still not found, global directory PROGRAMS or CMDFILES is searched, using the .RUN or .CMD default file type extension, respectively.

Defining UDSPs

User-Definable Directory Search Paths (UDSP) allow you to change the default search sequence used to find command and program files. The RU command uses UDSP #1 and the TR command uses UDSP #2.

For example, suppose you want CI to search the following directories when searching for a command file:

1. Current working directory
2. /JONES/UTILITIES/CMDS
3. /CMDFILES

The following PATH defines UDSP #2 to use this search sequence:

```
CI.65> path 2 . /jones/utilities/cmds /cmdfiles
```

The period (.) indicates that your working directory at the time the TR command is entered is to be searched for the file.

To display the contents of UDSP #2, enter the following:

```
CI.65> path 2
UDSP #2:  /JONES/STUFF [current WD]
/JONES/UTILITIES/CMDS
/CMDFILES
```

The first directory displayed, /JONES/STUFF, is the name the working directory when you entered the PATH command to display UDSP #2.

UDSP #1, which is used by the RU command, can be defined to a different search pattern. Assume you want the RU command to use the following search sequence:

1. /MINE/PROGRAMS
2. Current working directory
3. /MINE/MORE/PROG
4. /PROGRAMS

The following PATH command sets UDSP #1 to this sequence:

```
CI.65> path 1 /mine/programs . /mine/more/programs
```

Refer to the description of the PATH command in Chapter 5 for more details.

Manipulating Volumes

This section describes what disc volumes are and how they are used. It is a more advanced topic which can be skipped by first time users.

A volume is a self-contained section of a disc. Each volume is independent of any other volume; files or directories never cross volumes. Each physical disc drive consists of one or more volumes; volumes never cross physical drives. Each volume is identified by a logical (LU) number. Volumes are always identified by their disc LU number. The unit range of volume LU numbers is 1 to 63.

Each volume contains a unique set of information about what files are included. This information includes the names of all the global directories on the disc, as well as a table that tells which disc blocks have been allocated to files. This table is called a bit map, because the table is composed of bits rather than addresses or values.

Common operations performed are: mounting a volume, dismounting a volume, and listing contents of a volume. An operation that is not commonly performed is initializing a volume, making it ready for system use.

Mounting/Dismounting Volumes

Mounting a volume makes that volume and all the files on it available to the operating system. Dismounting a volume removes that volume and makes the files on it inaccessible to the system. These operations are not performed frequently except with removable media such as floppy discs, where discs must be mounted after they are installed and dismounted before they are removed.

To mount a volume, enter:

```
CI.65> mc <LU>
```

For example, to mount a volume with disc LU number 12:

```
CI.65> mc 12
```

Mounting a volume will initialize it if there is no valid data on the volume. Initializing a volume sets up information needed by the operating system, including the list of directories and the bit map for keeping track of space use.

When you mount a volume there is a chance that directory names on the volume just mounted will conflict with directory names on already mounted volumes. In such cases the duplicate directories are ignored, and the names of duplicate directories are displayed. If you need the new directories, you can rename the duplicate directories already mounted, then dismount and remount the volume.

To dismount a volume, enter:

```
CI.65> dc lu
```

For example, to dismount volume LU 12:

```
CI.65> dc 12
```

When you dismount a volume there should not be any open files, working directories, or restored programs on that disc volume. If you try to dismount a volume with one or more of the conditions mentioned, you will get an error message each time an error is encountered and the dismount command will be aborted. The DC command will show only one error at a time which means that you must repeat the command until all the errors are found. You must identify and correct all the errors separately before the dismount command can be completed. The following commands can be used to check for conditions that can prevent dismounting a disc LU.

WH,AL	(Check all RP'd programs.)
DL lu o	(Check opened files. This will list all files on the disc volume which can take a long time if there is a large number of directories and files.)
WD 0	(Remove working directory. This command must be used for every user having a working directory on that LU in a multiuser environment.)
WHOSD	(Check for any session accessing the specified directory or a directory on the specified LU as part of a User-Definable Directory Search Path (UDSP).)

Listing Volumes

The CL command is used to list the volumes that are currently mounted. The CL command has no parameters. It gives a list of two types of volumes: those mounted as described previously and those mounted as FMGR cartridges as discussed under the heading "FMGR Files" in this chapter. The unmounted volumes are not listed; use the IO command (described in Chapters 2 and 5) to find the LU number of unmounted volumes.

```
CI.65> cl
File System Disc LUs: 19 17
FMGR Disc LUs (CRN):  16(16)  20(A2)
```

Initializing Volumes

Initializing a volume prepares it for first time system use. The IN command can be used but this function is done automatically by the MC command. The IN command can be used to remove all the data on a volume without having to purge all the files. Initializing a disc volume permanently destroys any existing files, so be certain that the files on that LU are no longer needed. This command may only be used by the System Manager.

For example, to remove all data on volume 12 without dismounting it, the following entry can be used:

```
CI.65> in 12
Re-initialize valid directory [N]? y
Initializing Disc
CI.65>
```

Data Transfer to and from Devices

Data can be sent to and from an I/O device instead of a file. This can be done by replacing the file descriptor with the LU number of the I/O device. Devices that can be used include printers, terminals, magnetic tape units, and HP-IB devices. This method of data transfer should never be used with discs and Distributed System (DS) network links.

The CI commands that transfer data to or from files are CO and LI. Other CI file commands that do not deal with the data in files cannot be used. For example, the RN and PU commands do not deal with the data in the files, so they cannot be used with LU numbers.

The CO command can include an LU as either the source or destination LU, or both. When an I/O device is specified as a source, the CO command moves data until the device sends an end-of-file mark. On a magnetic tape there is an end-of-file mark; on a terminal, enter a control-D character to end the input. This character is entered by pressing the letter D and the control key (CNTL) at the same time. After entering the CO command and specifying your terminal as the source device (LU 1), all inputs are interpreted as data until you enter the control-D character.

```
CI.65> co 1 newfile.txt
```

The CO command will put everything you type into the file, even if you are trying to enter a command. The only easy way to get out of this is with a control-D, so remember to use control-D when you are using commands that read from the terminal.

The CO command is also used to send data to an I/O device. The file mask feature can be used to send several source files with each CO command. These will be sent to the device a file at a time, so they will be written sequentially. Note that both the source and destination parameters in the CO command can be LU numbers representing different I/O devices. You may even copy from the terminal keyboard to the display by entering:

```
CI.65> co 1 1
_                        (Enter CNTL-D to terminate input)
```

The LI command can also be used to list information from an I/O device. The same rules apply as when you use the CO command with I/O devices.

The file system does some special processing depending on what type of device you are using. Some devices must be used by one user at a time to get good results; for example, it would not do to have line printers or magnetic tapes with output from several different programs being interleaved. The system locks the LU to the program using it to prevent access by other programs. If another program already has the LU locked, then the second program will wait until the LU becomes available. Terminals are not locked so that messages can still get through to them.

Other special processing involves making sure that the data is transferred to or from the LU in the proper format. The system recognizes printers, magnetic tapes and terminals, and does special processing required for them. For most devices, data transfer is not a problem. If you have special devices, then a special program must be written for computer-device interface. This type of program is called a device driver. Refer to the Driver Design Manual for details.

In addition to the CI commands, most programs that use files will accept an I/O device LU number as a file. For example, EDIT can list part of a file to an I/O device. However, there are times when a program expects a disc file and in this case, a logical unit number will not be accepted. This may occur because the program wants to be able to read the data twice, or because it wants to be able to refer to the directory information for the file. I/O devices do not have file directory information.

FMGR Files

The following paragraphs present an overview of how FMGR files are handled by CI. CI file commands can be used to a limited extent for FMGR files. For example, with the proper parameters, DL can be used to list a FMGR disc and/or CI directory, and CO will copy files to and from a FMGR disc directory. Other commands that can be used with FMGR are MC, DC, CL, LI, PU, and RN. It is not possible to set your working directory to be a FMGR directory, but you can set it to zero:

```
CI.65> wd 0
```

This indicates that you have NO working directory. When you have no working directory, the file system will search for a file specified with no directory name by searching all of the FMGR cartridges in the order they are mounted (as reported by CL).

Although CI can handle FMGR files, note the following cases:

1. Names with slashes are unusable.

2. Names starting with dots or ending with dots are not acceptable. However, a single dot in character position 2, 3, 4, or 5 is acceptable.

3. The at sign (@) is interpreted as a wild card character in CI commands although a FMGR file name containing the at sign will eventually be selected.

For these files, it is recommended that their names be changed. Otherwise, only FMGR can be used to access them. If CI commands are used for FMGR files, they must observe the FMGR restrictions given here. For example, you cannot change the protection status of an FMGR file with the CI PROT command. In addition, FMGR is the only program that can initialize or pack an FMGR directory.

If you are interested in working extensively with FMGR files, refer to the FMGR description contained in the RTE-6/VM Terminal User's Reference Manual.

DS File Access (DS Only)

Systems which use the DS/1000-IV Distributed System Network can access files located on other RTE systems within the DS network. This includes FMGR files located on other systems connected to your system. The same operations used to access files on any local system can be used to access files in the DS network. The term local system (or local node) means your system and the term remote system (or remote node) means any other system connected to your system via the DS network. If your system does not use the DS/1000-IV Distributed System Network, skip the following paragraphs.

Specifying Remote Files

The DS transparency software is used to access files at remote systems. Files in a remote system can be listed and copied to and from your system; directories at a remote system can also be listed to your system. Wildcard characters can be used in the filename parameter and file masks can be used in the file descriptor. You can specify a remote file as an input to programs such as LINK, EDIT, or other utility programs.

To specify a file located in a remote system, the node number or name of the remote system is included in the file name. Each system has a node number; these numbers are explained in the DS manuals. Each system can also be assigned a node name and these names are kept in a file called NODENAMES in the SYSTEM directory. This file is used to associate node names with node numbers. The DS software uses it to build a table of names for node numbers.

The NODENAMES file contains entries of the form:

```
    * <comment>
 or
    node#  nodename
```

As an example:

```
*Test System 1   (Comment line)
1   SYS1
*Test System 2
2   SYS2
* Central Systems
3   Central1
4   Central2
```

Specify the node number (or name) by appending it to the file descriptor, separated by a '>' sign, for example:

```
    /Directory/File>sys1
 or
    File::Directory>sys1
```

This specifies a file located at the node named SYS1. The > sign must follow all other file information, including type, size and record length. The directory is not required but it is recommended in order to exclude files with identical names.

Note that the nodename delimiter is the > sign and it is a valid FMGR file name character. Any FMGR file name with the > sign anywhere except as the first character cannot be accessed. For example, the name >FILE can be used in a file specification but not A>FILE. The latter will be interpreted as file A in the remote system named FILE.

Remote File Access

If the remote system operates in the CI environment, the appropriate account log-on entry may be included in the remote file specification. The account name and password, if one is required, are specified within square brackets, e.g., [USER]. The trailing bracket is optional but is recommended for clarity. The account delimiter ([) cannot be used in a FMGR file name except as the first character. To specify a file at node 27 in the DS environment:

```
    /directory/file[user]>27
 or
    /directory/file>27[user]
```

If the USER account has a password, you must enter the password using a slash as a delimiter:

```
   /directory/file[User/Password]>27
```

Note that the password will be displayed on your terminal screen. If you enter the wrong password or log-on without it, an error message will be displayed:

```
Incorrect password
```

Upon successful log-on, you can access all files available in that account and under the same restrictions applicable to that account. You will remain logged on during the time that the file is open; you will be logged off at the remote node when the file is closed.

Files within the DS network can be transferred to and from any two nodes, local-remote or remote-remote. When transferring files from one remote system to another, two log-on entries and two nodes are required for the source and destination system. The node specification for a local system may be omitted. File masks can be used.

```
CI.65> co /mydir/@.ftn>systemA[UserA] /dir/@>systemB[UserB]
```

This example copies all FORTRAN source files from a directory in SYSTEMA to a directory in SYSTEMB. This sample entry is valid as long the systems specified (and your system) are actively connected in the DS network and the file system access rules are observed. If you are at either SYSTEMA or SYSTEMB, the local node name can be omitted:

```
CI.65> co /mydir/@.ftn /dir/@>systemB[UserB]
```

DS File Access Considerations

In accessing remote files through the DS network, keep in mind the following considerations.

FMGR files are accessible unless the file name contains one or more characters which have special meaning, such as a > or [. The DS transparency software operates from CI and other programs that use the CI file system. If your system operates strictly with FMGR, then refer to the DS manuals for all DS operations.

It is legal and useful to specify the local system in the node specification. For example, this allows you to move a file from another account on your local system. If an account name is specified without a node, the local system is assumed.

Some file names may begin with a greater-than sign (>). For example, the entry "dl /dir/>27" does not specify a remote file. To specify a remote file, use:

```
CI.65> dl /dir/@>27
```

While remote files are being accessed, system failure such as power-fail may occur. If that occurs, note the following:

1. If the remote system is down, requests to it will time out. This will cause an error return from the FMP call making the request.

2. If the remote system goes down and comes back up immediately, files that were open on that system will no longer be open, though it may appear that they are open at the local end. Accesses to such files will also get errors. These files must be closed, using the CLOSE utility described below.

3. If the local system goes down, its files will be left open at the remote system. The DS transparency software will not be able to close these files. The recommended way to close these files is to use the CLOSE utility described below.

To close open files while accessing remote files:

```
    CI.65> close /directory/file          (at local node)
 or
    CI.65> close /directory/file>node     (at local or remote node)
```

This sample entry closes a file if it is open to the DS transparency monitor TRFAS. You must specify a log-on name for the local file if one was supplied when it was opened, even if the file is in your local node.

Remote File Access Limitations

There are cases where CI file manipulation commands cannot be used on a remote system. The most common cases are:

1. Cannot use the default working directory at a remote system.

2. Cannot run a program contained in a remote file. (Although you can copy the file to the local system and then run the program.)

3. Cannot mount or dismount volumes at remote systems.

4. Cannot examine or change ownership of directories at remote system.

5. Cannot access I/O devices (such as terminals or printers) at a remote system.

Chapter 4
Controlling Programs

Introduction

This chapter explains how to use the CI commands for controlling programs. You can manipulate programs in several ways: restore them into system tables, remove them from system tables, stop programs momentarily or completely, resume execution of a suspended program, and modify the memory requirements.

Examples are given to illustrate the usage of the program control commands. In these examples, user inputs are underlined and variables are shown within angle brackets.

A brief summary of the program control commands is shown in Table 4-1. Refer to Chapter 5 or the RTE-6/VM Terminal User's Reference Manual for a full explanation of these commands.

Program Identification

There are many different programs provided with RTE-6/VM to support a variety of tasks. These programs can be run from CI. Programs are scheduled by name, along with a program runstring that may include program parameters. The program name consists of up to five characters and it must begin with a letter. If a program file with a file name of more than five characters is specified in the run command, only the first five characters are used as the program name.

The RTE system manages execution of programs by means of identification (ID) segments. Before a program can be executed, it must be assigned an ID segment. The ID segment identifies the program and the location of its associated program file. It also maintains information such as program size, status and priority. The ID segment may be released at the end of program execution or can be established permanently with the RP command and removed with the OF command.

Table 4-1. Program Control Commands

Command	Task
AS prog <part #>	Assign partition
BR[prog]	Break program execution
GO[prog[parm*5]]	Resume suspended program
IT prog[res[mpt[hr m sec ms]]]	Set execution time
OF[prog[ID]]	Remove program
ON[prog[NOW[parm*5]]]	Schedule a program
PR prog[priority]	Display/modify program pri.
RP file[prog]	Restore program
[RU]prog[parm*5]	Run program with wait
SS[prog]	Suspend program
ST[prog/part #/0]	Display program/part status
SZ prog[size[mseg size]]	Display/Specify program size
VS prog[lastpg]	Display/modify virtual EMA size
WS prog[wrksz]	Display/modify VMA working set size
XQ prog[parm*5]	Run program without wait

Program Priorities

Each program has an assigned priority. When you schedule a program for execution, the system may not execute your program immediately depending on the priority of your program in relation to that of other scheduled programs.

Priority is an attribute assigned to all programs to indicate their importance. Program priority is in the range of 1 to 32767, lower numbers indicating higher priorities. If two programs are scheduled to run at the same time, the higher priority program will be run first. In addition, programs with equal priorities may be timesliced to appear to run concurrently. Program priorities can be changed interactively as explained later in this chapter.

Running a Program

A program may be run from CI by using the RU command. For example, to run the editor program (EDIT), you would enter:

```
CI.65> ru edit
```

In CI, RU is an implied command, meaning that it is not necessary in the command runstring. Therefore, the editor may also be run by entering:

```
CI.65> edit
```

CI will assume that the RU command was intended anytime a non-CI command is entered. The remaining examples in this manual will use the implied RU command.

As you run the editor program, you may want to specify a file to be edited. The editor has been written to accept a filename parameter in the runstring. For example:

```
CI.65> edit,prog.ftn
```

Program EDIT will also accept, as a second parameter, a command to be entered after opening the file. The entry

```
CI.65> edit prog.ftn s
```

will run the editor, open file PROG.FTN, and execute the editor S command (enter screen mode).

Parameters are accepted by other programs such as LINK, FTN7X, and Macro. These are described in their respective manuals. User programs may be written to accept up to five parameters from the runstring or one long character string up to 80 characters including delimiters. This facility is described in the RTE-6/VM Programmer's Reference Manual.

Program Execution

Upon receipt of a RU command, the system will search for an existing ID segment for the program specified or create one for that program. Then the program is scheduled to run by having its ID segment placed in a list of programs ready to execute. The system will dispatch programs from this list in order of their priority.

Program CI will be suspended to allow interaction between the program and the user's terminal. When the program terminates, CI will again issue its prompt and accept commands. This cycle is known as "run with wait".

Sometimes it is desirable to allow a program to run while continuing CI interaction. This may be the case for lengthy programs that require no user interaction. The XQ command will schedule a program to run and return control to CI. Its use is described in the following section.

Running Programs with Wait

To start a program with wait, enter the program name after the CI prompt.

```
CI.65> edit
```

Program CI first checks that this is not a CI command. If the program name is the same as a CI command, precede the program name with CI run command RU. For example, to run a program called OWNER, enter:

```
CI.65> ru owner
```

If the program has been restored (i.e., has been assigned an ID segment), CI will execute it. After the program terminates, it remains restored. If the program has not been restored, CI will restore it, execute it, and release its program ID segment at completion.

Special processing occurs when a program file needs to be restored. When CI looks for a program file, it uses the name and directory specified. If only the program name is specified, CI first searches for a restored program, next for a file in the working directory, and then for a file in a system global directory called PROGRAMS. The following example illustrates how CI searches for programs.

When the command EDIT is entered, it is examined by CI and interpreted as a program since it is not a command. CI searches for an ID segment restored for EDIT. If one is found, CI will run EDIT using that ID segment.

If there is no restored EDIT, CI scans the working directory for a file named EDIT or EDIT.RUN. If one is found, CI allocates an ID segment for that program file and executes it. If EDIT is still not found, CI searches for EDIT.RUN in directory PROGRAMS.

If there is no working directory (such as after a "wd 0" command), CI scans all FMGR cartridges in the same way FMGR searches for files. If unsuccessful, CI then searches for /PROGRAMS/EDIT.RUN.

The above program search sequences apply to the RU, XQ, IT and RP commands, as well as to scheduling operations done by other system programs such as EDIT. The program search sequence does not apply to other CI commands. For example, entering "li edit.run" will not find EDIT.RUN unless it is in the working directory. You must specify the directory (or FMGR cartridge) where EDIT.RUN resides.

Specifying the directory/subdirectories allows CI to skip the search sequence and proceed directly to the file. Entering /DIRNAME/EDIT will allow CI to find EDIT quickly in directory DIRNAME.

One way to make use of the program search sequence is in program development. Since your working directory is searched first, you can have your own version of any program in the working directory, leaving the unmodified version in directory PROGRAMS where it is accessible to other users.

Program CI can handle cases where there are two or more programs scheduled with the same name. This might happen because of several situations: several copies of a program may be running at the same time (e.g., EDIT may be run by several users); shortening of two different file names may lead to the same 5-character program names (e.g., DATALATCH.RUN and DATALOGGER.RUN). CI handles these situations by changing the names of the duplicate programs, replacing characters four and five of the program name with a .A, .B, .C, etc. For example, the second copy of EDIT becomes EDI.A, the third copy EDI.B, the fourth copy EDI.C, and so on. This process is known as "cloning a program". In a multiuser environment, programs are also identified by a session number. The session number is described in Chapter 2 of the RTE-6/VM Terminal User's Reference Manual.

Running Programs without Wait

To run programs without wait, the XQ command is used. The XQ command starts the program specified and returns control to you, indicated by the CI program prompt. For example:

```
CI.65> xq prog        (scheduling prog without wait)
CI.65> _              (prog executing; CI back in interpretive mode)
```

The XQ command is NOT recommended for use with interactive programs. It is best used for programs which take a long time to run, and which will not require any user intervention.

You can run several programs at the same time using the XQ command. This command works the same way as the run command, including restoring the program and changing the name if necessary. If you start a program with XQ and that program is already running, a message will be displayed to report that the program is busy. CI returns to the interactive state with the CI.65> prompt. To run a program without wait:

```
CI.65> xq /testdata/subharmonics.run
CI.65> _
```

Any errors reported during the program execution will be displayed at the terminal as well as any completion message. The WH command can be used to check the status of the program scheduled with the XQ command. Refer to Chapters 2 or the RTE-6/VM Utility Programs Manual for a full description of the WH command.

Time Scheduling Programs

To schedule a program to start at a later time (up to 24 hours), the IT (execute time) command is used. The IT command runs a program at a particular time based on the processor time-of-day clock after being scheduled by the ON command. It operates in the same way as the XQ command, except for the time delay. For example, to run program CLOCK with parameters a, b and c at noon:,

```
CI.65> it clock,,12
CI.65> on clock a b c
CI.65> _
```

At 12:00, program CLOCK will run once. The IT command handles ID segments and program naming in the same way as in the RU and XQ commands.

The time can be specified in 24-hour format; 1:30 pm is 13,30. Seconds and milliseconds are optional. The maximum time delay is 24 hours. If at 4:05 pm you specify the program start time as 4 pm, the program will run at 4:00 pm tomorrow, rather than running immediately.

You can also use IT to start a program and subsequently run that program repeatedly at some time interval. To run program CLOCK at one-hour intervals in the above example:

```
CI.65> it clock 4 1 12 30
CI.65> on clock a b c
```

The time interval for repeated execution can be specified as hours, minutes, seconds, or tens of milliseconds, the first parameter is 4, 3, 2, and 1 respectively. The next parameter is a multiplier, e.g., one hour would be specified as "4 1" and 30 minutes, "3 30".

To remove program CLOCK from the time list, enter the following:

```
CI.65> it clock
```

The IT command syntax is shown in Chapter 5. Refer to the RTE-6/VM Terminal User's Manual for a complete description of the IT and ON commands.

Restoring Programs

Restoring a program is a process that assigns to the program file an ID segment in a system table that keeps track of programs to be executed. The ID segment contains information necessary to run the program: the 5-character program name, its location on disc, scheduled run time, priority, partition assignments, and other information required by the operating system. CI commands that affect these program attributes cannot be used until the program has been restored.

A program can be restored in one of two ways. The most common way is an implicit restoration through the use of a RU, XQ, or IT command where no ID segment has been allocated for that program. The ID segment is released upon program termination. A second way is to explicitly restore a program with the RP command. The program will be allocated an ID segment but will not be scheduled for execution. The ID segment is permanently assigned until removed with the OF command.

For example, to restore program TEST.RUN, enter:

```
CI.65> rp test.run
```

The program can now be run using the RU, XQ, or IT command and the ID segment will remain allocated upon program termination.

If a second user tries to run a program restored with the RP command, an error message will be issued indicating that the program is busy. The second user may wait for the program to finish or may use a second parameter in the RP command to create another ID segment with a new program name:

```
CI.65> rp test.run test2
```

The second user may now use the RU, XQ, or IT command with program TEST2. Note that the second program name must be five characters or less. This method is not required for programs that were previously restored implicitly because the system will automatically create a new name in this case (described in the Running Programs With Wait section). RP'd programs that are not running will be OF'd when the user logs off.

Removing Programs

The OF command is used to remove a program. To remove a program restored by the RU, XQ, or IT command, enter:

```
CI.65> of <program name>
```

If the program was not restored with RP, then its ID segment will be released. If the program was restored with the RP command, only the execution is terminated. The program ID segment remains intact. To free the ID segment of this program, the second parameter, ID, is needed. For example, to remove the ID segment of program TOWER that was restored with the RP command:

```
CI.65> of tower id
```

The OF command (with or without the ID parameter) stops an executing program abruptly. Stopping a program in this way terminates the program execution without performing the normal clean-up operations. This command is normally used to stop a program in trouble. Any I/O operation in progress is terminated (any system resources used are returned). Data being written to a file is not posted, possibly leaving the file in an abnormal state.

Break Program Execution

To stop a program in an orderly manner rather than abruptly as with the OF command, the BR command can be used. This command can be entered when you do not want to wait for a program to finish. If the program was scheduled with wait (RU command), you must first interrupt the system and obtain the break mode prompt. If the program was scheduled without wait (XQ or IT command), the BR command can be issued from CI. This command can be entered with or without a program name. For example:

```
CI.65> test2
<press any key>
S=65 COMMAND?br test2
CI.65> _
```

The program name must be the same name as reported by the WH command because CI may have made up the name to avoid having duplicate names. The BR command can be used without the program name to break the program most recently run without wait. Refer to Chapter 4 of the RTE-6/VM Terminal User's Reference for details.

For this command to work, a program must acknowledge the break bit in the ID segment using the system call IFBRK (refer to the Programmer's Reference Manual). This is implemented in all system programs but not necessarily in user programs. If BR does not halt the program, you must wait till the program finishes or use the OF command.

Suspending Program

Another method of stopping an executing program is to suspend the program with the SS command. This command does not adversely affect the program or open file status; it simply suspends execution. The SS command is used the same way as the OF or BR command. However, the suspended program can be restarted with the GO command or terminated with the OF command.

The SS command will not interrupt any I/O operation in progress. It will wait until the I/O operation is finished. Sometimes this may take a long time and there will not be any message while CI is waiting.

Resuming Execution

Suspended programs can be restarted with a GO command. It is used the same way as the OF, BR and SS commands. The suspended program will be restarted at the point of suspension. For example:

```
CI.65> xq test2
CI.65> ss test2
CI.65> go test2
```

Normally, the GO command can be entered without any program name to resume the currently suspended program. The System Manager can resume programs of other system users by specifying the name of the suspended program.

Changing Program Priorities

All programs running under RTE-6/VM have a priority number which is recorded in the respective program ID segments. The priority number can be assigned when the program is written or when it is linked. It may also be changed dynamically with the PR command as shown below.

The priority number may be in the range of 1 to 32767, with smaller numbers representing higher priorities. Typical values for user application software would be in the range of 50 to 200. Higher priority real-time and system programs may be in the range of 1 to 40.

A primary task of the operating system is to run the highest priority executable user program followed by the next highest and so on. When there are programs of the same priority, a technique called "timeslicing" is used. Programs of the same priority share the processor by having small intervals (or slices) of time allocated to them by the operating system in a round robin fashion. Timeslicing need not be implemented for all programs. A value called the timeslice fence is established at system generation time to set the priority below which timeslicing will be implemented.

If a user program has an unduly long elapsed running time in a busy system, or if it does not run at all, then its priority may be too low. On the other hand, if it runs to the exclusion of other user programs, then its priority may be too high.

To change the priority of a program, use the PR command. For example:

```
CI.65> pr test2 50
```

Program priorities should be handled with caution. If you have a program with a very high priority, it may run continuously and prevent other programs from executing indefinitely.

Changing Memory Requirements

Some programs may require dynamic memory allocation as is the case with reentrant routines or Pascal recursive procedures and dynamic data structures. Memory requirements may vary depending on input parameters or data given to the program. The operating system will not be aware of these factors and may not allocate enough memory to the program unless explicitly instructed to do so. You can change the amount of memory allocated to a program in two ways. One way is to use LINK to make sure the program will get extra memory every time it runs. This is described in the LINK manual. The other way is with the SZ command. This command is used after the program has been restored but before running it. For example, to change the memory allocation of DATALOGGER to 20 pages:

```
CI.65> rp datalogger
CI.65> sz datal 20      (Note the 5-character program name)
```

Now DATALOGGER will have 20 pages. The new memory partition allocation remains in effect as long as DATALOGGER is restored. If it is removed, it will revert to that defined by LINK. The SZ command cannot be used for a program that is executing.

If a program uses EMA, the SZ command modifies the EMA data space only (in the range of 2 to 1022 pages). For example:

```
CI.65> rp emapr
CI.65> sz emapr 300   (changes the EMA space of EMAPR to 300 pages)
```

The size of a program can be displayed with the SZ command. This is done by entering the SZ command without parameters. For example:

```
CI.65> sz proge

65211  32  32
```

minimum required partition size in pages
program size
address (last word 1) of the program

Assigning Partitions

The system memory is divided at bootup time into dynamic and reserved partitions. Normally, when a program is run, it is assigned memory as required from the dynamic memory. Reserved partitions are partitions of fixed sizes that can be reserved for specific programs. You can assign a reserved memory partition to a program with the AS command. The reserved partitions available can be checked with the WH,PA command.

For example, to assign PROGA to partition 1 which was previously created in the system, enter:

```
CI.65> as proga 1
```

Program PROGA must be restored and must not be running.

When it is no longer necessary for a program to run in a reserved partition, you can remove the designation by using the AS command again, assigning the program to partition 0. There is never a partition zero; this number is used to remove the assignment. For example, to reassign PROGA to run in dynamic memory, enter:

```
CI.65> as proga 0
```

Changing Virtual Memory Area

VMA programs are those that utilize an RTE feature which enables execution of programs requiring a very large amount of data storage. The data for a VMA program is contained in an area on disc called the Virtual Memory Area (VMA). The portion of data being processed is moved from disc to an area in memory called the Working Set (WS) so data is being transferred between VMA and WS as necessary during program execution.

The WS size and the VMA space (VS) are defined using LINK, from 2 to 1022 pages of WS and up to 65535 pages of VS. These can be changed with the WS or VS commands respectively.

It may be desirable to change the size of WS and VS with LINK, because this changes it permanently. The other way to do this is with CI commands after restoring the program. The WS and VS command can also be used to find out the area defined. For example:

```
CI.65> rp datalogger
CI.65> ws datal          (displays working set area)
20
CI.65> vs datal          (displays virtual EMA area)
3000
```

To change the WS and VS areas of a program:

```
CI.65> ws datal 45       (change working set size to 45 pages)

CI.65> vs datal 2500     (change VS size to 2500 pages)
```

The change made with the WS or VS command is effective as long as the program ID segment is in memory; when the program ID segment is released, the size will revert to that defined at program link time. Refer to Chapter 5 for more information on the WS and VS commands.

Chapter 5
CI Command Descriptions

Introduction

This chapter contains descriptions of all CI commands. The commands are described in alphabetical order. A tutorial of most of these commands as well as a command summary has been given in previous chapters. Commands or capabilities that are specific to the System Manager are indicated as "superuser".

AG (Modify Partition Priority Aging)

Purpose: Modifies the rate a partition's priority number is increased and turns on or off partition priority aging.

Syntax: AG numb/of

number — Number of 10-millisecond intervals to be used as the aging rate. This value must be in the range of 10 and 32767.

of — Turns off partition priority aging.

Description:

Partition aging is a feature that allows high-priority suspended (state 3) programs to be swapped out, replaced by lower priority programs. Details of the AG command are given in the RTE-6/VM Terminal User's Manual, Chapter 4. Increasing the priority number of a program lowers the program priority.

Examples:

```
CI.65> ag 100      (Increase the partition priority number by two
                    every second.)

CI.65> ag of       (Turn off all partition priority aging.)
```

AS (Assign Partition)

Purpose: Assigns a program to a reserved partition.

Syntax: AS prog <part #>

prog	Program name, up to five characters, session identifier optional.
part #	A number that identifies the partition to which the named program will be assigned. Partition number = 0 removes the current assignment.

Description:

The AS command is identical to the SYSTEM AS command. Refer to Chapter 4 of the RTE-6/VM Terminal User's Reference Manual for a complete description.

Examples:

CI.65> as test2 2 (assigns program TEST2 to reserved partition 2)

CI.65> as test 0 (program test to run in any partition)

BL (Examine or Modify Buffer Limits)

Purpose: Allows the general user to examine the current buffer limits and a System Manager to change the current buffer limits.

Syntax: BL[lower limit[upper limit]]

lower limit — Used by the System Manager only; specified in number of words. If upper limit is changed and lower limit is not specified, it defaults to 1.

upper limit — Used by the System Manager only; specified in number of words. If lower limit is changed and upper limit is not specified, it remains the same as the existing upper buffer limit.

Description:

The BL command is identical to the SYSTEM BL command. Refer to the RTE-6/VM Terminal User's Reference Manual for a complete description.

Examples:

```
CI.65> bl              (Displays lower and upper buffer limits)
100 400

CI.65> bl 200 500      (Change buffer limits)
```

BR (Break Program Execution)

Purpose: Sets a flag which may be interrogated by a program.

Syntax: BR[prog]

prog — Program name, up to five characters.

Defaults to the last scheduled program.

Description:

The BR command is identical to the SYSTEM BR command. Refer to the RTE-6/VM Terminal User's Reference Manual for a complete description.

CL (List Mounted Discs)

Purpose: Displays all mounted disc volumes.

Syntax: CL

Description:

The CL command is used to show the mounted disc volumes by their logical unit numbers. It lists separately all LUs mounted as file system LUs and all LUs mounted as FMGR LUs. For the FMGR system discs, the LUs and the associated CRNs are listed in the FMGR search order.

Examples:

```
CI.65> cl
File System Disc LUs:  54  56
FMGR Disc LUs (CRN):  27(DB)      45(TY)      46(PM)      30(XX)
 61(SO)      59(GR)
```

CN (Control Device)

Purpose: Controls peripheral devices.

Syntax: CN lu function[parm*4]

lu — Logical unit of device to receive the control request.

function — The control function code (0-63B) as defined in the function field of CNTWD (listed for each driver in the appropriate driver reference manual), or a two-character mnemonic code from the following:

Mnemonic Code	Equivalent Octal Function Code	Parm 1-4 Definition	Action
TO	11B	# lines on page	Issue top-of-form or line spacing on printer
RW	4	None	Rewind cassette tape
EO	1	None	Write end-of-file
FF	13B	None	Forward space file
BF	14B	None	Backward space file
FR	3	None	Forward space record
BR	2	None	Backward space record

For magnetic tapes and cassette tapes, the function parameter defaults to rewind tape, for printers, form feed.

parm*4 Optional parameters that specify additional device details as appropriate for a given driver. Specific meanings for each parameter may be found in the appropriate driver reference manual for each driver.

Examples:

CI.65> cn,4,rw (rewinds the tape in cassette tape unit, LU 4)

CI.65> cn,6,to,-1 (causes a top-of-form, page feed, on printer LU 6)

Refer to the appropriate driver reference manual for full information on the control requests that can be issued for each driver.

CO (Copy Files)

Purpose: Copies one or more files between directories and or I/O devices.

Syntax: CO <file1/lu> <file2/lu>[parm]

file1/lu The source file descriptor or the LU number of an I/O device. (Refer to the CR command syntax description for the definition of file descriptor.) May be masked to operate on more than one file. (Refer to the File Mask section in Chapter 3 for the mask syntax.)

file2/lu The destination file descriptor or the LU number of an I/O device. May be masked to allow the system to generate destination names (refer to File Masks description in Chapter 3). When copying from a device, the default file type is type 6; different file type desired must be specified.

parm The following characters indicate particular actions to be taken. Defaults to A. These include:

A	ASCII records, no checksum.
B	Binary absolute.
C	Clear backup bit.
D	Replace duplicates; existing file with the same name will be replaced.
N	No carriage control in source
P	Purge source after copying.

Description:

The CO command can be used to copy a group of files from one directory to another. Masking the file1 parameter allows matches of a number of files. If a wildcard character is used in the name field of file1, an appropriate destination mask must be used to default destination file names.

The file mask is a very powerful tool but complicated; it should be used with caution. For example, you can copy all type 6 files on several different directories to a particular directory. That directory can be a global directory or a subdirectory. An implicit "d" qualifier is used whenever copying with a wildcard mask. This means that if any directory matches the mask, then all files in that directory will also be copied. This can be overridden with the mask qualifier "n". Qualifier n is particularly useful with time qualified copies since directory time stamps are not maintained. Note that the d qualifier is automatically appended to the unspecified mask and will appear in error messages. For example:

```
CI.65> co /global/@.ftn /new/@.ftn
No such directory @.FTN.D::GLOBAL       (d appended to file name)
```

When copying a file from one directory to another, the creation and access times will be that of the copying process. However, the update time of the new file will be that of the current file to maintain a history of the latest revision date.

The file type of the destination file is the same as the source file if unspecified. If the destination file size is unspecified, a size will be selected to eliminate extents. The protection of the destination file will be the same as the source file if the source is not an LU or a FMGR disc cartridge. Otherwise, it will have the protection of the directory into which it is copied.

Examples:

```
CI.65> co @.src.e /backup/archive/source/@.@
```

This example copies all files with file type extension .SRC on all accessible directories to subdirectory SOURCE of subdirectory ARCHIVE of directory BACKUP. Their names and file type extensions will be unchanged.

```
CI.65> co @.rel 8 b
```

This example copies all files with type extension REL on the working directory to LU 8. Note that this example shows that CO can be used to copy to an I/O device. The preferred method is to use the TF utility for this type of copying.

CI.65> co 8 /programs/program.run:::6:1000

When copying from a device (such as a tape unit), the default file size is 24 blocks. If the file is longer and extents are not desirable (i.e., type 6 files), a longer file size must be explicitly specified. After copying, the file will be truncated to its actual size.

CI.65> co @.dir.d sub/@.@ (SUB.DIR is in the working directory)

This command will find subdirectory SUB in the working directory and copy it into subdirectory SUB, creating file SUB/SUB.DIR. Then following the d directive, all files in subdirectory SUB will be copied, including SUB/SUB.DIR. This will continue until the string reaches 63 characters. To avoid this unpleasant effect, either do not copy from a directory into it's own subdirectory or use the n qualifier to disqualify copy subdirectories.

CR (Create File)

Purpose: Creates a disc file.

Syntax: CR <file descriptor>

For Remote Access (DS Only):

CR filedescriptor[user]>node

filedescriptor — File descriptor; up to 63 characters. Can be any of the following:

Standard /dir/[subdir/]filename.typex:::[type[:size[:rlen]]]

Combined [subdir/]filename.typex::dir[:type[:size[:rlen]]]]

FMGR filename:[sc]:dir/crn[:type[:size[:rlen]]]]]

where:

dir — Specifies the unique (global) directory for the file. The directory name can be up to 16 characters long, not counting delimiters (slashes). If omitted, the working directory is used.

subdir — Specifies one or more subdirectories for the file, separated by slashes (/). Each subdirectory can be up to 16 characters long not counting delimiters. Any number of subdirectories can be specified with the limit of 63 characters for the full file descriptor.

filename — Specifies the name of the file; can be up to 16 characters. Mask characters (@ and -) can be used to specify a group of files; @ masks any one or more characters and the dash (-) masks one character position for any character except a blank. Only the first 6 characters are valid for FMGR files; other characters, the type extension, and qualifier options are ignored.

typex — A file type extension field appended to filename with a period as the delimiter; can be up to four characters used to describe the type of information in the file. The @ and dash (-) mask characters can be used in the typex field. Standard file type extensions are:

.cmd	command file
.dat	data file
.dbg	debug file
.dir	directory or subdirectory entry
.ftn	FORTRAN source file
.lib	indexed library of relocatables
.lod	LINK loader command file
.lst	listing
.mac	Macro source file
.map	loader map listing
.pas	Pascal source file
.rel	relocatable (binary) file
.run	program file
.snp	system snapshot file
.stk	command stack file
.txt	text file

type — A number used to indicate how the file is organized. Standard types are:

1 — Type 1 files are random access files which do not have any structure information in them. They can be read and written very quickly, but they are not suitable for use as text files. Fixed length records are 128 words long.

2 Type 2 files are fixed-length record, random access files. The record length is defined when the file is created. They are not suitable for use as text files.

3-7 Type 3 and above files are variable length record, sequential files. They are suitable for use as text files. There is no difference in the handling of file types 3 and above. By convention, types 5, 6 and 7 are used for relocatable object, executable program and absolute binary files, respectively.

If type is not specified, 3 is used. Types greater than 7 are user defined.

size Specifies the file size in number of blocks. Default is 24 blocks.

rlen Specifies the record length in type 2 files in number of words.

user Optional parameter used in Distributed Systems to specify the user account under which this file exists.

node Optional parameter used in Distributed Systems to indicate the node where the file resides.

crn Used in the FMGR compatible format only; can be a positive number (cartridge reference number CRN), or a negative LU number, or two characters.

Description:

The CR command creates an empty file. The minimum information which must be specified is the name. The remaining parameters can be defaulted. Default values are:

```
file type extension: blank
directory:           working directory
type:                3
size:                24 blocks
```

To create a file, you must have write access to the directory where the file will reside. The owner of this file is the owner of the directory. The protection status of this file is the same as that for the directory it is on. This allows you to write into a file or create a file on another directory that you have write access. Only the owner of the directory can alter the protection status of the file thus created.

Examples:

CI.65> cr /applications/documentation/compiler

This example creates an empty file called compiler with the following attributes: blank type extension, size = 24, type = 3, on subdirectory DOCUMENTATION on global directory APPLICATIONS.

CI.65> cr /joe/notes.txt:::4:10

This example creates file NOTES.TXT with the following attributes: file type 4, size = 10 blocks, on directory JOE.

CI.65> cr data.dat:::2:5:18

This example creates file DATA.DAT as a type 2 file with 5 blocks and a record length of 18 words in the working directory.

CI.65> cr notes/project.txt

This example creates file PROJECT.TXT on subdirectory NOTES on the current working directory. The default attributes are used: type 3, 24 blocks.

CRDIR (Create Directory/Subdirectory)

Purpose: Creates a global directory or a subdirectory.

Syntax: CRDIR directory[lu]

directory — A character string that identifies the directory. It can be up to 63 characters and can be either a global directory or a subdirectory. The directory in which a subdirectory is created must already exist.

The name can include an optional size subparameter specified in number of blocks as follows:

<directory>::::<size>
(/jones::::24)

Default size is equal to the track size of the disc used, typically 48 or 64 blocks for hard discs and 30 or 16 for flexible disc. Directory size is extended as needed.

lu — Specifies where to place a global directory. It must be a mounted disc volume. If set to zero, the disc volume of the working directory is used. This parameter is ignored for subdirectories, which go on the same volume as the directory in which it resides.

Description:

The CRDIR command creates a directory or a subdirectory. A subdirectory can be created within a subdirectory. There is no limit to the level of subdirectory nesting except for the 63 character limit to any file descriptor.

If the optional disc volume parameter is omitted and there is no working directory, the lowest numbered disc volume is used.

The size of the directory can be specified in the same way as in a file creation. There are four directory entries per block, and two directory entries used for internal information. Thus if a size of four blocks were specified, the directory could hold 14 file entries (extents require additional entries) before the directory had to be extended. As is the case with files, extents slow directory search performance. The created size is not a limit on the number of entries in a directory. Some programs assume that directories contain no more than 32767 files.

If a directory is created with the same name as an FMGR CRN, the FMGR disc cartridge cannot be accessed by any CI command unless the working directory is set to 0.

Examples:

crdir jones	(Create subdirectory JONES in the working directory.)
crdir jones::::12	(Create subdirectory JONES in the working directory with 12 blocks.)
crdir smith/jones	(Create subdirectory JONES on subdirectory SMITH in the working directory.)
crdir /smith/jones	(Create subdirectory JONES, in global directory SMITH.)
crdir jones::smith	(Create subdirectory JONES in global directory SMITH.)
crdir ::HP	(Create global directory HP on the same LU as the working directory.)
crdir /HU	(Create global directory HU on the same LU as the working directory.)

CU (CPU Utilization)

Purpose: Displays a bar graph of CPU display registers showing the percentage of CPU utilization.

Syntax: CU on/off

on Turns display on.

off Turns display off.

Description:

The CU command is identical to the SYSTEM CU command. Refer to the RTE-6/VM Terminal User's Reference Manual for a complete description.

DC (Dismount Disc Volume)

Purpose: Dismounts a disc volume.

Syntax: DC lu

lu The positive LU number of the disc volume to be dismounted.

Description:

The DC command dismounts a disc volume, making the global directories on that disc inaccessible. If there are any open files, working directories, or active type 6 files (or the swap file), an error message will be displayed and the LU specified will not be dismounted. The first problem encountered will cause the error message. If there are more problems, it may take several tries to discover and correct all of them.

For SYSTEM disc volumes, use the SYSTEM DC command because it provides more information when there are active programs. If the dismount fails on a FMGR disc cartridge, the disc will remain mounted but will move to the bottom of the volume list.

DL (Directory List)

Purpose: Lists files in a directory.

Syntax: DL[mask[options[<output file/lu>[msc]]]]

mask — A field specifying the names of files matching the mask to be displayed. Default is all files in the working directory.

The file mask field can include any or all of the file descriptor parameters and a mask qualifier appended to the filename parameter. Refer to the File Mask section in Chapter 3 for the file mask syntax description.

options — Parameters that specify what particular information from the directory will be displayed. They can be listed without any delimiters and may be in any order.

A Time last accessed displayed in following format:
Wed Jun 30, 1982 9:55:48 am

B files that have not been backed up to be marked with asterisk (*).

C Creation time displayed in same format as option a.

E File type extension (for sorting only).

F File type.

L Location of file; displays the block address and LU of the main file entry. The first block on disc is address 0.

M Main file size in blocks, excluding extents.

N Number of records.

O Mark open files by displaying the name of the program that has the file open next to that file. If there are no open files, this field will not be displayed.

P Protection on file displayed in following form: owner/other (rw/r); read and write abbreviated to first letter.

R Record length; gives length of longest record in the file in words.

S Size; total number of blocks used by file, including extents.

T Temporary file marked with asterisk (*).

U Time last updated displayed in same format as option a.

W Words in file, up to EOF.

X File with extents to be marked with an asterisk (*).

Y Security code (FMGR files only).

* A useful subset of the above (fwnsxp).

! All of the above.

+ Ascending sort by item specified.

- Descending sort by item specified.

output file/lu An optional file or LU where the DL output is to be stored.

msc The master security code for the system; used only for FMGR files and only needed if the Y (or !) option is specified.

Description:

The DL command displays a list of the files which match the specified mask in a directory or subdirectory. The display format is as many names as possible per row if no options are specified. If any display option is specified, the format will require one line per file. If several options are specified, multiple lines per file may be required.

The display is normally sorted by name. Sorting can be by means of the two sorting options, "+" for ascending order and "-" for descending order sort. Preceding an option specifier with "+" will cause the list of files to be sorted with the lowest value first, "-" will cause the reverse. If either "+" or "-" is specified and not followed by an option specifier, then the names are ascending or descending sorted. The default is an ascending sort by name. The number of files which can be sorted depends on the amount of free memory the program has.

If there are too many files, as many as possible will be sorted and displayed, then another list of files will be sorted and displayed until all the files have been displayed. Sizing the DL program scheduled by CI to a larger size will increase the number of files that can be sorted at one time.

Some of the information in the directory is dynamic and may not always be accurate, particularly if a file is open or if the last program which accessed that file failed to close the file. This information includes access time, total size, time last updated and words in file. These fields can be specified with the options a, s, u, and w respectively. Note that for FMGR files, only the options f,l,m,o,r and y will be displayed; other fields are not maintained in the directory for FMGR files.

The E option is used only for sorting because the file type extension is always displayed. If specified with the + or - option, the files are sorted by type extension and file name The E option is ignored if specified without a + or -.

For FMGR files, the master security code parameter is needed only if the y option is specified. If an incorrect master security code is entered, none of the security codes will be displayed. Note that if the master security code is zero, then any value (or no value) can be entered for the msc parameter. If necessary, see your System Manager for the system security code.

Examples:

CI.65> dl (display all files in the working directory)

CI.65> dl @.dir (display all subdirectories on the working directory)

CI.65> dl,a@..c83,-s (display files which start with a, and were created during 1983, sorted in descending order by size)

CI.65> dl /program/ (display all files in directory PROGRAM)

CI.65> dl /joe/foo (display file FOO in directory JOE)

CI.65> dl @.txt +s (display files with file type extension TXT on the working directory, displaying the size in number of blocks sorted in ascending order)

CI.65> dl /joe/f@.@.sc80-83 (display files in directory JOE that start with f, have any file type extension, and were created during 1980 through 1983. The s option in the mask qualifier directs a search of all subdirectories of directory JOE for similar files)

CI.65> dl /joe/@.dir (display all subdirectories in JOE)

CI.65> dl,@::sc y,,hp (display all files on CRN SC with their security codes; msc is HP)

```
CI.65> dl,,!

directory DEMO
    name   ex ba tmp  prot type msize blks words recs rlen addr/lu

COPY.REL       *     rw/r     5    86   86  6312  127  128 8390/38
create time   Wed Jan 12, 1983  9:16:13 am
access time   Wed Jan 12, 1983  9:47:09 am
update time   Wed Jan 12, 1983  9:39:47 am

COPY.SRC     *  *     rw/r     4    92  184 13418  820   38 7908/38
create time   Wed Jan 12, 1983  9:00:33 am
access time   Wed Jan 12, 1983  9:44:29 am
update time   Wed Jan 12, 1983  9:30:35 am
```

The above example gives a complete directory listing of the working directory with two files. The display columns of those shown above and those in the O and Y options are:

ex - extent; * indicates file has extents (x option)

ba - backup; * indicates file needs to be backed up (b option)

tmp - temporary; * indicates file is a temporary file (t option)

prot - protection; shows file access for owner/other (p option)

type - file type (f option)

msize - size of main file (m option)

blks - size of file in blocks (both main and extents) (s option)

words - number of words up to the end-of-file mark (w option)

recs - number of records in the file (n option)

rlen - length of longest record in file (r option)

addr/lu - block address and LU of beginning of file (l option)

open - name of program (if any) accessing the file (o option)

sc - security code; displayed only for FMGR files (y option)

```
CI.65> dl @ -l*m
directory DEMO
    name   ex  prot type msize blks words recs addr/lu

COPY.REL       rw/r     5    86   86  6312  127 8390/38
COPY.SRC     * rw/r     4    92  184 13418  820 7908/38
```

```
CI.65> dl &fdlx::db !
directory ::DB
    name    sc type msize rlen addr/lu

&FDLX        0    4   131    0 4830/27
```

This example demonstrates the limited directory information available for FMGR files.

DN (Down a Device or I/O Controller)

Purpose: Declares a device or I/O controller down (i.e., unavailable for use by the RTE system).

Syntax: DN,,lu or

DN,eqt

lu — Specifies the system LU of the device to be declared down.

eqt — Specifies the Equipment Table (EQT) entry number of the I/O controller to be declared down.

Description:

Downed device (or I/O controller) can be made available by the UP command. The EQT and LU number can be displayed with the LU command under CI. The DN command is identical to the SYSTEM DN command. Refer to the RTE-6/VM Terminal User's Reference Manual for a complete description.

Examples:

```
CI.65> dn,,6        (Declares LU 6 down)

CI.65> dn,28        (Declares EQT 28 down)
```

ECHO (Display Parameters at Terminal)

Purpose: Displays parameters, separated by commas, at the terminal.

Syntax: ECHO[parameters]

parameters One or more parameters separated by blanks or commas. Positional, user-defined, and predefined variables can be included in the string. If this parameter is omitted, a blank line is displayed.

Description:

The ECHO command displays the specified string after CI has shifted the input to uppercase, put commas between the parameters in the string, performed variable substitution, and removed CI quotes (grave accents and backslashes). You can use CI quotes to keep CI from altering any parameters in the input string.

Positional, user-defined, and predefined variables are referenced by including a dollar sign ($) before the variable name. If you want to examine the value of only one variable, you can use the ECHO command instead of the SET command.

Examples:

```
CI.65> echo ru edit test.ftn              (Display specified string.)
RU,EDIT,TEST.FTN                          (String is uppercase and
                                          commas separate parameters.)

CI.65> echo $session                      (Displays value of $SESSION.
65                                        (Session number is 65.)

CI.65> wd /mine/temp                      (Set working directory.)
CI.65> echo `Your working directory is `$wd  (Display message indicating
Your working directory is /MINE/TEMP      your current working directory.)
```

EQ (Displays I/O Controller Status)

Purpose: Displays a description and the status of an I/O controller, as recorded in the Equipment Table (EQT) entry.

Syntax: EQ eqt

eqt Specifies the EQT entry number of an I/O controller.

Description:

The EQT number can be displayed with the LU command under CI. The EQ command is identical to the SYSTEM EQ command. Refer to the RTE-6/VM Terminal User's Reference Manual for a complete description.

EQ (Buffering)

Purpose: Changes the automatic buffering designation for a particular I/O controller.

Syntax: EQ eqt un/bu

eqt Specifies the Equipment Table (EQT) entry number of the I/O controller.

un Turns off (unbuffer) buffering.

bu Turns on buffering.

Description:

The EQ command is identical to the SYSTEM EQ command. Refer to the RTE-6/VM Terminal User's Reference Manual for a complete description.

EX (Exit)

Purpose: Terminates the Command Interpreter program.

Syntax: EX

Description:

The EX command is identical to the SYSTEM EX command. Refer to the RTE-6/VM Terminal User's Reference Manual for a complete description.

GO (Resume Suspended Program)

Purpose: Resumes execution of a suspended program.

Syntax: GO[prog[parm*5]]

prog — Name of the suspended program.

parm*5 — Parameters to be passed to the program only if the program has suspended itself.

Description:

The GO command is identical to the SYSTEM GO command. Refer to the RTE-6/VM Terminal User's Reference Manual for a complete description.

HE (Help)

Purpose: Provides explanation of an error and guidance in possible corrective action.

Syntax: HE[keyword[lu]]

keyword — A select group of eight or less characters identifying the error for which an explanation is requested. All keywords and the corresponding explanations are contained in a disc resident HELP file. Default is the last error occurred in that session.

lu — LU of the device where the explanation is to be sent. Default is the session user's terminal.

Description:

The HE command is identical to the SYSTEM HE command. Refer to the RTE-6/VM Terminal User's Reference Manual for a complete description.

IF-THEN-ELSE-FI (Control Structure)

Purpose: Allows decision making in a command file.

Syntax:
```
IF command-list1
THEN command-list2
[ELSE command-list3]
FI
```

command-list — A list of commands either one command per line or multiple commands per line separated by semicolons. A command-list can be null.

Description:

The IF-THEN-ELSE-FI control structure allows you to control execution of a command file. The control structure can be entered interactively, but is more useful in a command file. The ELSE branch is optional.

The return status of the last command in the command-list for IF determines which branch of the IF structure is executed. If the return status is zero (the command was successful), then CI executes the THEN branch. If the return status is non-zero (the command was unsuccessful), then CI executes the ELSE branch, if one exists, or FI, which terminates the IF control structure.

CI determines the end of a command-list to be the CI command before the next expected control structure command. For example, the command-list for IF ends when CI reach THEN.

An IF-THEN-ELSE-FI control structure can be nested in either another IF-THEN-ELSE-FI or a WHILE-DO-DONE control structure.

FI is required to end the IF-THEN-ELSE-FI control structure. If you do not include FI, CI does not recognize the control structure as being finished and continues to process succeeding commands as though the commands were part of the THEN or ELSE command-list. Therefore, if an IF-THEN-ELSE-FI control structure has just been executed and CI is not executing commands that should be executed, check that you entered an FI command to terminate the control structure.

Examples:

The following interactive IF-THEN-ELSE-FI control structure copies file TEST, if it exists, to another directory or creates file TEST if it does not exist:

```
CI.65> if dl test; then co text /junk/@; else edit test; fi
```

The following command file compiles a FORTRAN source file. If successful, a library is created from the relocatable and the intermediate files created during the merging and indexing of the library are purged.

```
IF ftn7x general_stuff.ftn - -
THEN
  * Merge general_stuff
  pu general_stuff.merg
  merge general_stuff.cmd general_stuff.merg
  *
  * Index the merged file
  lindx general_stuff.merg general_stuff.lib
  *
  * Clean up
  pu general_stuff.merg
  pu general_stuff.lst
  pu general_stuff.rel
FI
```

IN (Initialize Disc Volume)

Purpose: Prepares a blank disc volume for use in the system.

Syntax: IN lu[block[OK]]

lu	The LU number of the disc volume to be initialized.
block	Specifies the number of blocks at the beginning of the disc to be reserved. These blocks will not be used by the file system and can be set aside for user software. Default is no reserved space.
OK	Optional parameter that suppresses the user prompt, indicating that the command should be executed as entered.

Description:

This command is used to clear a disc volume, eliminating all its files. Before reinitializing a disc volume with files on it, a prompt will be displayed to verify intent. A yes entry must be received to start the process. The OK parameter can be used to suppress this prompt. After initializing the disc volume, it will be mounted to the file system.

Only a System Manager may initialize a disc volume.

To initialize a disc volume for use with FMGR files, you must run FMGR and use the SYSTEM IN command. Refer to the FMGR description in the RTE-6/VM Terminal User's Reference for details.

IS (Compare Strings or Numbers)

Purpose: Compares two character strings or numbers.

Syntax: IS string1 <rel operator> string2[option]

string1	A numeric or character string.
rel operator	Relational operator indicating the relation being tested. The two sets of operators recognized are as follows:

=	or	EQ	Equal to
<>	or	NE	Not equal to
<	or	LT	Less than
<=	or	LE	Less than or equal to
>	or	GT	Greater than
>=	or	GE	Greater than or equal to

string2	A numeric or character string.
option	Specifies special comparison instructions. The possible values are as follows:

-i Integer comparison. A suffix of B following string1 or string2 in either upper or lowercase indicates an octal value. A leading - sign is accepted for decimal values.

-a Do not fold alphabetic characters before comparison.

Description:

IS compares two strings either with an ASCII comparison or as integers after converting both strings to integers.

The ASCII comparison is normally performed with alphabetic characters folded to uppercase. In an ASCII comparison, a shorter string is extended with blanks before the comparison is made.

IS is most useful when used in either the IF-THEN-ELSE-FI or WHILE-DO-DONE control structures.

IS returns the following status values in $RETURN1:

0	Relation is true
1	Relation is false
2	Relational operator missing or invalid
3	Option not recognized
4	Non-digit appears with -i option in effect

Examples:

```
CI.65> is 1024 eq 2000B -i  (The two strings are compared as integers.)
CI.65> echo $return1        (Display the result.)
0                          (The two numbers are equal.)

IF is $wd ne /system/test  (IS compares $WD with a specified working
THEN wd /system/test       directory; $WD is changed if the comparison
FI                         is TRUE.)
```

IT (Interval Timer)

Purpose: Sets execution time and interval of repetition when a program is scheduled with the ON command. Places a program into the time list.

Syntax: IT program res mpt[hr[min[sec[ms]]]]

To take a program out of the time list:

IT program

program — Name of the program to be placed in the time list.

res — Time interval resolution:

1	tens of milliseconds
2	seconds
3	minutes
4	hours

mpt — Multiplier used in conjunction with time interval resolution value. Can be in the range of 0 to 4095. If 0 is specified, the program runs only once.

hr min sec ms — Optional parameters setting the initial time in terms of hour, minute, second, and tens of milliseconds. Default for any parameter is zero (0).

Description:

The IT command is identical to the SYSTEM IT command. Refer to the RTE-6/VM Terminal User's Reference Manual for a complete description.

LI (List Files)

Purpose: Lists a file or a group of files to a device.

Syntax: LI mask[format[line1[line2]]]

mask — Specifies a file mask. Refer to the DL command in Chapter 5 for the file mask description.

format — Specifies text (ASCII) or binary (octal) output. The values for this parameter are as follows:

A ASCII
B octal

If omitted, type 3 and 4 files are listed in ASCII. All other files are listed in octal.

line1 — Specifies start line. Defaults to beginning of file.

line2 — Specifies stop line. If omitted, only line1 is listed. If both are omitted, the whole file is listed.

Description:

If both line1 and line2 are omitted, the entire file is listed, a screenful (21 lines) at a time. After each screenful, you are prompted for further action. The following options are available:

Action	Enter
List the screenful of lines (21 lines plus the last line of the current display)	Space bar
List the remainder of file	Carriage return
Abort listing	a

If you enter a file mask, you are prompted as follows before each file is listed:

Action	Enter
Do not list this file	n (no carriage return needed)
Abort the list command	a (no carriage return needed)
Display the next file	Any character other than "n" or "a"

The no list character "n" can be either uppercase or lowercase. It causes LI to skip to the next file.

The abort character "a" can be either uppercase or lowercase. After it is entered, the LI command stops and the CI prompt is displayed.

Example:

```
CI.65> li /mary/csort.ftn a 1 5
       Program CSORT

       Parameter (name_bound = 5)
       Character*12 name(name_bound),name_temp
       DO i=1,name_bound

      tm[ month day year hr:min[ sec[ pm] (change system time)
```

LU (Display/Modify Device Assignment)

Purpose: Displays information associated with a device specified by its LU number. Selected status can be modified by the System Manager using this command.

Syntax: LU lu[eqt[subchannel]]

lu — Specifies the system LU for which information or reassignment is desired.

eqt — Used by the System Manager only. Assigns the EQT entry number to the LU specified. If 0 is specified, LU becomes the bit bucket.

subchannel — Used by the System Manager only. Assigns subchannel number (0 to 63) to specified LU.

Description:

The LU command is identical to the SYSTEM LU command. Refer to the RTE-6/VM Terminal User's Reference Manual for a complete description.

MC (Mount Disc Volume)

Purpose: Mounts a disc volume and makes its contents available to the system.

Syntax: MC lu

lu The LU number of the disc volume to be mounted. Must be a positive number.

Description:

The MC command mounts a disc volume to the file system making it accessible to users of the file system.

If the disc volume has a valid FMP or FMGR directory, the volume is mounted. If the disc volume does not have a valid FMP or FMGR directory, you are prompted to confirm that the volume should be initialized. This is to avoid accidental corruption of volumes that are not of FMP or FMGR types (special backup utility volumes, for example).

The MC command does not place reserved blocks at the beginning of the volume. Use the IN command if reserved blocks are required.

There is no significance to the order in which disc volumes are mounted, unless there are duplicate global directory names on two or more volumes. If a global directory on the newly mounted disc volume has the same name as a previously mounted global directory, the new directory is inaccessible. You need to rename the previously mounted directory, then dismount the new disc volume and remount it. Now the new directory can be accessed.

This command should NOT be used to mount FMGR cartridges. Use the FMGR MC and DC commands to manage FMGR cartridges. Refer to the RTE-6/VM Terminal User's Reference Manual for details.

MO (Move Files)

Purpose: Move files from one directory to another, on a given disc volume. Also renames files. File mask can be used to move a group of related files.

Syntax: MO file1 file2

file1 The source file descriptor. (Refer to CR command syntax description for the definition of file descriptor.) May be masked to move a group of files.

(Refer to the File Mask description in the DL command section in this chapter for the mask syntax.)

file2 — The destination file descriptor. The file name may be defaulted to that of the source file name. May be masked to allow the system to generate destination names.

Description:

The MO command can be used to move a group of files from one directory to another. Masking the file1 parameter allows matches of a number of files. If a wildcard character is used in the filename field of file1, an appropriate destination mask must be used to default the destination file names.

Note that this command is very similar to the CO command. It uses the same syntax and performs nearly the same operation, but with the following important differences.

1. Files are MOVED, not COPIED. This means that after the MO command a file will no longer be where it used to be.

2. The file contents are not moved, only the directory entry is moved. This is much faster, particularly for large files. This is also more reliable since the data is not altered.

3. Files cannot be moved across disc volumes. This is because the data is not moved, and the data must be on the same volume as the directory entry. If you wish to move files across volumes, the CO command can be used with the 'p' option (purge source after copy) to move the files.

Examples:

```
CI.65> mo @.@.a-8306 /backup/archive/@.@
```

This example moves all files which have not been accessed since June 1983 into the archive subdirectory of the backup directory.

```
CI.65> mo /myglobal/mysubdirectory.dir /mynewglobal
```

This example causes the subdirectory to become a global directory. A file which formerly had the name '/myglobal/mysubdirectory/myfile' now has the name '/mynewglobal/myfile'. The file data has not changed, nor has the directory data in 'mynewglobal'.

OF (Stop/Remove Program)

Purpose: Stops a scheduled program or releases a program ID segment.

Syntax: OF[prog[parm]]

prog Program name, up to five characters, session identifier optional.

parm An optional parameter used to specify action to be taken. Possible values are:

0 - remove from time list (default)

1 - terminate immediately; release disc tracks

8 - terminate immediately and remove ID segment

ID - same as 8

Description:

The System Manager can use this command to remove any program if the need arises. General users can only remove non-system programs in their own session. This command is identical to the SYSTEM OF command. Refer to the RTE-6/VM Terminal User's Reference Manual for a complete description.

ON (Schedule Program)

Purpose: Schedules a program for execution. Up to five parameters and the command string may be passed to the program.

Syntax: ON[program[NO[parm*5]]]

program Specifies the name of a program to be scheduled.

NO(W) Schedules immediately a program that is normally scheduled by the system clock. If the program is placed in the time list, but not scheduled for immediate execution, this parameter and its preceding comma are omitted. It may be entered as NOW.

parm*5 Up to five parameters may be passed to the program when it is scheduled.

Description:

The ON command is identical to the SYSTEM ON command. Refer to the RTE-6/VM Terminal User's Reference Manual for a complete description.

OWNER (Display/Change Owner)

Purpose: Displays or changes the owner of a directory or a subdirectory.

Syntax: OWNER directory[newOwner]

directory The name of the directory or subdirectory. No wildcard characters allowed.

newOwner The name of the new owner. Needed only if a change is required. If omitted, the owner of the directory or subdirectory specified is displayed.

Description:

This command assigns or displays ownership of the named directory. Only the current owner of the directory can assign ownership. Ownership is associated with directories (including subdirectories), and cannot be specified on individual files. The directory cannot be on a remote system (if DS is used) or specified with an account.

When the owner is changed, the current user is no longer the owner of that directory, thus unable to change the owner back. This change can also make all subdirectories of this directory inaccessible to the original owner. Note that the ownership of subdirectories is not changed when the ownership of the directory they are in is changed.

Ownership is maintained through owner numbers, rather than owner names, so ownership remains correct even if the user's log-on name is changed with the ACCTS program. Note that if a removable disc is moved to another system with different user accounts, ownership will not be correct.

The newowner parameter must be a name which is usable for log on. That is, there must be a user with an account with that name on the system.

PATH (Display/Modify UDSP)

Purpose: Allows you to display or modify a User-Definable Directory Search Path (UDSP).

Syntax:

```
PATH[ -E]

PATH[ -E][ -N:n] udspnum[ dirname1[ dirname2[...[ dirnameN]...]]]

PATH[ -E] -F,file/lu
```

-E Turn off echo; non-error messages are not displayed.

-N:n Display or modify the specified entry. Set n equal to 1 for UDSP #0 (home directory); otherwise, set n to a value between 1 and the UDSP depth.

udspnum Specifies the UDSP number. The values for udspnum are as follows:

- 0 Home directory.
- n UDSP number between one and the number of UDSPs defined for this session (maximum is eight).
- -A All UDSPs defined for current session.

dirname Specifies the directory name. The following special characters can be used:

- . Use the working directory that is current when the UDSP is referenced.
- ! Delete this UDSP or entry; this character must be the only dirname in the command line.

-F,file/lu Indicates that the commands will be input from the specified file or LU.

Description:

The first format of the PATH command displays current UDSP information: the total number of UDSPs defined for the session, the depth (number or entries per UDSP), and the next available UDSP.

The second format displays or defines a specific UDSP or a specific entry of a UDSP.

The third format indicates that the specified file or LU contains commands to define or display the UDSPs. Specifying the -E parameter inhibits echoing of commands from the specified file. The file or LU can contain one or more command lines. The syntax for a command line is as follows:

```
[ -N:n ]udspnum[ dirname1[ dirname2[...[ dirnameN]...]]]
```

A unique set of UDSPs is associated with your session. The number of UDSPs and the depth (number of entries) for each UDSP are set when your user account is created or modified. You can have from zero through eight separate UDSPs; each UDSP has the same depth.

At logon, all UDSPs are undefined. You must issue a separate PATH command for each UDSP you want to define. The UDSPs created by the PATH command are valid only for the current session. By placing PATH commands in your HELLO file, you ensure that the UDSPs are defined the same each time you log on.

Although eight UDSPs are available; the first three have the following special meanings:

UDSP #0 Represents the home directory and has a predefined depth of one.

UDSP #1 Used by the RU command. Whenever you enter an RU command, implied or explicit, without specifying any directory information, the search path defined for UDSP #1 is used. If you do not define UDSP #1, the default search sequence is used. It is recommended that /PROGRAMS always be the last entry in the search path.

UDSP #2 Used by the TR command. Whenever you enter a TR command, implied or explicit, without specifying any directory information, the search path defined for UDSP #2 is used. If you do not define UDSP #2, the default search sequence is used. It is recommended that /CMDFILES always be the last entry in the search path.

UDSPs #3 through #8 can be used for your application programs. See the description of FmpOpen in the RTE-A Programmer's Reference Manual.

Only CI hierarchical directories can be entered as part of a UDSP; FMGR cartridges cannot be specified. However, if a period (.) is defined as a UDSP entry and the working directory is set to zero before the UDSP is referenced, all mounted FMGR cartridges are searched.

PATH returns the following values in the five $RETURN variables:

$RETURN1	If zero, the command was successful; otherwise, an FMP error code is returned.
$RETURN2	Number of UDSPs defined for this account
$RETURN3	Depth value
$RETURN4	Next available UDSP (first UDSP that is undefined)
$RETURN5	Zero (not used)

The name of the directory is returned in $RETURN_S when a specific entry (-N:n option) or the home directory (PATH 0) is requested.

Examples:

Display current UDSP information:

```
CI.65> path
```

Display UDSP #1:

```
CI.65> path 1
```

Display all UDSPs:

```
CI.65> path -a
```

Set home directory to /MINE:

```
CI.65> path 0 /mine
```

Set UDSP #2 to the following:
(1) current working directory
(2) /MINE/CMDFILES
(3) /CMDFILES

```
CI.65> path 2 . /mine/cmdfiles /cmdfiles
```

Read PATH commands from file SETPATH.CMD without echoing messages:

```
CI.65> path -e f- setpath.cmd
```

where SETPATH.CMD contains the following command lines to set UDSPs #0, #1, and #2:

```
0 /mine
1 . /mine/progs /programs
2 . /mine/cmds /cmdfiles
```

Display the third entry of UDSP #1:

```
CI.65> path -n:3 1
```

Set the first entry of UDSP #2:

```
CI.65> path -n:1 2 /groups/cmds
```

Delete all entries of UDSP #3:

```
CI.65> path 3 !
```

Delete the second entry of UDSP #4:

```
CI.65> path -n:2 4 !
```

Return the contents of the second entry of UDSP #3 in $RETURN_S without echoing the name to the terminal:

```
CI.65> path -e -n:2 3
```

Return the name of the home directory without echoing it and then set the working directory to the home directory:

```
CI.65> path -e 0
CI.65> wd $return s +s
```

PR (Change Program Priority)

Purpose: Change priority of a restored program. It can also be used to display the priority of a program.

Syntax: PR prog priority

- prog — Program name, up to five characters, session identifier optional.
- priority — Range is between 1 and 32767.

Description:

The PR command is identical to the SYSTEM PR command. Refer to the RTE-6/VM Terminal User's Reference Manual for a complete description.

PROT (Display/Change Protection)

Purpose: Displays or changes the protection status of a file or directory.

Syntax: PROT mask[owner/users]

mask	A file mask that includes all fields of the file descriptor and a qualifier. Refer to the File Masks section in Chapter 3 for a full description of the file mask.
owner/users	Specifies access allowed for owner and other users. Abbreviations used for read and write access, r for read and w for write access. The slash is a required delimiter. Any combination can be specified. If omitted, the current protection status is displayed.

Description:

If new protection is not specified, this command displays the current protection on files which match the mask. If new protection is specified then all files which match the mask will have their protection changed to the new protection.

To change protection on a file the user must be the owner of the directory on which the file resides. Following are typical combinations of file protection status:

rw/rw	Everyone has access to this file.
rw/r	Owner has full access; others read only
r/	No one can write, only the owner can read.
/rw	The owner cannot access this file, others can.
/	No one can read or write. Note that in this mode only superusers can access this file.

Examples:

```
CI.65> prot xyz.dat rw/rw

CI.65> prot /dir/secret.dir rw/
```

PU (Purge Files)

Purpose: Purge files.

Syntax: PU mask[OK]

mask — A file mask that may include all fields of the file descriptor and a qualifier.

Refer to the File Masks section in Chapter 3 for a full description of the file mask.

OK — Optional parameter indicating that the purge is as intended and the files specified will be purged without further user intervention.

If omitted and a file mask is used, a prompt is displayed for each file to be purged:

Purging <file descriptor> (Yes, No, Abort, stop asking) ? [Y]

Enter: y or <cr> to purge the file

n to skip this file

a to abort the purge operation

s to stop asking and purge the rest of the files matching the mask.

Description:

Wildcard purge uses the file mask feature to specify a group of files to be purged. It is a powerful capability but should be used with great care. If a wildcard purge is specified there are two possible checks which will be made. If the OK parameter is specified, it is assumed that the entry is correct and the purge will be performed on all files that match the mask specified. Their names will be displayed on the log device as they are purged. If the OK parameter is not specified, this command switches to an interactive mode. A prompt is displayed for each file to allow you to step through the files, purging the selected files. A carriage return will purge a file and another prompt is displayed for the next file. The command can be terminated by entering the letter a (abort). If the input device is not an interactive device and OK is not specified, wildcard purges will not be executed.

FMGR files with a security code must be purged individually with the security code specified.

If no mask is specified, no files will be purged. To purge a file, you must have write access to the directory containing the files. The file must not be an active type 6 file, the system swap file, an opened file, or a directory containing files.

The mask can be a single file which will be purged with the following message displayed:

```
Purging <file descriptor>
```

The PU command can be used to purge an empty directory. If any of the conditions previously mentioned exists, an error message will be displayed. Note that purging a global directory is done by "PU /GLOBAL". The command "PU ::GLOBAL" will purge all files on directory GLOBAL but not the directory. Note that in this case, the form /GLOBAL/ is not the same as ::GLOBAL and will not produce the desired results.

Examples:

CI.65> pu /dir/file	(purge FILE in directory DIR)
CI.65> pu @.tmp ok	(purge all files in working directory with file type extension .TMP)
CI.65> pu /dirname/	(purge all files in directory DIRNAME)
CI.65> pu /dirname	(purge directory DIRNAME)
CI.65> pu /dirname/sub.dir	(purge subdirectory SUBDIR. Note that the file type extension DIR is required here to avoid confusion with files named SUB.)

QU (Timeslice Quantum)

Purpose: Displays examination of the current system timeslice quantum and the program priority level at which timeslicing begins. The System Manager may change the timeslice parameters with this command.

Syntax: QU[quantum[limit]]

quantum — Specifies the new system slice quantum; value must be in the range between 0 and 32767 milliseconds. Default is 1500.

limit — Specifies the priority level at which timeslicing begins; default is 50. All programs of equal or lower priority (higher priority number) will be timesliced.

Description:

The QU command is identical to the SYSTEM QU command. Refer to the RTE-6/VM Terminal User's Reference Manual for a complete description.

RETURN (Return from Command File)

Purpose: Returns to the previous level of command file nesting or to the interactive mode.

Syntax: RETURN[,return1[,return2[,return3[,return4[,return5[,return_s]]]]]]

return1 — Integer return status indicating success or failure of command file. A value of zero indicates success; a nonzero value indicates failure. If omitted, return1 is set to zero.

return2 - 5 — Four integer values made available to return additional status information. Each omitted parameter is set to zero.

return_s — A string of up to 80 characters. If omitted, return_s is set to null.

Description:

The RETURN command allows exiting from a command file at any point. (Note that LU 1 (your terminal) is treated as a command file.) If RETURN is entered interactively when a parent program is waiting, CI returns to the parent program. If RETURN is entered interactively and a parent program is not waiting, the command is ignored.

All the parameters for the RETURN command are position dependent; therefore, you must include commas to mark the positions of any omitted parameters.

The values returned are available in the predefined variables $RETURN1 through $RETURN5, and $RETURN_S.

You can include a RETURN command anywhere in IF-THEN-ELSE-FI or WHILE-DO-DONE control structures.

CI always executes a RETURN command when the end of a command file is reached, whether or not you included the command at the end of the file.

Examples:

The following command exits a command file and specifies 5 integer values and a string:

```
return,0,2,3,4,5,`Command file successful`
```

The following command exits a command file using the default return values:

```
return
```

The following command exits a command file returning a value in only the variable $RETURN_S:

```
return,,,,,,`Successful completion`
```

RN (Rename File, Directory, or Subdirectory)

Purpose: Renames a file, a directory, or a subdirectory.
rn foo Syntax: RN file newName

file	The existing name; a mask can be used (refer to the File Mask description in Chapter 3)
newName	A destination mask can be used.

Description:

The RN command will change the name, file type extension, or any combination of the above of name1 to those for name2. The new name must not already exist on the directory. You must have write access to the directory.

Directories and subdirectories can be renamed. Renaming can also be done using masks for the source and destination names. This command does not move files into a different directory. If the directory field of the destination file name is blank, the source directory will be used. If source and destination directories are different, an error message will be displayed. In this case, use the MO command.

Examples:

(1) Rename file FOO on the working directory to JOE:

```
CI.65> rn foo joe
```

(2) Change the file type extension of FOO from TXT to FTN:

```
CI.65> rn foo.txt foo.ftn
```

(3) Change all files with file type extension SRC to type extension FTN:

```
CI.65> rn @.src @.ftn
```

RP (Restore Program File)

Purpose: Establishes a permanent program ID segment.

Syntax: RP file[prog]

file — File descriptor of the type 6 program file to be restored. The first five characters of the file name are used as the program name, unless the optional parameter is specified.

prog — New program name to be used instead of file name, up to five characters.

Description:

The RP command sets up an ID segment for the type 6 program file specified. This restores the program ID segment, making it available for use by program control commands and subroutines that require a restored program, e.g., the WS, VS, SZ command. If newname is not specified, it will be derived from the file name. Refer to the RU command description for details on searching for the correct file to restore.

The RP'd program remains associated with the session that RP'd it and will be removed when the user logs off.

RU (Run Program)

Purpose: Immediately schedules a program for execution and waits for its completion.

Syntax: [RU]prog/file[parm*5]

RU	An optional parameter that is only required if the program name is two characters that can be interpreted as a CI command or if the prog/file parameter may be confused with a command file (refer to section on TR command and predefined variables).
prog/file	A 5-character program name or a file descriptor that identifies a type 6 file. Including the optional ":IH" in the program name (for example, PROG1:IH) inhibits cloning of the program.
parm*5	Parameters to be passed to the program. The maximum run string length, including the implied RU and delimiters, is 256 characters. This may be five parameters or one long character string.

Description:

If the program is not restored, CI restores it, then frees the program ID segment after it finishes running. CI will modify the program name if necessary to make it unique when it restores the program. The last character will be changed to A, B, etc.

It is recommended that you always use the .RUN file type extension in the program file name.

If you will be executing more program files than command files, you should set the predefined variable $RU_FIRST to TRUE. When $RU_FIRST is set to TRUE, CI assumes that any file name entered without a CI command or file type extension is a program file and immediately attempts to execute the file as a program.

When you enter an implied or explicit RU command, the following procedure is used to find the program file:

1. If a directory is specified, this directory is searched for the file. If the file is found, it is restored. If the file is not found and a file type extension was not specified, .RUN is assumed and the directory is searched again. If the file is still not found, an error is returned.

2. If no directory information is given, the following occurs:

   a. If a program with the specified or assigned name is already restored and is clonable, this program is cloned. If the program cannot be cloned, and is dormant, then the original program is used.

   b. If the program has not been restored, a search is made for the program file. If User-Definable Directory Search Path (UDSP) number one is defined, a default file type extension of .RUN is assumed and the search path defined by UDSP #1 is used to find the file. If the file is not found, an error is returned. (Refer to chapter 3 for a description of UDSPs.)

   c. If UDSP #1 is not defined, the following default search sequence is used:

      - The current working directory is searched. If the file is not found, a default file type extension of .RUN is assumed and the working directory is searched again.

      - If you do not have a working directory, all mounted FMGR cartridges are searched.

      - If the file is still not found, global directory PROGRAMS is searched, using the .RUN default file type extension. If the file is not found, an error is returned.

For example, the search sequence for program EDIT specified in "RU,EDIT" if a working directory exists and UDSP #1 is undefined is as follows:

1. Search for a restored (RP'd) EDIT.

2. Search for EDIT in working directory.

3. Search for EDIT.RUN in working directory.

4. Search for EDIT.RUN in directory PROGRAMS.

If there is no working directory, the search sequence is:

1. Search for a restored (RP'd) EDIT.

2. Search for EDIT in FMGR disc cartridges.

3. Search for EDIT.RUN in directory PROGRAMS.

Parameters passed to the program can be either integer binary or ASCII. If an ASCII string is specified for a program that uses RMPAR, the string is parsed into two-character words that are each passed as separate parameters up to the maximum of five.

SET (Display/Define Variables)

Purpose: Displays all positional, user-defined, and predefined variables, or defines a user-defined or predefined variable.

Syntax: SET[variable = string]

variable — A string of up to 16 letters, digits, and underscores, not starting with a digit.

string — A string of up to 80 characters.

Description:

CI provides variables that you can define (positional and predefined variables) and, also, allows you to create variables. The SET command displays these variables, defines user-defined variables, and modifies the predefined variables. Positional variables cannot be defined with the SET command.

If a variable is not specified, all positional, user-defined, and predefined variables are displayed.

Examples:

```
CI.65> set filename = /mine/stuff/my progs/test.ftn

CI.65> set auto logoff = 3

CI.65> set greeting = `How are you today?`

CI.65> set                                  (Displays all variables.)
```

SL (Display Session LU Information)

Purpose: Displays the corresponding system LU, Equipment Table (EQT) entry number and subchannel number for either a specified session LU or all session LUs.

Syntax: SL[lu]

lu — Specifies the session LU for which information is desired; default is to list information for all session LUs accessible to the user.

Description:

The session LU information is always displayed on the user terminal. The SL command is identical to the SYSTEM SL command. Refer to the RTE-6/VM Terminal User's Reference Manual for a complete description.

SS (Suspend Program)

Purpose: Suspends an active program.

Syntax: SS[prog]

prog	Name of an active program, session identifier optional.

Description:

The SS command places the program in operator suspend state. This is done immediately if the program is scheduled or executing. If the program is currently suspended for any reason other than an operator suspend, or if the program is dormant, the SS command is illegal.

The SS command is similar to the EXEC 7 program suspend call. Execution of programs suspended with the SS command may be resumed with the GO command or aborted with the OF command.

If prog is not specified and the session startup program (CI or FMGR) has scheduled another program, this command is executed on the scheduled program unless it, in turn, has scheduled a program. The search continues down the program scheduling chain and the SS command is executed on the last program. The only exception is if the last program is a protected system program, the program that scheduled it will be suspended.

The System Manager can suspend any program in the system. The general user can suspend only programs scheduled within that session.

Example:

```
CI.65> ss timer        (Suspends program TIMER)
```

ST (Display Program Status)

Purpose: Displays the status of a program. Information requested can be program priority, current list, time values, or the partition number of the program currently executing. A special case is to display the name of the program occupying a specified partition.

Syntax: ST[program/partition #/0]

program — Specifies the name of the program whose status is to be displayed.

partition # — Specifies the number of a partition (1 to 64) to display the program occupying that partition. If the partition is empty, 0 is displayed. If an undefined partition number is entered, an error message "NO SUCH PROG" is displayed.

0 — Entering zero (0) displays the name of currently executing program and its partition number. If there is no program executing, 0 is displayed.

Examples:

```
CI.65> ST 0
```

SZ (Display or Modify Program Size)

Purpose: Displays program size information of a restored program or modifies the program size requirements.

Syntax: SZ prog[size[msegSize]]

prog — Program name, up to five characters, session identifier optional.

size — Program size in pages for non-VMA programs or the EMA size for EMA programs, not including PTE. Range is $2 \leq size \leq 1022$ for EMA size.

msegSize — New MSEG size for EMA programs. Range is $1 \leq$ MSEG size ≤ 30.

Description:

This command changes the amount of memory which the specified program can use when it runs. The program must be restored with the RP command and must be dormant.

Increasing program size will help programs that use memory at the end of their partition for buffer or table space. Such programs include EDIT, LINK, Macro, and CI. To change the size permanently, use the LINK program.

This command is identical to the SYSTEM SZ command. Refer to the RTE-6/VM Terminal User's Reference Manual for a complete description.

TI (Display Time)

Purpose: Displays the system real-time clock.

Syntax: TI

Description:

The current system time is displayed in the following format: year, day (Julian), hour (24-hour format), minutes, and seconds.

Example:

```
CI.65> ti
1983 285 18 45 48
```

TM (Display or Set System Clock)

Purpose: Displays or sets the system clock.

Syntax: TM[month day year hr:min[:sec[pm]] (change system time)

month	Jan to Dec
day	1 to 31
year	1976 to 2144
hr	0 to 23
min	0 to 59
sec	0 to 59
pm	defaults to am

Description:

This command displays the system clock in the format shown in the following example:

```
Mon Sep 27, 1982  11:37:13 am
```

Only the System Manager can set the time. Parameters can be entered as they would be printed; time can be specified in 24-hour format if desired, with or without seconds.

Resetting the time affects programs in the time list but not programs set to run after a particular length of time. It also affects the time stamping of files, so use this command with caution.

TO (Display or Modify Device Time-Out)

Purpose: Displays or sets time-out limit for a device.

Syntax: TO eqt[interval]

eqt	EQT number of device.
interval	Number of 10ms intervals to be used as the time-out value for device EQT. Value can be in the range of $0 \leq \text{interval} \leq 32767$. If interval=0, time-out limit does not apply to this device.

Description:

The time-out value is displayed in the form:

```
    TO # eqt = interval
 or:
    INPUT ERROR (indicates given EQT number does not exist or value entered
                 was illegal)
```

The time base generator (TBG) generates an interrupt every 10 milliseconds. When a program sends an unbuffered I/O request to a device, the system puts the program into I/O suspension and begins counting the number of TBG interrupts. When the request is fulfilled, the program resumes execution. If, however, the number of TBG interrupts exceeds the time-out value defined, the program is put into a downed device wait state and the device may be set down. This prevents an off-line or down device from causing a program to remain I/O suspended indefinitely. When the program goes into this wait state it can be swapped out to the disc and another program can begin execution. When the device is once again available to the system, the original program can resume execution after the device has been UP'd.

If you set a time-out value too low for a device, that device may appear to be failing, when in fact it is performing properly. If the driver times the device out before it can respond to a request, the device will appear to be down.

To calculate the interval parameter, multiply the desired time-out value (in seconds) by 100.

When the system is rebooted, time-out values revert to those set at system generation time.

Examples:

To display time-out value:

```
CI.65> to 6
TO # 6 = 500            (5 second time-out)
```

To modify EQT 6 time-out to 10 seconds:

```
CI.65> to 6 1000
TO # 6 = 1000           (new value displayed)
```

TR (Transfer to Command File)

Purpose: Transfers control to a command file.

Syntax: [TR]file[parm*9]

TR	An optional parameter that is only required if the command file name is two characters that can be interpreted as a CI command, or if the file parameter can be confused with a program file (see sections on the RU command and predefined variables).
file	Specifies the file containing the commands. Refer to the CR command for a definition of a file descriptor.
parm*9	One to nine parameters that are used to replace occurrences of the postional variables $1 through $9 in the command file. Defaults to zero-length strings.

Description:

A command file (also known as a transfer file) contains a sequence of CI commands. The commands are executed as if you had entered them from the terminal. Command files are useful for executing command sequences repeatedly.

Command files can be nested by using the TR command in the command file. Control is returned to either the terminal or the command file depending on whether the TR command was issued from the terminal or another command file.

Positional variables $1 through $9 can be used in command files. A parameter in the runstring is substituted wherever the corresponding positional variable appears in the command file. Positional variables can be concatenated with characters in the command file.

It is recommended that you always use the .CMD file type extension in the command file name.

If you will be executing more command files than program files, you should set the predefined variable $RU_FIRST to FALSE. When $RU_FIRST is set to FALSE, CI assumes that any file name entered without a CI command or file type extension is a command file and immediately attempts to execute the file as a command file.

When you enter an implied or explicit TR command, the following procedure is used to find the command file:

1. If a directory is specified, this directory is searched for the file. If the file is found, it is executed. If the file is not found and a file type extension was not specified, .CMD is assumed and the directory is searched again. If the file still is not found, an error is returned.

2. If no directory information is given, the following occurs:

   a. The TR command checks User-Definable Directory Search Path (UDSP) number 2. If defined, the search path specified by UDSP #2 is used to find the file. If a file type extension is not specified, .CMD is assumed. If the file is not found, an error is returned.

   b. If UDSP #2 is not defined, the following default search sequence is used:

      - The current working directory is searched. If the file is not found, a default file type extension of .CMD is assumed and the working directory is searched again.

      - If you do not have a working directory, all mounted FMGR cartridges are searched.

      - If the file is still not found, global directory CMDFILES is searched, using the .CMD default file type extension. If the file is not found, an error is returned.

For example, if MYCMD is the name of the command file specified in the TR command, a working directory exists, and UDSP #2 is undefined, the default search sequence is as follows:

1. Search for MYCMD in working directory.
2. Search for MYCMD.CMD in working directory.
3. Search for MYCMD.CMD in directory /CMDFILES.

If there is no working directory, the search sequence is as follows:

1. Search for MYCMD in FMGR cartridges.
2. Search for MYCMD.CMD in directory /CMDFILES.

Examples:

In the following example, COMP.CMD, a command file that compiles, links, restores, and sizes program TEST4 and then removes its ID segment, is executed:

```
CI.65> tr comp.cmd
```

where COMP.CMD contains the following commands:

```
ftn7x test4.ftn test4.lst -
link test4.rel
rp test4
sz test4 28
test4
of test4 id
```

The following example shows executing a command file that uses positional variables:

```
CI.65> comp2 test4 28
```

where file COMP2.CMD contains the following commands:

```
ftn7x $1.ftn $1.lst -
link $1.rel
rp $1
sz $1 $2
$1
of $1 id
```

By specifying a different file name and program size in the TR command, this command file can be used with any FORTRAN program and program size.

UL (Unlock Shareable EMA Partition)

Purpose: Unlocks a shareable EMA partition so that it can be used by other programs.

Syntax: UL label

label A name that identifies a shareable EMA partition label, up to five characters.

Description:

This command is used to unlock a shared EMA partition. The partition is unavailable to other programs when the program that used the partition aborted. The UL command allows the user to release the partition. This command is identical to the SYSTEM UL command. Refer to the RTE-6/VM Terminal User's Reference Manual for a complete description.

Example:

```
CI.65> ul carea
```

UNPU (Unpurge Files)

Purpose: Recover purged files.

Syntax: UNPU mask

mask A file mask that specifies what files to unpurge. Refer to the File Masks section in Chapter 3 for a full description of file mask.

Description:

UNPU will restore a purged file to active status. This can be done only if the directory entry of the specified file has not been reclaimed by the file system and the disc space of the file has also not been reclaimed. There are no guarantees as to how long these conditions may last. If there are multiple purged files with the same name, the one recovered will be indeterminate. In this case the file can be unpurged and renamed, then the next copy of the file with the same name can be unpurged until all copies have been unpurged.

FMGR files and directories cannot be unpurged. Files recovered with the UNPU command retain the same attributes in effect when they were purged, including the time stamps.

UNSET (Delete User-Defined Variable)

Purpose: Deletes a user-defined variable.

Syntax: UNSET variable

variable String of up to 16 letters, digits, and underscores, not starting with a digit. The variable must exist.

Description:

The UNSET command deletes a variable that you defined earlier in the session. Deleting unneeded user-defined variables frees space that CI can use for defining other variables. You cannot use the UNSET command to delete positional and predefined variables.

Examples:

The following command removes user-defined variable $TEST_NAME:

```
CI.65> unset test name
```

The following command attempts to delete predefined variable $SESSION, which causes an error to occur:

```
CI.65> unset session
Cannot unset SESSION
```

UP (Up a Device)

Purpose: Notifies the system that a specified device is available.

Syntax: UP eqt

eqt The EQT number of the device.

Description:

The system downs a device when an error such as a time-out occurs. The EQT remains unavailable until the UP command is given with that EQT number. When a device is UP'd, any pending requests are retried. It is not an error to up a device that is not down.

This command is identical to the SYSTEM UP command. Refer to the RTE-6/VM Terminal User's Reference Manual for more information.

UR (Release Reserved Partition)

Purpose: Releases a partition previously reserved during system generation or reconfiguration.

Syntax: UR partition

partition Specifies the number of the partition to be released. The number can be in the range of 1 to 64 depending upon the system configuration.

Description:

This command is identical to the SYSTEM UR command. Refer to the RTE-6/VM Terminal User's Reference Manual for a complete description.

VS (Display or Modify VMA Size)

Purpose: Displays the VMA size or changes the VMA size requirements of a restored program.

Syntax: VS prog[lastpg]

prog Program name, up to five characters.

lastpg Specifies the last page of VMA; range is $31 \leq vsSize \leq 65535$. Note that the actual VMA size will be one page greater than the value entered. For example, if the last page specified is 53, then 54 pages of VMA will be allocated. The default value of lastpg is 8191 pages.

Description:

The virtual space is the disc area used for paging data that would not fit in memory. Increasing this space may allow the program to process more data. Decreasing it will cut down on disc space required to run the program. The program must be dormant when this command is given. It must have been linked as a VMA program.

This command is identical to the SYSTEM VS command. Refer to the RTE-6/VM Terminal User's Reference Manual for a complete description.

Examples:

```
CI.65> vs test4 199        (set VMA size to 200 pages)
```

WD (Display or Change Working Directory)

Purpose: Displays or changes the working directory.

Syntax: WD[directory name[file/+s]]

directory name	The name of the new working directory. May be a subdirectory.
file	The command stack file associated with the new working directory (.STK extension recommended). All subsequent posting of command stack contents will be to this file until another file is designated with another WD command.
+s	Causes posting of the contents of the command stack to either the file associated with the working directory or cleared if the file does not exist.

Description:

This command sets up the working directory which will be used when no directory is specified in a file name. It is searched first by the file system in the file search path. The new working directory can be a subdirectory.

The WD command changes the working directory associated with a session. Ownership, read, or write access to the working directory is not required, However, setting the working directory to a read or write protected directory may cause programs such as EDIT problems when these programs try to create scratch files.

The working directory cannot be defined as a FMGR cartridge, but it can be set to zero. This causes all FMGR cartridges to be searched when a directory is not specified in a file referencing CI command.

The second parameter provides the option of manipulating the command stack files. It allows posting of the contents of the command stack to a file on a particular directory so that the same commands can be used later. If a new command stack file is requested via a WD command, the command stack in memory is posted to the current command stack file if one exists, and the new file is opened and the command stack rewritten with the contents of the new file. If the new file does not exist, the stack is cleared. It will be created at log off or when another file is requested. Refer to the examples shown below for details.

Examples:

The following commands are entered in sequence with the following assumptions: working directory is DEBBIE with the associated command stack file CI.STK.

CI.65> wd,,+s	(Command stack contents posted to /DEBBIE/CI.STK)
CI.65> wd /tsmas ts.stk	(New working directory is TSMAS. Command stack contents posted to /DEBBIE/CI.STK. Contents of /TSMAS/TS.STK are written into command stack. If TS.STK does not exist, the command stack is cleared and TS.STK will be created at log-off or when the next WD command with the command stack option is executed.)
CI.65> wd /debbie +s	(Working directory is changed to DEBBIE. Command stack contents are posted to /TSMAS/TS.STK; if it does not exist, it is created. Contents of CI.STK are written into the command stack.)

WH (System Status Reporting)

Purpose: Runs the system status program WH for a report of system information.

Syntax: WH[parm]

parm Specify the information to be displayed:

AL	all programs
PA	memory partitions
SM	all system programs
PR/PL	all ID segments

The information produced from these run strings is described in the WHZAT utility program description given in the RTE-6/VM Utilities Manual.

WHILE-DO-DONE (Control Structure)

Purpose: Allows repeated execution of a group of commands. WHILE-DO-DONE can be used only in a command file.

Syntax:
```
WHILE command-list1
DO command-list2
DONE
```

command-list A list of commands either one command per line or multiple commands per line separated by commas. A command-list can be null.

Description:

The WHILE-DO-DONE control structure allows you to control execution of a command file.

The return status of the last command in the command-list for WHILE determines if the command-list for DO is executed. If the return status is zero (the command was successful), then CI executes the DO branch. If the return status is non-zero (the command was unsuccessful), then CI executes the DONE, which terminates the WHILE control structure.

CI determines the end of a command-list to be the CI command before the next expected control structure command. For example, the command-list for WHILE ends when CI reach DO.

A WHILE-DO-DONE control structure can be nested in either another WHILE-DO-DONE or an IF-THEN-ELSE-FI control structure.

DONE is required to end the WHILE-DO-DONE control structure. If you do not include DONE, CI does not recognize the control structure as being finished and continues to process succeeding commands as though the commands were part of the DO command-list. Therefore, if a WHILE-DO-DONE control structure has just been executed and CI is not executing commands that should be executed, check that you entered a DONE command to terminate the control structure.

Examples:

The following command file compiles a program and, if the compilation is errorless, links the program. The WHILE loop is repeated eight times, once for each file TEST_FILE1.FTN through TEST_FILE8.FTN. Program CALC performs the specified math operation on the two integers and returns the result in variable $RETURN_S.

```
set count = 0
WHILE IS $count lt 8 -i
DO
   *
   * Increment counter and
   * retrieve result from $RETURN_S
   *
   calc $count + 1
   set count = $return_s
   *
   * Compile and link file if no errors
   *
   if ftn7x test_file$count.ftn 0 -
      then link test_file$count.rel
      else echo $return1 `errors in test_file `$count
   fi
DONE
```

WHOSD (Report User of Directory or Volume)

Purpose: Reports the session that is using a specified directory or directory on a specified volume as a working directory or as part of a UDSP.

Syntax: WHOSD directory/lu

directory	Specifies the directory to be checked.
lu	Specifies disc LU of CI volume to be checked.

Description:

You cannot purge a directory being used as a working directory or as part of a User-Definable Directory Search Path (USDP). Also, you cannot dismount the volume on which it resides. You can use the WHOSD command to return information on which session is using the directory or LU.

An error is returned if you attempt to purge the directory or dismount the volume containing the directory.

Examples:

CI.65> whosd /programs (Scan for all sessions using /PROGRAMS as a working directory or as part of a UDSP.)

CI.65> whosd 65 (Scan for all sessions using a directory on LU 65 as a working directory or as part of a UDSP.)

WS (Display or Modify VMA Working Set Size)

Purpose: Displays the VMA working set size or modifies the working set size requirements of a restored program.

Syntax: WS prog[wrksz]

Prog Program name, up to five characters, session identifier optional.

wrksz Working set size in pages (not including PTE). Range is 2 ≤ wsSize ≤ 1022. Default is 31 pages.

Description:

The working set is a number of pages in a VMA user's partition which is used to hold a portion of the virtual memory space, including the page currently being accessed. Increasing the working set will generally improve performance, at a cost of more memory for running the program. The program must be dormant when this command is used. This command only works for VMA programs. For EMA programs, use the SZ command.

This command is identical to the SYSTEM WS command. Refer to the RTE-6/VM Terminal User's Reference Manual for more information.

XQ (Run Program Without Wait)

Purpose: Immediately schedules a program for execution.

Syntax: XQ prog/file[parm*5]

prog/file	Program name, up to five characters, or a file descriptor that identifies a type 6 program file to be executed.
parm*5	Parameters to be passed to the program. The total runstring has a limit of 80 characters.

Description:

The XQ command performs a "schedule without wait" operation. All other actions, i.e., comments and error handling, are identical to that of the RU command.

?/HE (Help)

Purpose: Display a summary of CI commands or a brief description of a command or item on the summary display.

Syntax: ?/HE[command]

Description:

This command provides a quick reference of CI commands and utility programs. The form "?" without any parameters gives a directory list of directory HELP to display a summary of files available. Entering "? <command>" list a file called /HELP/<command>, e.g., "? owner" lists file /HELP/OWNER". If there is no file by the name specified, a message is displayed. You can add files to the HELP directory to provide a quick reference of selected topics.

/ (Display Command Stack)

Purpose: Displays the command stack to allow selection of a previously entered command for execution.

Syntax: /[n]

n is a command line count that specifies the number of command lines from the last command entered. Then up to 20 of the most current commands are displayed beginning with the command line specified. The cursor is positioned at the top of the display.

Description:

The command stack holds between 25 and 200 lines; the actual number depends on the length of each line. Most command stacks hold approximately 100 lines. Duplicate commands and commands entered from a command file are not saved. When this command is executed, up to 20 lines of the command stack are displayed. The slash command is not saved in the command stack.

Command stacks can be saved in files. At logon, a file called CI.STK is searched for on the working directory or in the FMGR disc cartridges if there is no working directory. If the command stack file is found, it is opened and its contents used to initialize the command stack. If it does not exist, the command stack remains cleared. In this case, the default file CI.STK is created and the contents of the command stack posted there at logoff.

The file associated with the command stack can be changed to any file. The name of this file is designated with the WD command and it can be in any directory accessible to the session user. Refer to the WD command description for details on changing and posting command stack files.

At logoff, if there is an open command stack file, the contents of the stack are posted to the file and the file is closed. If the file does not exist, it is created on the working directory or the top cartridge on the FMGR cartridge list. The contents of the stack is posted to this file which is then closed. If a command stack file had not been specified, file CI.STK is used. If you do not want to either save your command stack in a file or have the current file updated, set the predefined variable $SAVE_STACK to FALSE.

Examples:

Assume that the command stack contains the following:

```
tr,transferfile.cmd
wh us
who
dl ::users
? dl
dl ::system
dl
hello
?
? ex
dl
ru dl
ss spot.run
go spot.run
br spot.run
tm
wh
io 6
up 6
luprn
wh al
? wh
? pu
pu spot.lst
pu spot.dbg
wh pa
edit testprog.ftn
co testprog.ftn backup.ftn
li testprog.ftn
```

To display the last 20 commands:

```
CI> /
---Commands---
? ex
dl
ru dl
ss spot.run
go spot.run
br spot.run
tm
wh
sl 6
up 6
luprn
wh al
? wh
? pu
pu spot.lst
pu spot.dbg
wh pa
edit testprog.ftn
co testprog.ftn backup.ftn
li testprog.ftn
-
```

Note that the cursor is at the bottom of the stack at a blank line. Pressing the return key will return to CI without any further action. Or the cursor can be moved up to select any command line. The terminal editing keys can be used to make changes and the command can be entered by pressing the return key.

To display 20 lines beginning at command line 25 from the last command:

```
CI> /25
---Commands---
? dl
dl ::system
dl
hello
?
? ex
dl
ru dl
ss spot.run
go spot.run
br spot.run
tm
wh
sl 6
up 6
luprn
wh al
? wh
? pu
pu spot.lst
```

Note that the cursor is positioned at the top of the stack. If n is less than 20, then the number of lines specified will be displayed. For example:

```
CI> /9
---Commands---
wh al
? wh
? pu
pu spot.lst
pu spot.dbg
wh pa
edit testprog.ftn
co testprog.ftn backup.ftn
li testprog.ftn
```

/n/ (Change Command Stack Display Size)

Purpose: Changes the size of the command stack display.

Syntax: /n/

n Specifies the number of command lines to be displayed when the / command is entered.

Description:

When you logon, the command stack display size is initialized to 20 lines. You can use this command to change the default command stack display size.

If you enter /0/, the zero is ignored and the command stack display size is set to 20.

Example:

Assume that the command stack contains the following command lines:

```
tr,transferfile.cmd
wh us
who
dl ::users
? dl
dl ::system
dl
hello
?
? ex
dl
ru dl
ss spot.run
go spot.run
br spot.run
tm
wh
sl 6
up 6
luprn
```

To change the display size to 7 lines and then display the last 7 lines, enter the following:

```
CI> /7/
CI> /
---Commands---
go spot.run
br spot.run
tm
wh
sl 6
up 6
luprn
_
```

Chapter 6
FMP Calls

The File Management Package (FMP) for the CI file system is a set of subroutines that manage disc files. FMP calls from a program can open, close, position, read from and write to files, and perform a number of sophisticated file manipulation tasks.

FMP can be called from FORTRAN, Pascal, Macro or other languages that support subroutine calls. Details for passing parameters in Pascal and Macro are given in the Character Strings section of this manual.

The FMP calls used in the CI file system are analogous to the FMGR FMP calls described in the RTE-6/VM Programmer's Reference Manual. Appendix C of this manual is a guide to converting FMGR FMP calls to FMP calls for use in the CI file system environment.

All FMP calls mentioned in this chapter and in Appendix C refer to those used in the CI file system environment. These are referred to as CI FMP calls. The FMP calls described in the RTE-6/VM Programmer's Reference Manual are referred to as FMGR FMP calls. Note that most of the FMP calls described in this chapter can be used to access files in the FMGR file system, observing the restrictions imposed by the differences in naming conventions and file system properties as described in Chapter 1.

The most common usage of FMP calls is to create or purge files and to read or write data at various locations in the files. The FMP calls provided for these basic tasks are described under the Basic FMP Subroutines section of this chapter. These calls can be used in the FMGR file system as long as the FMGR file system conventions are followed. There are other special purpose FMP calls grouped under the following categories: CI file system calls, utility calls, and Distributed System (DS) calls.

The first category of special purpose calls consists of subroutines used primarily in the CI file system environment. These calls are used to create or manipulate the hierarchical directories as well as handling time stamps, file mask, and other CI file properties. Some of these calls can be used in the FMGR file system environment. These calls are described in the CI File System FMP Calls section of this chapter.

The utility calls are subroutines that perform a variety of functions, e.g., copy data, error handling, device control, and mount or dismount disc volumes or disc cartridges (FMGR). Most of these calls can be used in both FMGR and CI file systems. These calls are described in the Utility FMP Calls section of this chapter.

The FMP calls used in the optional DS environment are described in the DS Communication Subroutines section of this chapter.

FMP Calling Sequence and Parameters

All parameters are required in every FMP call; there are no optional parameters. This simplifies programming, and minimizes FMP call coding differences between languages. Most of the FMP routines can be called as integer functions as well as subroutines. When called as functions, they return values to program variables. When called as subroutines, the function value is returned in the A-Register. In FORTRAN, FMP subroutines called as integer functions must be declared as integers. The FMP routine names are shown in upper and lower case letters throughout this manual to make it easier to identify their functions, but they can be specified in either case in user programs.

The FMP parameters common to most calls, such as the Data Control Block (DCB), file descriptor, character string handling, and error code are described in the following paragraphs.

Data Control Block (DCB)

A Data Control Block (DCB) is an integer array, defined by the calling program, that FMP uses to keep information about a file open to the program. A program may have several files open at once, and there must be a DCB for every open file, so the program should define several arrays to contain the DCBs. The FmpOpen subroutine sets up the DCB contents. Once a file is open, FMP refers to the DCB for file information.

The DCB array must be defined as a minimum of 144 words in length. Its contents are maintained entirely by FMP and must not be modified by the user program.

The first 16 words of the DCB contain file control information used by the FMP routines. The remaining words are used as a buffer to minimize the number of data transfers to disc. The smallest buffer permitted is one 128-word block. Larger DCB buffers must be a multiple of 128 words (128, 256, 384, and so on), up to a maximum of 127 blocks. The buffer size is independent of the file; a file created with a DCB buffer of 127 blocks can later be accessed with a DCB buffer of 128 words. The buffer only serves to reduce the number of disc accesses. File types 0 and 1 do not require buffers, so a DCB of only 16 words can be used.

File Descriptors

Files are specified by file descriptors which contain a file name, a file type extension, a directory and optional parameters such as subdirectory, file type, size, and record length. There are three file descriptor formats: standard, mixed, and FMGR compatible as shown below.

1. /dir/sub/filename.typex.qual:::type:size:rlen[user]>node

2. sub/filename.typex.qual::dir:type:size:rlen[user]>node

3. filename:sc:crn:type:size:rlen[user]>node

Where:

dir - A global directory name of up to 16 characters. The name must conform to the file name convention. In the standard and mixed formats, the delimiters (slashes) are required. If the leading slash is omitted, the first entry is assumed to be a subdirectory. In the FMGR compatible format, a subdirectory may be specified in the DIR parameter field. This parameter is optional when creating a file descriptor; defaults to the working directory.

sub -One or more subdirectory names of up to 16 characters each. The rules for directories apply to subdirectories. In the standard or mixed format, each subdirectory name is followed by a slash (/). In the standard format two or more subdirectories may follow the directory entry. As many subdirectories as necessary may appear, with the limitation that the entire file descriptor cannot be longer than 63 characters. This parameter is optional when creating a file descriptor.

filename - A CI file name of up to 16 characters. It must conform to the naming conventions described in Chapter 3. It can also be an FMGR file name, conforming to the FMGR file system rules.

typex - File type extension. It is a field appended to filename with the period as a delimiter; can contain one to four characters; it is used to describe the type of information in the file. Refer to Chapter 3 for the RTE standard file type extensions. This parameter is optional when creating a file descriptor; defaults to a null file type extension.

qual - (Optional) Mask qualifier, separated from the type extension by a period. Mask qualifiers are described in Chapter 3 of this manual.

type - (Optional) The RTE file types are:

0 I/O device (non-disc file); variable length records.

1 Random access file; fixed length 128-word records.

2 Random access file; fixed length user-defined records.

3 Sequential access file; variable length records; can be ASCII or binary.

4 ASCII text file; similar to type 3 file.

5 Relocatable binary file; similar to type 3 file.

6 Memory-image program file; similar to type 3 file, but accessed as a type 1 file.

7 Absolute binary program file; similar to type 3 file.

8 and above: User-defined file types, accessed as type 3 files. Any special processing based on file type must be supplied by the application program.

size - (Optional) The number of 128-word blocks in the file.

rlen - (Optional) For type 2 (fixed length) files, specifies the length of the records in the file.

user - Used only with the optional DS network. The user account name under which the file exists; delimited by square brackets. The full form is:

<user name>.<group>/<pass word>

node - Used only with the optional DS/1000 product. Number of the DS node where the file resides; preceded by a right angle bracket (>).

sc - (Optional) Security code used for FMGR files only. It can be a positive integer, a negative integer, or two ASCII characters. A positive integer other than zero (or two ASCII characters) provides write protection. A negative integer provides read and write protection.

crn - (Optional) A positive cartridge reference number, the negative logical unit number, or two ASCII characters; specifies the disc cartridge where the file is located. Used only in the FMGR compatible format.

When creating a file descriptor, all parameters except filename are optional. However, in accessing existing CI files, the correct directory/subdirectory path and the file type extension must be specified. Otherwise, the file may not be found.

Optional parameters can be omitted if not required, but place holders must be used if the omitted parameter is between specified parameters. For example,

1. ProgramDesc.txt[user.tp]>111

2. /pubs/ProgramDesc.txt::::24

3. /pubs/manual/Chapter@..x

The first example is in the FMGR compatible format; it specifies the file name with a type extension, defaults the security code (none), the directory/crn, and type, omits the size and record length, and specifies the DS parameters. The second example uses the standard format; it uses a place holder for the default file type and specifies a size of 24 blocks. The third example is also in the standard format and it uses a mask qualifier to specify a search directive; the second period is a place holder for the default (null) type extension.

Character Strings

The FMP calls pass filenames as character strings. This eliminates the need to count characters or treat characters as integers. The character strings are stored in the FORTRAN 77 character string format, which is described in the FORTRAN 77 Reference Manual.

The FMP routines are coded in FORTRAN 77, so the character strings are treated as fixed-length strings, and are padded or truncated from right to left to fit target strings. Zero-length strings are not permitted, so null strings are filled with blanks.

Note that nulls in a string (integer value of zero) are not treated as blanks but are treated as non-blank ASCII characters. Character strings are not always initialized to blanks but are initialized to nulls instead. Therefore, it is important for you to insure that strings are initialized to blanks. You can use a data statement or blank fill the buffer before the EXEC(14) call. For example, in the call SplitString(CBUFF,CPARM,CFUFF), blank fill the buffer with the following:

```
CBUFF=' '
```

Compilers (such as Pascal) or assemblers that do not use the FORTRAN 77 character string format must create a file descriptor in a format that the program can manage and that FMP can use.

File Descriptors in Pascal

Pascal supports a variable length character string format that can communicate with FMP routines when used with the Pascal FIXED_STRING compiler option. The Pascal character string format is not directly compatiable with the FORTRAN 77 character string format. The Pascal PACKED ARRAY OF CHAR is not compatible with the FORTRAN 77 character string format.

The FIXED_STRING compiler option indicates that string parameters of procedures or functions declared EXTERNAL should be converted from the Pascal variable length character string format to the FORTRAN 77 character string format before being passed.

The current length of the Pascal variable length character string is used as the maximum length of the FORTRAN 77 character string that is passed to the EXTERNAL routine.

Strings that are passed from your program to FMP should have a current length that indicates to FMP the part of the string FMP wants. The current length can include trailing blanks but should not include uninitialized areas of the string.

Strings that FMP sets to an initial value and passes back to your program should have a current length large enough to hold the number of characters expected from FMP (usually a maximum of 63 characters). The length must be greater than zero; FMP truncates or blank pads as necessary. The contents of the string within the current length do not need to be initialized.

Strings that your program passes to FMP and that FMP modifies and returns should have a current length large enough to hold the number of characters expected from FMP. The strings must be blank padded from the end of the data being passed to FMP, out to the current length.

The following Pascal program uses FIXED_STRING to call FMP routines. Note that while a constant is used as the file name to the FmpOpen call, any Pascal string variable or expression with a length less than or equal to the length of the string type PATH could be used. Also, note that anywhere FMP expects a FORTRAN 77 character string parameter, a Pascal string type must be specified in the EXTERNAL declaration and the FIXED_STRING compiler option must be in the ON state.

```
PROGRAM fmpexample;

CONST
   max_file_path = 64;
   dcb_words     = 144;
   welcome_file  = '/SYSTEM/WELCOME.CMD';

TYPE
   INT  = -32768..32767;
```

```
   PATH = STRING [max_file_path];
   DCB  = ARRAY [1..dcb_words] OF INT;

VAR
   error_number:  INT;
   error_message: PATH;
   file_dcb:      DCB;
   terminal:       TEXT;

$FIXED_STRING ON$

PROCEDURE FmpOpen
   (VAR dcb:  DCB;
    VAR err:  INT;
        name: PATH;
        opts: PATH;
        bufs: INT); EXTERNAL

PROCEDURE FmpError
   (    err:  INT;
    VAR mess: PATH); EXTERNAL

$FIXED_STRING OFF$

BEGIN
  rewrite (terminal, '1', 'NOCCTL');

  FmpOpen (file_dcb, error_number, welcome_file, 'ROS', 1);

{Check for error on open.  If error occurred, make the current}
{length long enough to hold the message, get the error message}
{from FMP, trim blank padding, and display the message on the }
{terminal.                                                    }

  IF error_number < 0 THEN BEGIN
     setstrlen (error_message, strmax (error_message));
     FmpError (error_number, error_message);
     error_message := strrtrim (error_message);
     writeln (terminal, welcome_file, '(', error_message, ')');
  END
  ELSE BEGIN
     .
     .
     .
  END;
END.
```

File Descriptors in Macro

This section describes how to call the StrDsc subroutine from a Macro program to convert character string file descriptors to a format that can be processed by the program and used by FMP.

All FMP calls that take a character string will require the caller to pass a file descriptor. FORTRAN 77 does this automatically, but Macro users must set up and pass their own filedescriptors. Note that these FMP calls do not work when a buffer of characters is passed as a parameter when a string is expected.

The StrDsc subroutine returns a two-word descriptor that describes a character buffer of a specified length, beginning at a specified character position. The characters in the buffer are numbered from 1 to the number of characters. The resulting two-word descriptor can be passed as an input or output parameter anywhere a FORTRAN 77 character string parameter is required. The string is transferred to and from the buffer described by the two-word descriptor. StrDsc is described in the Relocatable Libraries Reference Manual - RTE-A . RTE-6/VM.

The following example opens a file with a known name and options string:

```
          ext FmpOpen, StrDsc
*
*   Create a filedescriptor for the name
*
          jsb StrDsc
          def *+4
          def nbuffer
          def =d1
          def =d19
          dst filename
*
*  And the options string
*
          jsb StrDsc
          def *+4
          def obuffer
          def =d1
          def =d3
          dst options
*
*  Open the file
*
          jsb FmpOpen
          def *+6
          def dcb
          def err
```

```
          def filename
          def options
          def =d1
          ...
*
*  Constants and data
*
nbuffer   asc 10,WELCOME.CMD::SYSTEM
obuffer   asc 2,ROS
filename  bss 2
options   bss 2
dcb       bss 144
```

Note that

```
          jsb FmpOpen
          def *+6
          def dcb
          def err
          def nbuffer         ; wrong!
          def obuffer         ; also wrong!
          def =d1
```

does not work with nbuffer and obuffer declared as above.

Error Returns

Errors can occur on FMP calls; for example, it is an error to try to open a non-existent file. The error is returned as a negative value, either as the function return value or in an error parameter. The error value can be passed to an error processing or reporting subroutine in your program. The error return values are listed in Appendix A. The FMP routines must be declared as integer functions in FORTRAN to receive the correct error code as the function return value.

Transferring Data to and from Files

In addition to the Data Control Block, a user buffer must be defined in the calling program for transferring individual records to and from files. Records to be sent to files must be stored in the user buffer before a write call. Records read from files are returned to the user buffer. The relationship between the user buffer, the Data Control Block buffer and a disc file is illustrated in Figure 6-1.

Each call that reads or writes a record transfers one record between the user buffer and the Data Control Block buffer. Such transfers within memory are known as logical reads or writes.

A physical read or write transfers a block of data between the disc file and the Data Control Block buffer. A physical write is performed automatically when the DCB buffer is full, when a file is closed, or when a request for a physical write is made with the FmpPost call.

On a read request, a block of data is physically read into the DCB buffer from the disc only if the entire requested record is not already in the buffer. If a needed record is not already within the DCB buffer, (see record 7 in Figure 6-1), then FMP performs physical reads or writes of blocks until the entire record has been transferred.

For type 1 file accesses, the intermediate transfer to the DCB buffer is omitted and each 128-word record is transferred directly between the user buffer and the file as shown in Figure 6-2. Such accesses are faster than transfers through the DCB buffer.

Non-disc (type 0) file reads and writes also bypass the DCB buffer. Records in type 0 files are written or read directly to or from the device identified as a type 0 file. Words, rather than records, are the units of type 0 transfers, to accommodate the record lengths of various devices.

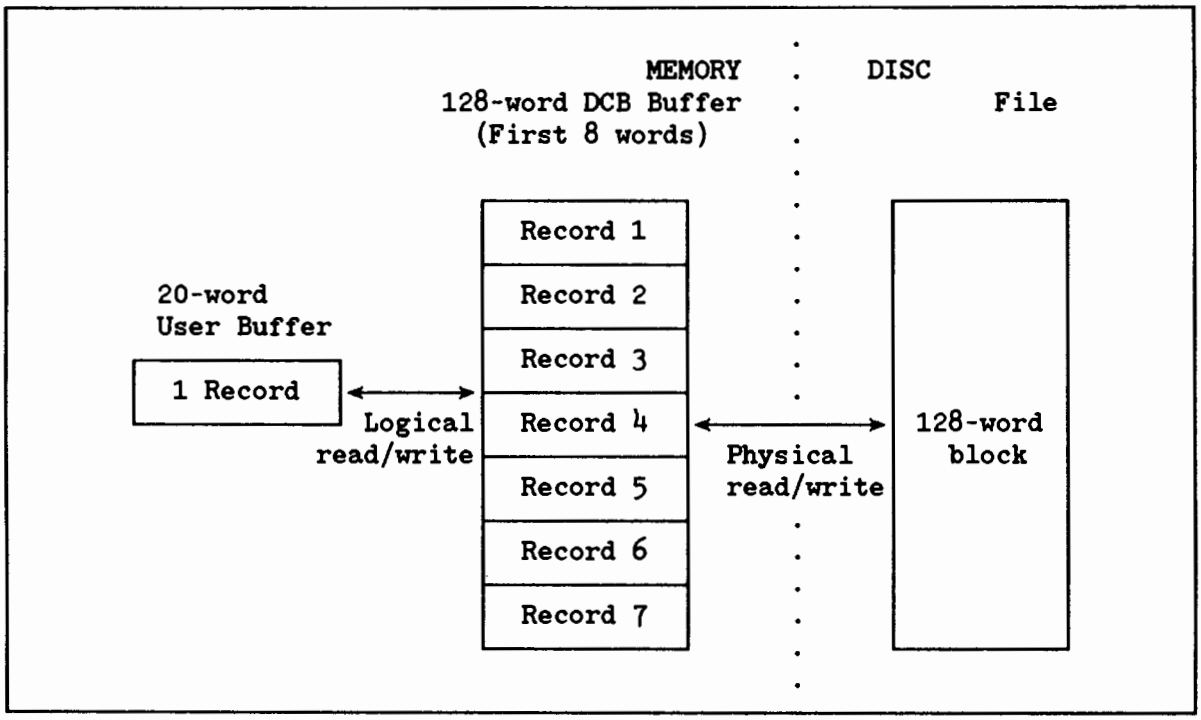


Figure 6-1. Logical Transfer Between Disc File and Buffers

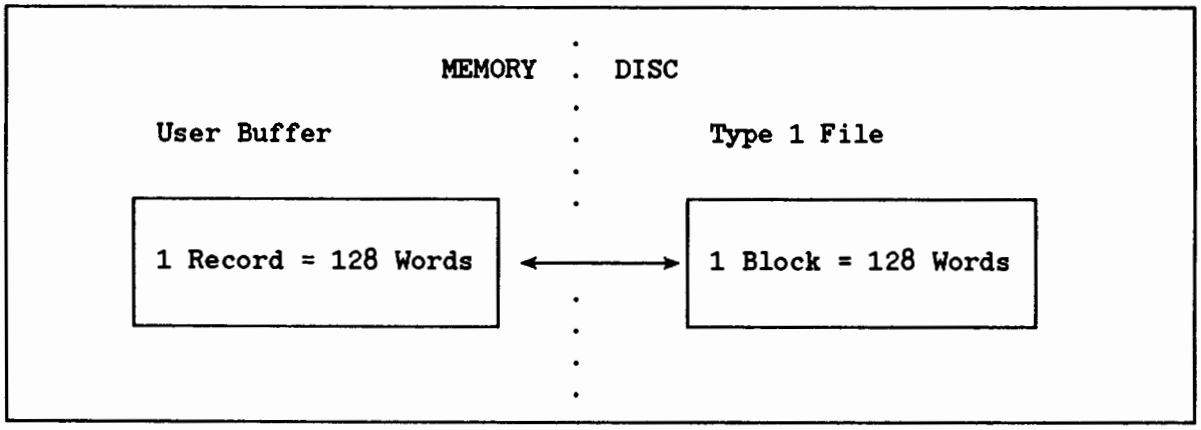


Figure 6-2. Data Transfers with Type 1 Files

Descriptions of FMP Routines

This section contains descriptions of all FMP routines; the routines are listed alphabetically. Tables 6-1 through 6-6 present functional groupings of the routines.

Table 6-1. File Manipulation FMP Routines

FMP ROUTINE	PURPOSE
FmpOpen	Opens a file for access
FmpOpenScratch	Opens file on scratch directory
FmpClose	Closes a file to end access
FmpRead	Reads from a file
FmpReadString	Reads a character string from a file
FmpWrite	Writes to a file
FmpWriteString	Writes a character string to a file
FmpPosition	Returns the current file position
FmpRewind	Sets file position to the first word of the file
FmpSetPosition	Changes the file position
FmpSetWord	Changes the file position
FmpAppend	Positions a file to the EOF mark
FmpSetEof	Sets EOF mark at the current position
FmpPost	Posts data to the file
FmpTruncate	Truncates the file
FmpSetDcbInfo	Changes information in the DCB
DcbOpen	Indicates if a DCB is open

Table 6-2. Directory Access FMP Routines

FMP ROUTINE	PURPOSE
FmpCreateDir	Creates a directory
FmpWorkingDir	Returns the working directory
FmpSetWorkingDir	Changes the working directory
FmpInfo	Returns the directory information for the file
FmpSetDirInfo	Changes information in a directory
FmpMount	Mounts a volume
FmpDismount	Dismounts a volume
FmpFileName	Returns the full path name of a file
FmpOpenFiles	Indicates which files in a directory are open
FmpOwner	Returns the name of the directory owner
FmpSetOwner	Changes the name of the directory owner
FmpCreateTime	Returns the time that the file was created
FmpAccessTime	Returns the time of the last access
FmpUpdateTime	Returns the time of the last update
FmpRecordCount	Returns the number of records in the file
FmpRecordLen	Returns the length of the longest record in the file
FmpProtection	Returns the access available to file or directory
FmpSetProtection	Changes the access to a file or directory
FmpEof	Returns the position of the EOF mark
FmpSize	Returns the physical size of the file
FmpRename	Changes the file name
FmpPurge	Purges a file
FmpUnPurge	Restores a purged file
FmpUdspInfo	Returns current UDSP information for the session
FmpUdspEntry	Returns the directory name in specified UDSP entry

Table 6-3. Masking FMP Routines

FMP ROUTINE	PURPOSE
FmpInitMask	Initializes data structures for the FMP mask calls
FmpNextMask	Returns the directory entry for the next file matching
FmpMaskName	Builds a full name for a file matching the mask
FmpEndMask	Closes the files associated with a mask search
WildCardMask	Checks for wildcard characters in a mask
FattenMask	Modifies the mask
MaskOldFile	Determines if a specified file is an FMGR file
MaskMatchLevel	Returns the directory level of the last file matched
MaskOpenId	Returns the D.RTR open flag of the last file returned by FmpNextMask
MaskSecurity	Returns the security code of the last file returned by FmpNextMask
Calc_Dest_Name	Creates a destination file name from a file name, match level, and destination mask

Table 6-4. Device FMP Routines

FMP ROUTINE	PURPOSE
FmpBitBucket	Determines whether type 0 file is LU 0
FmpDevice	Indicates whether a DCB is associated with a device file
FmpInteractive	Indicates whether a DCB is associated with an interactive device
FmpIoOptions	Returns the I/O options word
FmpSetIoOptions	Changes the I/O options word
FmpIoStatus	Returns the A- and B-Register values of last I/O request
FmpControl	Issues a control request to an LU
FmpLu	Returns the LU of the file or device

Table 6-5. Parsing FMP Routines

FMP ROUTINE	PURPOSE
FmpBuildHierarch	Builds a file descriptor in hierarchical format from its component fields
FmpBuildName	Builds a file descriptor from its component fields
FmpBuildPath	Builds a file descriptor that includes hierarchical directory information and file masks from its component fields
FmpHierarchName	Converts a file descriptor to hierarchical format
FmpStandardName	Converts a file descriptor to the standard format
FmpLastFileName	Returns the last file name in a path
FmpParseName	Parses a file descriptor into its component fields
FmpParsePath	Parses a file descriptor that includes hierarchical directory information and file masks into its component fields.
FmpShortName	Returns the shortened version of a file descriptor
FmpUniqueName	Creates and returns a unique file name

Table 6-6. Utility FMP Routines

FMP ROUTINE	PURPOSE
DcbOpen	Indicates whether a DCB is open
FmpCopy	Copies a file to another file
FmpList	Lists a file to a specified LU
FmpError	Returns an error message for an FMP error code
FmpReportError	Prints an error message for an FMP error on LU 1
FmpExpandSize	Unpacks file size word to double integer
FmpPackSize	Packs double integer file size into one word
FmpCloneName	Generates program clone names
FmpRpProgram	Restores a program
FmpRunProgram	Schedules a program
FmpRwBits	Checks a string for the letters R and W

Calc_Dest_Name

Calc_Dest_Name generates a full destination file name.

```
call Calc_Dest_Name(sourcename,matchlevel,destmask,destname)
character*(*) sourcename, destmask, destname
integer matchlevel
```

sourcename

A character string that returns a full source file descriptor.

matchlevel

An integer that specifies the number of the directory level in which the last file was matched as returned by MaskMatchLevel.

destmask

A character string that specifies the destination mask.

destname

A character string that returns the full destination file descriptor.

Calc_Dest_Name uses a file name, its matchlevel (returned by the MaskMatchLevel routine), and a destination mask, and generates a full destination file name. If the destination mask contains an "@" in the filename or file type extension fields, then the sourcename values of those fields are used. The Command Interpreter (CI) CO and MO commands use Calc_Dest_Name generated destination names.

DcbOpen

DcbOpen returns an integer value that indicates whether or not the specified DCB is open.

```
error = DcbOpen(dcb,error)
integer dcb(*), error
```

dcb

An integer array containing the DCB for the file.

error

An integer indicating the status of the DCB. If the DCB is open, error is set to zero. If the DCB is not open, error is set to a negative error code.

FattenMask

FattenMask modifies the mask parameter by adding the character "@" to the name or file type extension if it is implied by the mask.

```
call FattenMask(mask,how)
character*(*) mask
integer how
```

mask

A character string specifying the mask to be modified.

how

An integer specifying how the mask is to be modified. If bit 0 is set, a "D" is appended to the qualifier. If bit 1 is set and the mask is blank, "@" is not inserted in either the name or file type extension.

If the name field of mask is blank, the "@" character replaces the blank. If the name field ends with "@" and the file type extension is omitted, then a file type extension of ".@" is inserted. If the mask is a global directory in the form /global, the file type extension .DIR is appended because it is the only file type extension possible for a global directory.

The overall purpose of this call is to make implied constructs such as /DIR/ explicit, by converting them to the fuller /DIR/@.@.D described in the last paragraph.

FmpAccessTime

FmpAccessTime returns the time of last access for the named file. The file does not have to be open, and it is not opened to read the access time.

```
error = FmpAccessTime(filedescriptor,time)
character*(*) filedescriptor
integer*4 time
```

filedescriptor

A character string that specifies the name of the file.

time

A double integer that returns the time of the last access expressed as the number of of seconds since Jan 1, 1970.

The access time is changed when a file is opened. It is not affected by calls which do not open the file, such as FmpRead or FmpClose. Access time is generally used to check activity on a file; inactive files that have outlived their usefulness, are often purged to make room for other files.

Routines are available to convert the returned time to an ASCII string. Usually, however, the returned time is compared to other times in the same format, so it may not be necessary to convert the returned time.

FmpAppend

FmpAppend positions a file of type 3 or above to the end-of-file mark to prepare for adding records to the file.

```
error = FmpAppend(dcb,error)
integer dcb(*), error
```

dcb

An integer array containing the DCB for the file.

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

The file must be open for write access, and must be a type 3 or above file; FmpAppend has no effect on device files, or type 1 and 2 files. FMGR files must be open write and read access.

The effect of FmpAppend is the same as calling FmpEof and using the returned value in an FmpSetPosition call to position the file to the EOF. FmpAppend removes one step from the process.

FmpBitBucket

FmpBitBucket determines if the type 0 file associated with the specified DCB is LU 0 (the bit bucket).

```
bool = FmpBitBucket(dcb)
logical bool
integer*2 dcb(*)
```

dcb

An integer array containing the DCB for the type 0 file.

bool

A flag that is set to TRUE (-1) if the DCB is open and associated with a type zero file, and the device is LU 0; otherwise, bool is set to FALSE (0).

FmpBuildHierarch

FmpBuildHierarch constructs a file descriptor in the hierarchical format.

```
call FmpBuildHierarch(filedescriptor,dirpath,name,typex,qual,
                      sc,type,size,rl,ds)
character*(*) filedescriptor, dirpath, name, typex, qual, ds
integer sc, type, size, rl
```

filedescriptor

A 63-character string that returns the file descriptor.

dirpath

A character string specifying the directory/subdirectory path. Dirpath can be a maximum of 63 characters.

name

A character string specifying the filename. Name can be a maximum of 63 characters.

typex

A character string specifying the file type extension. Typex can be a maximum of 4 characters.

qual

A character string specifying the mask qualifier. Qual can be a maximum of 40 characters.

sc

An integer that specifies the security code of an FMGR file.

type

An integer that specifies the file type.

size

An integer that specifies the size of the file in blocks.

rl

An integer that specifies record length.

ds

A character string that specifies the DS node name, a user name, or both. Ds can be a maximum of 63 characters.

Dirpath must conform to the following conventions:

- The global directory and each subdirectory name be followed by a slash (/).
- Dirpath must begin with a slash except in the following cases:
  - If the filedescriptor is specified relative to the working directory and one or more subdirectories are specified, dirpath must begin with the name of the highest-level subdirectory (for example, SUBDIR1/SUBDIR2).
  - If the filedescriptor is specified relative to the working directory and no subdirectories are specified, dirpath must be blank.

If any of the component fields are zero or blank, the corresponding field in the filedescriptor parameter is left empty, with any necessary place holders. All delimiters except those in the DS field are automatically inserted. The DS delimiters must be included in the DS parameter string. Trailing fields that are zero or blank are omitted without placeholders. There is no error detection for the component fields, so illegal parameters generate an illegal file descriptor.

FmpBuildName

FmpBuildName creates a file descriptor from its component fields. It is the inverse of FmpParseName. Its call sequence is the same as FmpParseName, but the component fields are specified, and the file descriptor is returned.

```
call FmpBuildName(filedescriptor,name,typex,sc,dir,type,size,rl,ds)
character*(*) filedescriptor, name, typex, dir, ds
integer sc, type, size, rl
```

filedescriptor

A 63-character string that returns the file descriptor.

name

A character string that specifies the filename. Name can be a maximum of 63 characters.

typex

A character string that specifies the file type extension. Typex can be a maximum of 4 characters.

sc

An integer that specifies the security code of an FMGR file.

dir

A character string that specifies the directory name. Dir can be a maximum of 16 characters.

type

An integer that specifies the file type.

size

An integer that specifies the size of the file in blocks.

rl

An integer that specifies record length.

ds

A character string that specifies the DS node name, a user name, or both. It can be a maximum of 63 characters.

If any of the component fields are zero or blank, the corresponding field in the filedescriptor parameter is left empty, with any necessary place holders. All delimiters except those in the DS field are automatically inserted. The DS delimiters must be included in the DS parameter string. Trailing fields that are zero or blank are omitted without placeholders. There is no error detection for the component fields, so illegal parameters generate an illegal file descriptor.

FmpBuildName example:

Assume that name = SANJOSE and dir = CITIES.

```
call FmpBuildName(fdesc,name,'txt',0,dir,4,24,0,' ')
```

Fdesc returns SANJOSE.TXT::CITIES:4:24.

FmpBuildPath

FmpBuildPath constructs a file descriptor from its component fields. It is similar to FmpBuildName, except that it more conveniently constructs file descriptors that contain hierarchical directory information, and it permits creation of file descriptors that contain a file mask qualifier. It is also similar to FmpBuildHierarch except that it creates file descriptors in the standard format, described in the FmpStandardName section.

```
call FmpBuildPath(filedescriptor,dirpath,name,typex,qual,
                  sc,type,size,rl,ds)
character*(*) filedescriptor, dirpath, name, typex, qual, ds
integer sc, type, size, rl
```

filedescriptor

A 63-character string that returns the file descriptor.

dirpath

A character string that specifies the directory/subdirectory path. Dirpath can be a maximum of 63 characters.

name

A character string that specifies the filename. Name can be a maximum of 63 characters.

typex

A character string that specifies the file type extension. Typex can be a maximum of 4 characters.

qual

A character string that specifies the mask qualifier. Qual can be a maximum of 40 characters.

sc

An integer that specifies the security code of an FMGR file.

type

An integer that specifies the file type.

size

An integer that specifies the size of the file in blocks.

rl

An integer that specifies record length.

ds

A character string that specifies the DS node name, a user name, or both. Ds can be a maximum of 63 characters.

Dirpath must conform to the following conventions:

- The global directory and each subdirectory name must be followed by a slash (/).
- Dirpath must begin with a slash, except in the following cases:
    - If the file descriptor is specified relative to the working directory and one or more subdirectories are specified, dirpath must begin with the name of the highest-level subdirectory, as in SUBDIR1/SUBDIR2/.
    - If the file descriptor is specified relative to the working directory and no subdirectories are specified, dirpath must be blank.

If any of the component fields are zero or blank, the corresponding field in the filedescriptor parameter is left empty, with any necessary place holders. All delimiters except those in the ds and dirpath fields are automatically inserted. The DS and hierarchical directory path delimiters must be included in the ds and dirpath parameters. Trailing fields that are zero or blank are omitted without placeholders. There is no error detection for the specified parameters, so illegal parameters generate an illegal file descriptor.

FmpBuildPath is the inverse of FmpParsePath. It has the same calling sequence, and uses the same parameters, except that the component fields are specified and a file descriptor is built and returned.

FmpBuildPath example:

Path = /CITIES/CALIFORNIA/, file = @, qual = D.

```
    CALL FmpBuildPath(fdesc,path,file,'TXT','D',0,4,24,0,' ')
```

Fdesc returns /CITIES/CALIFORNIA/@.TXT.D:::4:24

FmpCloneName

FmpCloneName generates program clone names that can be used by FmpRpProgram.

```
Call FmpCloneName(name,init)
character*(*) name
logical init
```

name

A character string that specifies the program name to be cloned. The specified name is modified by the system and returned to the calling program.

init

A logical indicating whether the current call is the first call to FmpCloneName.

Before calling FmpCloneName for the first time, set the init parameter to TRUE (-1). When the call is executed, FmpCloneName resets the value to FALSE (0).

The sequence of names generated by FmpCloneName is as follows (PROG is the original program name):

PROG, PRO.A, PRO.B, ..., PRO.Z, PROAA, PROAB, ..., PROZZ

FmpCloneName can be called in a loop to generate program names until a name that does not already exist on the system is found. This name then can be used in an FmpRpProgram call to RP a program.

FmpClose

FmpClose closes a file, and removes its entry from the FMP open file table.

```
error = FmpClose(dcb,error)
integer dcb(*), error
```

dcb

An integer array containing the DCB for the file.

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

If the program wrote data to the file while it was open, the FmpClose call sets the time of last update to the system time when the file is closed. It also sets the backup bit in the directory. FmpClose also sets the end-of-file position in the directory to the file position at the time of the close, if the DCB specified a sequential file positioned at EOF.

Files should be closed after a program's access is finished, to make sure that all writes are posted to the disc, and to unlock files or devices to make them available to other programs. It is good practice to close files after access is finished, whether or not write accesses were performed.

FmpControl

FmpControl performs a control request on the LU associated with a device file DCB.

```
error = FmpControl(dcb,error,pram1,pram2,pram3,pram4)
integer dcb(*), error, pram1, pram2, pram3, pram4
```

dcb

An integer array containing the DCB of a device file.

error

An integer that returns a negative code if an error occcurs or zero if no error occurs.

pram1 - pram4

Integers that can be passed as control request parameters if required.

FmpCopy

FmpCopy copies one file to another.

```
error = FmpCopy(name1,err1,name2,err2,buffer,buflen,options)
character*(*) name1, name2, options
integer buffer(*), buflen, err1, err2
```

name1

A character string that specifies the source file or logical unit.

err1

An integer that returns errors associated with name1.

name2

A character string that specifies the destination file or logical unit.

err2

An integer that returns errors associated with name2.

buffer

An integer buffer that contains the source and destination DCBs and DCB buffers. Buffer must be a minimum of 288 words in length.

buflen

An integer that specifies the length of the buffer in words. Buflen must be set to at least 288 words.

options

A character string that specifies the data transfer mode if the source or destination is a device, as well as manipulation of the source and destination files. Options can be set to any of the following values:

A	ASCII
B	Binary
C	Clear backup bit on source
D	Overwrite existing file
N	Source does not have carriage control
P	Purge source after copy

FmpCopy works for all file types, including type 6 files, and type 1 or 2 files with missing extents. It uses the most efficient copy operation that works for the given files.

The calling program must specify a work buffer to contain the source and destination file DCBs and transferred records. The buffer must be at least large enough to contain two DCBs of 16 words each plus two 128-word (one block) DCB buffers. The minimum buffer size, thus, is (2 * 16) + (2 * 128) = 288 words. The larger the buffer is, the faster the copy operation can execute. Larger buffers must be larger by 256-word increments (2 * 128 words per block).

The A and B options are used only when the source or destination is a device or a type 3 or 4 file, and the destination is a device. If the destination is a device or a type 3 or 4 file, and the source is a device, the default option is A. In all other cases, the default option is B.

If the destination name does not specify a file type, the source file type is used. If the source is a device and the A option is in effect, the default destination type is 3, if the B option is in effect, the default destination type is 6.

If the destination name does not specify a size, the total size of the source file (the sum of the sizes of the main and all its extents) is used. As a result, the destination file does not have any extents. If the source is a device, the default size is 24 blocks.

FmpCopy tests the break flag while copying. If it finds it set, it stops copying and reports error -235 (Break Detected). If the calling program uses the break flag, it should use the error indication to detect breaks when FmpCopy is used.

If either err1 or err2 contains an error code, the same error code is returned in error. If error = 0, then neither err1 nor err2 contains an error code.

FmpCreateDir

FmpCreateDir creates a directory.

```
error = FmpCreateDir(name,lu)
character*(*) name
integer lu
```

name

A character string specifying the name of the directory to be created.

lu

An integer specifying the disc LU on which to create the directory.

A global directory is specified by a name beginning with "::" or "/", as in ::USERS or /USERS. A subdirectory is specified with its parent directory, separated by "::", as in SUBDIR::USERS or /DIR/SUBDIR/. The parent directory must already exist.

The calling program can specify a size (::DIRNAME::12), but the file system should set the directory size automatically. The default size is chosen to make the most efficient use of the disc on which the directory is created.

Subdirectories will be placed on the same LU as their parent directory. Global directories will go on the specified logical unit. If logical unit zero is specified, the global directory will be created on the same logical unit as the working directory, if any, or on the lowest numbered disc logical unit on which directories can be created.

FmpCreateTime

FmpCreateTime returns the time of creation for the named file. The file is not opened in the process.

```
error = FmpCreateTime(filedescriptor,time)
character*(*) filedescriptor
integer*4 time
```

filedescriptor

A character string that specifies the name of the file.

time

A double integer that returns the time that the file was created, expressed in seconds since January 1, 1970.

The create time is set when the file is created, and is never changed afterwards, except by the FmpSetDirInfo routine.

Routines are available to convert the returned time to an ASCII string. Usually, however, the returned time is compared to other times in the same format, so the calling program may not have to convert the format.

FmpDevice

FmpDevice indicates whether the specified DCB is associated with a device file.

```
bool = FmpDevice(dcb)
logical bool
integer dcb(*)
```

dcb

An integer array containing the DCB of the file.

bool

A boolean set to TRUE (-1) if the specified DCB is associated with a device file. Bool is set to FALSE (0) if the DCB is associated with a disc file or is not open.

FmpDismount

FmpDismount dismounts a disc volume.

```
error = FmpDismount(lu)
integer error, lu
```

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

lu

An integer that specifies the LU of the disc volume.

Global and subdirectories on the specified LU are made unavailable, and the disc is removed from the cartridge list.

If there are any open files, RP'd programs, or working directories on the volume, D.RTR will report an error identifying the first such conflict that it finds.

FmpEndMask

FmpEndMask closes the files associated with a mask search.

```
call FmpEndMask(dirdcb)
integer dirdcb(*)
```

dirdcb

An integer array initialized by FmpInitMask.

FmpEnd Mask should always be called after a masked search terminates. If it is not called, directories may be left open to your program after the search ends.

FmpEof

FmpEof returns the current word position of the end-of-file mark for the specified file.

```
error = FmpEof(filedescriptor,eofpos)
integer error
character*(*) filedescriptor
integer*4 eofpos
```

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

filedescriptor

A character string specifying the name of the file.

eofpos

An integer that returns the word position of the last word in the main file area, or of the highest numbered extent, if any, plus 1.

The first word in the file is word 0, so if eofpos = 0 for a file of type 3 or above, the file is empty. For type 1 or 2 files, eofpos is the word position of the last word in the main file area, or of the highest numbered extent, if any, plus 1.

If the file is currently open, the returned value may not be accurate because the program that has it open may have added to the file without posting its DCB.

FmpError

FmpError returns a string that describes the error identified by the error parameter. FmpError should be used to report errors to ensure consistent error reporting.

```
call FmpError(error,message)
character*(*) message
integer error
```

error

An integer that specifies the error code.

message

A character string variable that returns an error message (for example, NO SUCH FILE or CAN'T PURGE FILE).

The list of possible messages is given in Appendix A. The maximum error description length is 30 characters. If there is not a defined error message for the error identified by the error parameter, a generic error message in the form "FMP error -xxx" is issued by the system.

The system program D.ERR generates the text of FMP error messages. If an FMP error occurs and the system cannot find D.ERR, the following message is generated:

```
(warning -250) FMP error xxx
```

The error code -250 indicates that D.ERR was not available and xxx is the FMP error that occured.

FmpError should be used by programs that need more flexible error processing than is provided by FmpReportError.

FmpExpandSize

FmpExpandSize unpacks the size word into a double integer value that specified the number of blocks in the file.

```
blocks = FmpExpandSize(size)
integer size
integer*4 blocks
```

blocks

A double integer indicating the number of blocks in the file.

size

An integer indicating the size of the file, in one word.

If size > 0, then the number is not changed. If size < 0, it is multiplied by -128.

For FMGR files, the packed size must be divided by 2 if it is positive, before the call to FmpExpandSize. If the size parameter of an FMGR file is negative, it works just as an FMP file size.

FmpFileName

FmpFileName returns the full file descriptor of the file associated with the specified DCB.

```
error = FmpFileName(dcb,error,filedescriptor)
integer dcb(*), error
character*(*) filedescriptor
```

dcb

An integer array containing the DCB for the specified file.

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

filedescriptor

A character string that returns the name of the file associated with the specified DCB. The file descriptor includes the full directory path, and file type, size, and (for type 2 files) record length, returned in decimal ASCII. The size is the total size of the file, including extents. For remote files, the file descriptor includes the user name and remote node name.

The normal string assignment rules apply to the returned string, although FmpFileName never returns a file descriptor longer than 63 characters. The file descriptor will be truncated to fit in 63 characters, even if it causes an incorrect name to be returned by truncating part of the file name or the directory name.

FmpFileName can be used to return the file descriptor of an open file for use in other calls that need a file descriptor, or for use in error reporting routines. The DCB must be open when the call is made.

FmpHierarchName

FmpHierachName converts a file descriptor to the hierarchical format, in which leading (/DIR/FILE) directory notation, rather than trailing (FILE::DIR), is always used.

```
call FmpHierarchName(filedescriptor)
character*(*) filedescriptor
```

filedescriptor

A character string containing the file descriptor to be converted.

Hierarchical names are much easier to use in programs that manipulate hierarchical directory structures. They cannot be used for FMGR files, however, so programs that must process FMGR files should call FmpStandardName to convert names to the FMGR-compatible standard format before passing the file descriptor to routines such as FmpOpen.

FmpInfo

FmpInfo returns a copy of the directory entry for the file specified by the DCB. It allows the calling program to get all of the information in the directory with minimum delay. This call should not be used unless absolutely necessary because it is likely to be affected by any future changes to the directory structure.

```
error = FmpInfo(dcb,error,info,flag)
integer dcb(*), error, info(32), flag
```

dcb

An integer array containing the DCB for the file.

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

info

A 32-word integer array into which the directory information directory information is returned. For FMGR, only the first 16 words are used; the last 16 words are zeros.

flag

An integer flag indicating the file system required; 0 for FMGR files and non-zero for FMP files.

FmpInitMask

FmpInitMask initializes the buffers, pointers, and control constructs used by FmpNextMask to select file names according to a file mask.

```
error = FmpInitMask(dirdcb,error,mask,diropenname,dcblen)
integer dirdcb(*), error, dcblen
character*(*) mask, diropenname
```

dirdcb

A control array to be used only with FmpNextMask.

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

mask

A character string which specifies a set of files. The mask format is:

```
dirpath/name.typex.qual:sc:dir:type:size:rl
```

diropenname

The returned character string directory path.

dcblen

The length of dirdcb in words.

Dirdcb and diropenname must not be altered between the FmpInitMask call and the FmpNextMask calls that follow. Dirdcb must be at least 356 words long.

The program example at the end of this chapter shows how FmpInitMask, FmpNextMask, FmpMaskName, FmpLastFileName, and FmpEndMask are related and work together.

The fields in the mask qualifier of particular interest to FmpInitMask are dir, dirpath, and qual. Using the dir and dirpath information the appropriate directory is opened in preparation for checking entries. If the search qualifier (qual) is included, its state is recorded to let FmpNextMask perform the search in the correct order. For a complete description of the mask qualifier, refer to the DL command description in Chapter 5 of this manual.

FmpInteractive

FmpInteractive returns a boolean value that reports whether or not the specified DCB is associated with an interactive device.

```
bool = FmpInteractive(dcb)
logical bool
integer dcb(*)
```

bool

A boolean that is set to TRUE (-1) if the specified DCB is associated with an interactive device. Bool is set to FALSE (0) if the specified DCB is not associated with an interactive device.

dcb

An integer array containing the DCB for the file.

FmpIoOptions

FmpIoOptions returns the 16-bit I/O option word for the specified DCB.

```
error = FmpIoOptions(dcb,error,options)
integer dcb(*), error, options
```

dcb

An integer array containing the DCB for the file.

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

options

An integer that returns the 16-bit I/O option word.

The upper ten bits of the option word correspond to the upper ten bits of the CNTWD that is used in EXEC calls. The returned option word is described in the Standard I/O chapter of this manual.

The value returned is undefined if the DCB does not represent a device file.

FmpIoStatus

FmpIoStatus returns the values in the A- and B-Registers after the last I/O request.

```
call FmpIoStatus(areg,breg)
integer areg, breg
```

areg

A one-word integer containing the value of the A-Register.

breg

A one-word integer containing the value of the B-Register.

Because it does not specify a DCB, FmpIoStatus returns the values of the A- and B-Registers saved after the last FmpRead or FmpWrite I/O request. The status information in the registers is guaranteed to be accurate only if FmpIoStatus is called immediately after the I/O operation that posted status in the registers.

The value returned is the status and transmission log of a successful request, or a two-word error return for an unsuccessful request. Unsuccessful requests are identified by an error code = -17.

FmpLastFileName

FmpLastFileName extracts the file name from the passed file descriptor.

```
call FmpLastFileName(filedescriptor,lastname)
character*(*) filedescriptor, lastname
```

filedescriptor

A character string that specifies the complete file descriptor.

lastname

The filename, portion of filedescriptor. The filename is identified as the characters between the slash after the directory path (if any) and the first period or colon.

For example, FmpLastFileName ('SUB/FILE.TXT:::3',last) returns "FILE".

FmpList

FmpList lists a file to the specified LU.

```
error = FmpList(filedescriptor,lu,option,rec1,rec2)
character*(*) filedescriptor, option
integer*4 rec1, rec2
integer error, lu
```

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

filedescriptor

A character string that specifies the name of the file.

lu

An integer that specifies the output LU.

option

A character string that selects the format of the output. The values are as follows:

A	ASCII output
B	Binary output displayed as octal

File types 0, 3, and 4 default to A; other file types default to B.

rec1

A double integer that specifies the first record to be listed.

rec2

A double integer that specifies the last record to be listed.

If both rec1 and rec2 are set to 0, the entire file is listed.

By default, the listing to an interactive device pauses after printing 22 lines. When it pauses, FmpList prompts you for one of three legal responses:

a	Abort the listing
<space>	List another 22 lines
<cr>	List the remainder of the file without pausing

If the LU is not interactive, the listing does not pause.

FmpLu

FmpLu returns the LU of the file or device associated with the specified DCB.

```
lu = FmpLu(dcb)
integer dcb(*), lu
```

dcb

An integer array containing the DCB for the file.

lu

An integer indicating the LU number of the file or device associated with the specified DCB.

If the DCB is associated with a type zero file, the value returned in the lu parameter is the number of the device LU. If the DCB is associated with a disc file, the value returned is the LU of the disc on which the file resides. If the specified DCB is not open, a -11 (DCB not open error) error is returned.

FmpMaskName

FmpMaskName builds a full file descriptor from the entry and curpath parameters returned by a call to FmpNextMask.

```
call FmpMaskName(dirdcb,newname,entry,curpath)
character*(*) newname, curpath
integer dirdcb(*), entry(32)
```

dirdcb

A control array, initialized by FmpInitMask.

newname

A character string that returns the file descriptor.

curpath

A character string directory path returned by FmpNextMask.

entry

A 32-word directory entry returned by FmpNextMask.

The file descriptor returned to newname includes all of the fields specified by entry (name, file type extension, full directory specification, type, size and record length). Null fields are omitted in the file descriptor.

The names generated by FmpMaskName often exceed the 63-character file system limit, because the names include the type, size and at least four colons.

FmpMount

FmpMount mounts a disc volume.

```
error = FmpMount(lu,flag,blks)
integer lu, flag, blks
```

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

lu

An integer that specifies the system LU of the disc.

flag

An integer that determines whether to initialize the disc before mounting it. The values of flag are:

0 Do not initialize before mounting
1 Initialize if the disc does not have a valid directory
2 Initialize disc before mounting

blks

An integer that specifies the number of blocks to leave free at the beginning of the volume. These blocks are never allocated to files or directories; they are used to contain bootable programs such as BOOTEX or an off-line utility.

When a volume is mounted, the disc becomes available to the system, global directories can be made available, and the disc space can be used by its owner. An entry is made in the cartridge list to let the system remount the volume automatically after a system shutdown.

It is an error to mount a disc that is already mounted, or to try to mount a non-disc LU.

FmpNextMask

FmpNextMask returns the directory entry for the next file in the directory.

```
more = FmpNextMask(dirdcb,error,curpath,entry)
logical more
integer dirdcb(*), error, entry(32)
character*(*) curpath
```

more

A boolean variable that indicates whether the search can continue. It is set TRUE (-1) if there is another entry to be searched, whether or not an error occurred. If it is TRUE and an error has occurred, the current entry is not valid. It is set FALSE (0) if an error occurred that prevents successful continuation of the current search process.

dirdcb

A control array, initialized by FmpInitMask.

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

curpath

The returned character string directory path.

entry

A 32-word array which returns the directory entry for each file found.

For recoverable errors, the calling program can determine the response, and then it can terminate or continue the search.

When the search is complete, error will return a 0 and variable MORE will be FALSE.

As the search changes directories, curpath is updated to reflect the new path. Curpath can be used by the calling program when the desired file is found. Errors reported by FmpNextMask are associated with curpath; they report errors in accessing the directory in curpath.

FmpNextMask tests the program's break flag (IFBRK) and if set, it returns error -235 (Break Detected). Thus, if your program also calls IFBRK, the break flag may have been cleared by FmpNextMask.

FmpEndMask should be called after a mask search terminates. If FmpEndMask is not called, directories may be left open to your program after the search ends.

FmpOpen

FmpOpen opens the named file with the specified options. Files must be opened before any operation that accesses their contents can be performed. Once opened, a file can be accessed until it is closed by FmpClose. When a file is opened, it is positioned to the first word in the file, at record number 1.

```
type = FmpOpen(dcb,error,filedescriptor,options,buffers)
integer dcb(*), error, buffers
character*(*) filedescriptor, options
```

type

A non-negative integer that returns the type of the opened file. If an error occurs, type returns a negative error code.

dcb

An integer array to contain the DCB for the file. The array must be at least 16 words long to contain file control information. For access to type 0 or 1 files, this minimum size is all that is required. For access to other type files, at least one DCB buffer of 128 words should also be allocated in DCB.

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

filedescriptor

A character string that specifies the name of the file.

options

A character string that selects options for opening the file. The options are selected by the letters in the following list:

```
Access mode:
    R   Open for reading
    W   Open for writing
File Existence:
    C   Create a new file
    O   Open an existing file
```

Miscellaneous:

D	File descriptor specifies a directory
F	Force type to 1 for nonbuffered access
Q	Open file quickly, do not record access time
S	Open a shared file
T	File is temporary
U	Open in update mode
X	Access extents in type 1 or 2 file
n	Use UDSP #n when searching for the file (n = 0,...,8)

The options can be specified in any order, and in uppercase or lowercase characters. Any combination of options is legal, but the options should be grouped by type for readability.

buffers

An integer between 1 and 127 that specifies the size of the DCB buffer, expressed as the number of 128-word buffers in the user array DCB, in addition to the 16-word file control information area. The larger the DCB buffer, the faster sequential file accesses can execute. The user array DCB must contain at least as many 128-word buffers as the parameter buffers indicates, or the file system may overwrite your program. The entire DCB buffer is used unless it is larger than the size of the accessed file or extent. Type 0 and 1 files (including files forced to type 1) do not use the DCB buffers, so the DCB need only have room for 16 words of file control information.

If the file being opened is on a FMGR cartridge, the file descriptor must be in the file::dir format.

FmpOpen updates the time of last access, unless the Q option is selected. FmpOpen sets the time of creation and time of last update for files that it creates.

The DCB specified in the call is closed before it is used for the file to be opened, even if it had last been used for the same file. Re-opening a file (to change the access options, for example) momentarily closes the file.

If the file descriptor specifies an LU number, FmpOpen assigns a DCB to the specified device. The device is referred to as a type 0 file, even though no real file exists on disc.

If the device is opened exclusively, the LU is locked unless the device is interactive. FmpOpen sets flags and option bits in the DCB according to the device type (that is, terminals are opened for read and write access, but line printers are open for write access only). The I/O options can be changed with the FmpSetIoOptions routine. An example of FmpOpen is as follows:

```
type = FmpOpen(dcb,error,'DATABASE.DB','rwso',8)
```

This call opens the existing file DATABASE.DB for shared read and write access, with a DCB buffer 1024 words (8 * 128) in length. The file must exist, because the create option is not selected. Your programs must coordinate shared write access.

Some examples of option combinations are:

To open an existing file for shared read access, specify ROS.

To create a new file for exclusive write access, specify WC. The O option can be specified at the same time as the C option for output files to create a new file if the specified file does not exist, or to overwrite an existing file. As a result, the C option should be used only for output files, not for sequential read files, because it can overwrite the file when it opens it. Note that because creating a file implies write access to the file, the W option always must be specified with the C option.

To create a temporary write/read scratch file, specify WRCT.

The calling program must have access privileges to all files that it tries to open. An error is generated if a program tries to access a file in a way that is not specified by the open request options, such as writing to a file that is opened only for reading. Changing the protection for a file after it is open to one or more programs has no effect on their access to the file.

C Option

The C option creates a file. The W option also must be specified because creating a file implies write access. If you do not specify the W option, error -203 (Did not ask to write) is returned.

FmpOpen can be used to create any type of file. The filedescriptor parameter must specify the filename, type, directory, and all other file information. To create a file of type 2, with 200 blocks of records that are 10 words in length, the following filedescriptor is used:

```
FILE.DAT::DIRECTORY:2:200:10
```

FmpBuildName or FmpBuildPath can be called to create a file descriptor from a file name and integer file information.

NOTE

If the O option is specified and the file already exists, all of the information after the directory is ignored, the existing file is opened and, for a variable length record file, the EOF mark is placed at the beginning of the file to make the file empty. The type of the existing file is unchanged; it is returned as a function value.

If only the file name and directory are specified, the file system will default to type 3, with a length of 24 blocks.

Files larger than 32767 (16383 blocks) sectors are created by specifying the size as a negative number of 128-block "chunks." A file of 128000 blocks is specified with a size of -1000. Positive numbers larger than 32767 are meaningless, but do not cause an error.

If a size of -1 is specified when creating an FMGR file, the rest of the space on the FMGR cartridge is used, up to a maximum of 16383 blocks.

D Option

The D option lets the filedescriptor parameter specify a directory rather than a file. It is used by programs that scan directories. Directories are usually read as type 2 files with 32-word records. Directories cannot be opened for write access.

F Option

The F option forces a file to type 1 for nonbuffered access, which ignores record marks. This option does not change the file type or extents of the file. The type parameter of FmpOpen returns the correct file type regardless of whether the F option is specified for the file.

Type 1 access is faster because a block of data is transferred directly from the disc to the user buffer (IBUF); the DCB buffer is bypassed. The calling program is responsible for calculating record length and accessing entire records.

An error occurs if you specify the F option for a device file.

Q Option

The Q option opens a file quickly, without recording the access time. This is useful when a file is opened repeatedly, which makes the access time unimportant. It is also used when the system time is not set.

S Option

The S option opens a file for shared access. By default, files are opened exclusively; no other program can access the file as long as it is opened exclusively to another program.

If a file is opened for reading only, it should be opened for shared access to let other programs read from the file at the same time.

No program can exclusively open a file that is already opened for shared access.

T Option

The T option creates temporary files. These files are flagged as temporary files in the directory, and should be purged by the calling program when no longer needed.

FMP automatically purges temporary files if a calling program creates and opens exclusively a temporary file, and terminates without closing the temporary file. The temporary file is purged the next time FMP scans its internal file table; for example, FMP scans its internal file table when a program accesses a file for the first time.

Temporary files that are closed by FmpClose are not automatically purged. You can make a temporary file permanent by opening the file without specifying the T option.

You can use the temporary flag to cleanup after a system failure by using the masking T option with the PU command (PU @.@.T).

The T option is ignored for FMGR files.

U Option

The U option reads the block containing the record to be updated into the DCB before the record is modified. This prevents existing records in the block from being destroyed.

Update mode is automatically in effect when a type 2 file is opened for write access. The U option must be specified in all other circumstances; for example, modifying a record in the middle of a sequential file.

Update mode is not related to the time of last update found in other FMP routines.

X Option

All file types can be extended to allocate additional disc space when the file becomes full. The X option is not required for sequential files, because they are automatically extended, but it is necessary for random access (type 1, 2 or 6) files, so that they can be extended when the last record of the existing file is filled. Some programs cannot automatically access extents for type 1 and 2 files; the X option lets them access the extents. Type 6 files are program files, so they should not be extended.

n Option

The number n specifies the number of the User-Definable Directory Search Path (UDSP) to be used in searching for the file. N can be set to a value from zero through 8, inclusive.

The n option is ignored if directory information is included in the file descriptor; FmpOpen seaches only the directory specified in the file descriptor.

If the file descriptor does not include directory information, FmpOpen searches each directory in the specified UDSP until the file is found. If the file is not found, a -6 (No such file) error is returned.

If the UDSP specified with the n option does not exist, a -247 (UDSP not defined) error is returned.

Refer to the PATH command in Chapter 5 of this manual for more information on UDSPs.

FmpOpenFiles

FmpOpenFiles finds open files in a directory.

```
error = FmpOpenFiles(dcb,error,loc,flag)
integer dcb(*), error, loc, flag
```

dcb

An integer array containing the DCB for the file.

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

loc

An integer that returns the directory position of an open file. The calling program initializes it to zero to indicate that this is the first call. Each time this routine is called, the location and flag value for one file are returned in the loc and flag parameters.

flag

An integer that returns the ID segment number of the program that opened the file (in bits 0-7) and the exclusive open bit (in bit 15).

The location is returned as a record number in a type 2 file (the directory). Loc = 1 is the first 32-word entry in the file, the directory header. The flag contains contains the ID segment number of the program that opened the file in bits 0-7, and the exclusive open bit in bit 15.

Locations are returned in ascending order. Only one flag is returned per file, so there is no way to tell how many programs are sharing an open file. When all of the open files in the directory have been reported, loc is returned as -1.

FmpOpenScratch

FmpOpenScratch is an interface to the FmpOpen routine. FmpOpenScratch standardizes the search path used in the creation of scratch files.

```
type = FmpOpenScratch(dcb,error,filedescriptor,options,buffers,
                      nameused)
       integer dcb(*), error, buffers
       character*(*) filedescriptor, options, nameused
```

type

A non-negative integer that returns the type of the opened file. If an error occurs, type returns a negative error code.

dcb

An integer array to contain the DCB for the file. The array must be at least 16 words long to contain file control information. For access to type 0 or 1 files, this minimum size is all that is required. For access to other type files, at least one DCB buffer of 128 words should also be allocated in DCB.

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

filedescriptor

A character string specifying the name of the file.

options

A character string that selects options for opening the file. Options are the same as the options for FmpOpen with the addition of the following option:

Z Use file name as prefix for FmpUniqueName

The options can be specified in any order, and in uppercase or lowercase characters. Any combination of options is legal, but the options should be grouped by type for readability.

buffers

An integer between 1 and 127 that specifies the size of the DCB buffer, expressed as the number of 128-word buffers in the user array DCB, in addition to the 16-word file control information area. The larger the DCB buffer, the faster sequential file accesses can execute. The user array DCB must contain at least as many 128-word buffers as the parameter buffers indicates, or the file system may overwrite your program. The entire DCB buffer is used unless it is larger than the size of the accessed file or extent. Type 0 and 1 files (including files forced to type 1) do not use the DCB buffers, so the DCB need only have room for 16 words of file control information.

nameused

A character string in which the name of the scratch file that was opened is returned.

If a directory is specified in the filedescriptor parameter, then FmpOpenScratch calls FmpOpen using that directory. If no directory is given, FmpOpenScratch calls FmpOpen one or more times using the standard sequence to find a scratch directory. FmpOpenScratch:

1. Tries the directory /SCRATCH/ first. If an error occurs (such as 'no such directory'), then it

2. Tries FMGR cartridge specified by entry point $SCRN. This entry point contains a FMGR disc LU defined at boot-up to be used as a scratch cartridge. The BOOTEX command, SC, sets the value of $SCRN. If any error occurs (such as 'cartridge full'), then it

3. Tries the default directory (' '). FmpOpen then uses either the calling programs working directory or, if there is no working directory, the first available FMGR cartridge.

With the exception of the Z option and the nameused parameter, the parameters for FmpOpenScratch are identical to FmpOpen parameters.

The Z option causes the routine to take the file name from the file descriptor given, and use it as a prefix to generate a unique name using the FmpUniqueName routine (refer to the description of this routine documented later in this chapter). For example, if the file descriptor is 'TEST:::4:5', with the Z option in the options parameter, FmpOpenScratch will call FmpUniqueName with the name 'TEST' as the prefix. The unique name that results will be used in the FmpOpen call.

FmpOpenScratch calls FmpFileName which builds the actual file descriptor. The file descriptor is returned in the nameused parameter. (For details refer to the description of FmpFileName.) Note that FmpOpenScratch uses this parameter to build the file descriptor that it uses in the FmpOpen call; therefore, the size of the variable passed should equal the size of the maximum file descriptor allowed (63 characters).

All parameters except nameused are passed by the FmpOpenScratch routine to FmpOpen. The FmpOpen routine returns any values directly to the routine calling FmpOpenScratch. The value of the FmpOpenScratch function is either the file type (if no error occurs), or the error (as returned by FmpOpen). This calling sequence is identical to the FmpOpen calling sequence. Therefore, you should be able to use this routine as a direct replacement for the FmpOpen call in situations where the scratch directory is used.

FmpOwner

FmpOwner returns the name of the owner of the specified directory.

```
error = FmpOwner(dir,owner)
character*(*) dir, owner
```

dir

A character string that specifies the name of the directory.

owner

A character string that returns the log-on name of the user who owns this directory.

FmpPackSize

FmpPackSize packs the double integer file size into a single word.

```
size = FmpPackSize(doublesize)
integer size
integer*4 doublesize
```

size

An integer that returns the file size in one word.

doublesize

A double integer specifying the file size.

If doublesize is less than 32767, there is no change. If doublesize is greater than 32767, it is rounded up to the nearest multiple of 128 and divided by 128, and the sign is changed. No overflow check is made. Refer to the FmpExpandSize routine for a description of special considerations for FMGR size parameters.

Because of overflow problems and rounding errors,

```
Size = FmpPackSize(FmpExpandSize(size))
```

is an identity for all values of size, but

```
Doublesize = FmpExpandSize(FmpPackSize(doublesize))
```

is not always an identity.

FmpParseName

FmpParseName parses the specified file descriptor into its component fields. It is similar to FmpParsePath.

```
call FmpParseName(filedescriptor,name,typex,sc,dir,type,size,rl,ds)
character*(*) filedescriptor, name, typex, dir, ds
integer sc, type, size, rl
```

filedescriptor

A 63-character string that specifies the file descriptor to be parsed.

name

A character string that returns the file name. Name can be up to 16 characters in length.

typex

A character string that returns the file type extension. Typex can be up to 4 characters in length.

sc

An integer that returns the security code.

dir

A character string that returns the directory name. Dir can be up to 16 characters in length.

type

An integer that returns the FMP file type.

size

An integer that returns file size in blocks.

rl

An integer that returns the record length.

ds

A character string that returns the DS node name, user account name, or both. Ds can be up to 63 characters in length. Refer to the DS File Access section in Chapter 3 of this manual for a description of the DS node name and user account name.

FmpParseName should be used to upgrade programs designed to manipulate FMGR files, or in new programs when the hierarchical and file masking features of FmpParsePath are not required. The differences between FmpParseName and FmpParsePath are described in the FmpParsePath section of this chapter.

FmpParseName converts the character string input fields of the filedescriptor parameter into integers when necessary, as for the type and size fields. When characters appear in numeric fields, they are returned as packed ASCII. For example, if the security code in the filedescriptor parameter is "DH," the returned sc parameter is 17480. Character fields are returned just as they appear in filedescriptor. Numeric fields omitted in the filedescriptor parameter are returned as zeroes; omitted character fields are returned as blanks. No error checking is made on filedescriptor or the returned parameters.

For example, assume that fdesc = SANJOSE.TXT::CITIES:4:24.

```
CALL FmpParseName(fdesc,file,ext,sc,dir,type,size,reclen,ds)
```

File = SANJOSE, ext = TXT, sc = 0, dir returns CITIES, type = 4, size = 24, reclen = 0, and DS is blank.

FmpParseName is not designed to parse file descriptors with hierarchical directory paths (that is the function of FmpParsePath), but it can parse them, with the following limitations.

When a leading directory and subdirectories are specified, the directory name is returned to dir, and the first 16 characters of the rest of the directory path and file name is returned in the name parameter. For example:

```
If fdesc = /CITIES/CALIFORNIA/SANJOSE.TXT:::4:24

   CALL FmpParseName(fdesc,name,ext,sc,dir,type,size,reclen,ds)
```

Name returns CALIFORNIA/SANJO, ext returns TXT, sc = 0, dir returns CITIES, type = 4, size = 24, reclen = 0, and ds = " "

FmpParsePath

FmpParsePath parses the specified file descriptor into its component fields. It is similar to FmpParseName, except that it parses hierarchical directory paths in a way that is more convenient for you to use programatically, and parses file descriptors that contain a mask qualifier field.

```
call FmpParsePath(filedescriptor,dirpath,name,typex,qual,
                  sc,type,size,rl,ds)
character*(*) filedescriptor, dirpath, name, typex, qual, ds
integer sc, type, size, rl
```

filedescriptor

A 63-character string that specifies the file descriptor to be parsed.

dirpath

A character string that returns the hierarchical directory path.

name

A character string that returns the file name. Name can be a maximum of 16 characters. Name does not return any part of the hierarchical directory information.

typex

A character string that returns the file type extension. Typex can be a maximum of 4 characters.

qual

A character string mask qualifier. Qual can be a maximum of 40 characters.

sc

An integer that returns the security code.

dir

A character string that returns the directory name. Dir can be a maximum of 16 characters.

type

An integer that returns the FMP file type.

size

An integer that returns file size in blocks.

rl

An integer that returns the record length.

ds

A character string that returns the DS node name, user account name, or both. Ds can be a maximum of 63 characters. Refer to the DS File Access section in Chapter 3 of this manual for a description of the DS node name and user account name.

FmpParsePath should be used when writing new programs that will use the hierarchical file system features, and must be used if file masking is required. Refer to the DL command description in Chapter 5 and to the FMP mask routines described in this chapter for more information about file masking.

The hierarchical directory path (returned in dirpath) is defined as everything that appears to the left of the first character of the file name. All of the directory information in the filedescriptor parameter is combined and returned in dirpath. If filedescriptor uses the trailing directory notation, as in FILE::GLB, FmpParsePath converts filedescriptor to the leading (hierarchical) notation, as in /GLB/FILE, and returns the directory path in dirpath.

Qual permits FmpParsePath to correctly parse file descriptors that contain masks. Mask qualifiers are described in the DL command description in Chapter 5.

FmpParsePath differs from FmpParseName in two main ways:

- FmpParsePath parses file descriptors with file masks as well as regular file names, and includes the qual parameter to return the mask qualifier field.

- FmpParsePath parses hierarchical directory path information in a way that is more convenient for you to use programatically. All of the directory information in the filedescriptor parameter is returned in dirpath, never in the name parameter as with FmpParseName.

The following examples illustrate these differences:

Input	FmpParsePath Output			FmpParseName Output		
Filedescriptor	dirpath	name	typex	dir	name	typex
/GLB/SUB/FILE.FTN	/GLB/SUB/	FILE	.FTN	GLB	SUB/FILE	.FTN
SUB/FILE.FTN::GLB	/GLB/SUB/	FILE	.FTN	GLB	SUB/FILE	.FTN
/GLB/SUB.DIR	/GLB/	SUB	.DIR	GLB	SUB	.DIR
/GLB.DIR	/	GLB	.DIR	GLB	blank	blank
/GLB/	/GLB/	blank	blank	GLB	blank	blank
::GLB	/GLB/	blank	blank	GLB	blank	blank
S1/S2/FILE.REL	S1/S2/	FILE	.REL	blank	S1/S2/FILE	.REL
FILE.REL	blank	FILE	.REL	blank	FILE	.REL

The following is an example of how FmpParsePath parses a full file descriptor:

```
Filedesc = CALIFORNIA/SANJOSE.TXT.T:23:CITIES:2:24:32[PLANNER]>SYS3

  CALL FmpParsePath(filedesc,path,name,extn,qual,sc,type,size,rl,ds)
```

Path = /CITIES/CALIFORNIA/, name = SANJOSE, extn = TXT, qual = T, sc = 23, type = 2, size = 24, rl = 32, and ds = [PLANNER]>SYS3.

FmpPosition

FmpPosition returns the current record number and reports the internal file position in a format that can be used later by FmpSetPosition.

```
error = FmpPosition(dcb,error,record,position)
integer dcb(*), error
integer*4 record, position
```

record

A double integer that returns the current record number.

dcb

An integer array containing the DCB for the file.

error

An integer that returns a negative code if an errror occurs or zero if no error occurs.

position

A double integer that returns the current internal file position.

Refer to the FmpSetPosition section of this chapter for a description of how the current record and internal file position are used to change the file position.

Each record in a file is numbered. The first is number one, and the others are numbered consecutively. As the file is read or as information is written to it, the current position is incremented. It is also changed by the FmpSetPosition and FmpRewind routines.

The current record position does not identify an exact byte location in variable record length files. The internal file position specifies the current word offset from the first word of the file. The first word of a file is position zero. The internal position does not depend on actual disc location of the file, so positions can be used even after a file is moved or copied. This value is meaningless for device files.

FmpPost

FmpPost posts the data in the DCB buffer into the disc file if the data has been changed. Other programs can then access the information by reading the disc file. FmpPost is also used to back up the DCB buffer into the disc file in case the program is aborted. When the DCB buffer is posted, the data in the buffer is invalidated, so the next read call reads the disc file, not the DCB buffer.

```
error = FmpPost(dcb,error)
integer dcb(*), error
```

dcb

An integer array containing the DCB for the file.

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

FmpPost is used to coordinate shared write access to a file. Resource numbers are often used with FmpPost to coordinate the sharing of write access. Refer to the RNRQ section of the Resource Management chapter for more information about resource numbering. Each of a group of cooperating programs that accesses the shared file should perform the following sequence:

1. Lock the file's resource number
2. Access the file
3. Call FmpPost to post the data in the disc file
4. Unlock the resource number

FmpProtection

FmpProtection returns the access rights of the owner and others to the specified file or directory.

```
error = FmpProtection(filedescriptor,owneraccess,othersaccess)
character*(*) filedescriptor, owneraccess, othersaccess
```

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

filedescriptor

A character string specifying the name of the file.

owneraccess

A character string that returns the access rights of the owner of the file or directory.

othersaccess

A character string that returns the access rights of all other users of the file or directory.

The access rights are returned as ASCII "R" for read access, "W" for write access, or "RW" for both.

The owner of a directory is the user who creates it or is assigned ownership via the FmpSetOwner routine. The owner of a directory owns all of the files within it.

FmpPurge

FmpPurge purges the file specified by the file descriptor, marking the directory entry as purged, to free the disc space allocated to the file. The file must exist, must not be open, and must not be an RP'd program. The calling program must have write access to the directory, but not necessarily write access to the file.

```
error = FmpPurge(filedescriptor)
character*(*) filedescriptor
```

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

filedescriptor

A character string specifying the name of the file.

The file descriptor can specify a directory by specifying it as ::NAME or /SUB.DIR (note the .DIR file type extension). If the directory contains anything other than purged files, it cannot be purged.

Purged files can be unpurged with the FmpUnPurge routine, unless their disc space or directory entry is overwritten.

FmpRead

FmpRead reads data from a file of any type. FmpRead reads the record at the current file position. The file positioning routines described in this chapter explain how to change the current file position. The file must be opened for read access before FmpRead is called.

```
length = FmpRead(dcb,error,buffer,maxlength)
integer dcb(*), error, buffer(*), maxlength
```

length

An integer that returns the number of bytes actually read, or a negative error code. If the call reads more than 32767 bytes, the return length may be negative even though no error occurs; in such cases the error variable should be compared to the length return. If they match, an error has probably occurred.

dcb

An integer array containing the DCB for the file.

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

buffer

An integer array that returns the data being transferred. The buffer is word-aligned.

maxlength

A one-word integer that contains the maximum number of bytes to transfer. Maxlength is treated as an unsigned single integer from 0 to 65534. Values larger than 32767 are expressed as negative numbers equal to the number of bytes to be transferred minus 65536; for example, 40000 bytes is expressed as -25534 (40000 - 65536 = -25536).

If an odd number of bytes are transferred, the lower byte of the word containing the last byte is undefined. The requested transfer length can be longer or shorter than the actual length of the record, but the number of bytes read never exceeds the maxlength.

The file position is set to the beginning of the next record even if some of the data that was read does not fit into the user buffer.

For sequential files (type 3 and above), one variable-length record is transferred from the current file position. The DCB buffer is used during the transfer. The record length is maintained with the record; if for some reason the record-length information is invalid, error -5 is returned.

For type 2 files, one fixed-length record is transferred, using the file record length, which is always an even number of bytes. The DCB buffer is used during the transfer. There is no end-of-file mark; if a program tries to read past the end-of-file, the actual length of the record is returned, and no error is indicated, but subsequent reads will report an error.

For type 1 files (or files forced to type 1), multiple records may be read, depending on maxlength. The data is read directly into the user buffer, without using the DCB buffer. Type 1 files are always positioned at a block boundary, so they behave like files with 128-word records. Type 1 files behave like type 2 files when the end-of-file mark is encountered.

For type zero (device) files, one record is read. The data is read directly into the user buffer, without using the DCB buffer. End-of-file is set if the end-of-file or end-of-medium bits are set in the returned status following the read. The returned length is -1. The control-D character is the end-of-file mark for reads from a terminal; zero-length reads are not treated as the end-of-file. No more than 32767 bytes can be read from type 0 (device) files.

FmpReadString

FmpReadString is an integer function that allows reading character from a file.

```
length = FmpReadString(dcb,error,string)
integer length, dcb(*), error
character*(*) string
```

length

An integer that returns the positive number of bytes transferred, or a negative error code. Length cannot be more than 256 because the data must pass through an internal buffer that is 256 bytes.

dcb

An integer array containing the DCB for the file.

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

string

A character string of up to 256 bytes into which data is transferred. The string cannot be more than 256 bytes because the data passes through an internal buffer that is 256 bytes. If string is longer than 256 bytes, an error code is returned in the error parameter.

FmpReadString is similar to FmpRead, except the data is returned in the string parameter. The returned length is the length of the record read; it may be less than the actual length of the string parameter, but never more. The string is filled with blanks if the record is shorted than the string.

FmpRecordCount

FmpRecordCount returns the number of records in the specified file.

```
error = FmpRecordCount(filedescriptor,nrecords)
character*(*) filedescriptor
integer*4 nrecords
```

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

filedescriptor

A character string specifying the name of the file.

nrecords

A double integer that returns the number of records in the file.

For type 1 and 2 files FmpRecordCount returns the maximum number of records that can fit in the file, not the actual number of records currently in the file. For type 3 files and above, nrecords is the number of records before the end-of-file; however, if the file is currently open for writing, nrecords may not reflect the actual record count because write requests that have not been posted may not be present in the file.

FmpRecordLen

FmpRecordLen returns the length of the longest record in a file.

```
error = FmpRecordLen(filedescriptor,len)
character*(*) filedescriptor
integer len
```

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

filedescriptor

A character string specifying the name of the file.

len

An integer that returns the length of the longest record in the file.

For a type 1 or 2 file, FmpRecordLen returns the fixed record length in words, that was defined when the file was created. For type 3 files and above, it returns the length, in words, of the longest of the variable length records in the file.

NOTE

The length returned for type 3 or higher files is actually the length of the longest record ever written to the file, even if that longest record has been overwritten.

FmpRename

FmpRename changes the name of the specified file.

```
error = FmpRename(name1,err1,name2,err2)
integer err1, err2
character*(*) name1, name2
```

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

name1

A character string specifying the name of the existing file. The file must be closed.

err1

An integer that returns any error associated with name1.

name2

A character string specifying the new name for the file.

err2

An integer that returns any error associated with name2.

The file specified by name1 must exist, and must not be open. It may, however, be an active program. Name2 must not already exist in the directory.

The calling program must have write access to the directory containing the file to be renamed, and to the directory that will contain the file after the rename, if it is not the same as the original directory.

FmpRename can change any combination of the file name, its file type extension, or directory. The security code, type, size, and recordlength cannot be changed. If they are specified in name2, they are ignored. The new file name (name2) must specify the desired security code and directory; they cannot be defaulted to match the security and directory of name1.

If the directory name is changed, the file directory entry is moved to the new directory, but the actual file data is not moved. The new directory must be on the same LU as the original. Name1 and name2 can specify directories as either ::NAME or /NAME.DIR (note the .DIR file type extension). It is possible to convert subdirectories into global directories, or vice versa. If the working directory is renamed, it remains the working directory, but under the new name. Err1 returns errors associated with name1 and err2 returns errors associated with name2. If either err1 or err2 contains an error code, the same error code is returned in error. If error = 0, then neither err1 nor err2 contains an error code.

FmpReportError

FmpReportError prints an error message at your terminal (LU 1).

```
call FmpReportError(error,filedescriptor)
character*(*) filedescriptor
integer error
```

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

filedescriptor

A character string that specifies the name of the file.

The printed message consists of the message returned by FmpError, followed by the passed filename; for example:

```
No such file FILE.EXT::USER
```

If it is necessary to print the message somewhere other than on LU 1, you should use FmpError to retrieve the error text and write the message to the desired file or device.

FmpRewind

FmpRewind positions the file specified by the DCB to the first word in the file. For disc files this is equivalent to an FmpSetPosition call with position set to zero. For device files, a rewind control call is issued.

```
error = FmpRewind(dcb,error)
integer dcb(*), error
```

dcb

An integer array containing the DCB for the file.

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

FmpRpProgram

FmpRpProgram tries to restore a program from a type 6 file, creating an ID segment for the program in the operating system.

```
error = FmpRpProgram(filedescriptor,rpname,options,error)
character*(*) filedescriptor, rpname, options
integer error
```

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

filedescriptor

A character string that specifies the name of the type 6 file.

rpname

A character string that either specifies the program name or returns it: if rpname is specified, the specified name is used; if rpname is blank, the name assigned by the system is returned. The returned name is the first five characters of filedescriptor (minus the directory path and file type extension). Note that a null (integer value of zero) cannot be used; nulls must be initialized to blanks, refer to the Character String section of this chapter for details.

options

A character string that contains "C", "P", or both, to select either of the following options:

C (clone) Create a clone name if the specified or assigned name already is assigned to an RP'd program. The program is not cloned if:

- There is a system program with the assigned or specified name.
- There is already a program with the assigned or specified name, but it is not RP'd.
- There is no program with that name currently RP'd.

P (permanent) Do not release the ID segment when the program completes.

If the program already existed, and cloning could not occur, error -239 is returned.

If FmpRpProgram needs to clone, it will replace the fourth and fifth characters of the program name with ".A". If that name is also taken, it will use ".B", and so forth.

If the RPL checksum on the type 6 file does not match the system, the file's checksum is changed, and the program is RP'd, but FmpRpProgram returns error -240 (RPL checksum changed). This error is a warning, and can be displayed to you or ignored.

FmpRpProgram is used by FmpRunProgram, CI, and most other program scheduling requests to search for an existing program with the specified or assigned name. FmpRpProgram searches for the program to be RP'd as follows:

1. If a directory is specified, this directory is searched for the file. If the file is found, it is RP'd. If the file is not found and a file type extension was not specified, .RUN is assumed and the directory is searched again. If the file still is not found, an error is returned.

2. If no directory information is given, the following occurs:

   a. If a program with the specified or assigned name is already RP'd and is dormant, this program is used. If the program is busy and cannot be cloned, an error is returned.

   b. If the program has not been RP'd or is busy but can be cloned, a search is made for the program (type 6) file. If User-Definable Directory Search Path (UDSP) number one is defined, a default file type extension of .RUN is assumed and the search path defined by UDSP #1 is used to find the file. If the file is not found, an error is returned.

   c. If UDSP #1 is not defined, the following default search sequence is used:

      - The current working directory is searched. If the file is not found, a default file type extension of .RUN is assumed and the working directory is searched again.

      - If you do not have a working directory, all FMGR cartridges are searched.

      - If the file is still not found, global directory PROGRAMS is searched, using the .RUN default file type extension. If the file is not found, an error is returned.

UDSP #1 can be defined using the CI PATH command. Refer to PATH command description in Chapter 5 for more information.

If a working directory exists, programs on a FMGR cartridge cannot be run unless the directory is specified by PROG::0 or PROG::crn.

FmpRunProgram

FmpRunProgram executes a program.

```
error = FmpRunProgram(string,prams,runname[,alterstring])
character*(*) string, runname
integer error, prams(5)
logical alterstring
```

error

An integer that returns a negative code if no error occurs or zero if an error occurs.

string

A character string that specifies a runstring. If the string does not begin with RU or XQ, FmpRunProgram inserts RU so the program can correctly parse the runstring. If XQ is specified, the program is executed without wait.

prams

An integer array that returns the RMPAR parameters from the program when it completes. If string specifies XQ, the prams are meaningless.

runname

A character string that returns the true name used to schedule the program.

alterstring

Optional boolean variable indicating how FmpRunProgram is to handle the string parameter. The possible values are as follows:

TRUE The string is converted to uppercase and each group of one or more consecutive blanks is converted to a comma. (default)

FALSE The string is not altered.

If a program with the same name and session ID already exists then an attempt will be made to create a clone name by replacing the last two characters with ".A". If that fails ".B" will be tried and so on. It is usually not necessary to clone a program, because programs are identified by their name plus their session number. If :IH follows the program name (for example, RU,PROG:IH), cloning is inhibited.

The order of search for the program is the same as for FmpRpProgram.

FmpRwBits

FmpRwBits is an integer function that determines whether the returned string of the FmpProtection routine indicates read or write access availability, and whether an options list for FmpOpen contains read or write access requests.

```
rwbits= FmpRwBits(string)
character*(*) string
```

rwbits

An integer that indicates read or write access availability for the string returned by FmpProtection, and read or write access requests for the options list of FmpOpen. FmpRwBits returns one of four values, depending upon whether or not the string parameter contains the uppercase letters R or W. The values for rwbits are as follows:

0	Neither W nor R present
1	W but not R present
2	R but not W present
3	R and W present

string

A character string. String can be a maximum of 256 bytes.

In the string parameter, the R and W can be in any order and other characters can be present.

FmpSetDcbInfo

FmpSetDcbInfo changes information in the DCB.

```
error = FmpSetDcbInfo(dcb,error,records,eofpos,reclen)
integer dcb(*), error, reclen
integer*4 records, eofpos
```

dcb

An integer array containing the DCB for the file.

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

records

A double integer that specifies the number of records in the file plus 1.

eofpos

A double integer that specifies the current internal file position.

reclen

An integer that specifies the length, in words, of the longest record.

FmpSetDcbInfo should be called only when a file of type 3 or above that has been forced to type 1 in the FmpOpen call is copied. The DCB for the copied file contains information for a type 1, rather than a type 3 file. FmpSetDcbInfo can be used to change the DCB information to reflect the fact that the file is really of type 3 or above. The call should be used with care, and only by users with a detailed knowledge of DCB information.

The records and eofpos parameters correspond to the current record and internal file position parameters of the FmpSetPosition routine.

Do not read or write any more data from the DCB after using this routine; call FmpClose to close the DCB, then FmpOpen to re-open it for further access.

FmpSetDirInfo

FmpSetDirInfo changes file directory information.

```
error = FmpSetDirInfo(dcb,error,ctime,atime,utime,bbit,prot)
integer dcb(*), err, bbit, prot
integer*4 ctime, atime, utime
```

dcb

An integer array containing the DCB for the file.

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

ctime

A double integer specifying the create time.

atime

A double integer specifying the access time.

utime

A double integer specifying the update time.

bbit

An integer specifying the backup bit.

prot

An integer specifying the new protection for the file.

The calling program can change the create, access, and update time stamps, set or reset the backup bit, and change the file protection.

If a supplied parameter is negative, the corresponding value in the directory entry is not changed.

If the calling program owns the file, it also can set the file protection to the lower 4 bits of prot. Prot is ignored if the calling program is not the owner.

Do not read or write any more data from the DCB after using this routine.

FmpSetDirInfo should be called after FmpSetDcbInfo if both are to be called.

FmpSetEof

FmpSetEof sets the end-of-file to the current position in a sequential file, or issues an end-of-file control request for a device file. It has no effect on type 1 and 2 files.

```
error = FmpSetEof(dcb,error)
integer dcb(*), error
```

dcb

An integer array containing the DCB for the file.

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

FmpSetEof is not required in normal operation because the end-of-file is set automatically following writes to sequential files that are not opened in the update mode. It should be used only to reset the end-of-file mark in files opened in the update mode, and for writing to device files which require an explicit end-of-file control request, such as magnetic tapes. It does not remove any other EOF marks in the file, so it cannot be used to expand a file; it can be used only to make the file smaller.

FmpSetIoOptions

FmpSetIoOptions changes the I/O option word for the specified DCB.

```
error = FmpSetIoOptions(dcb,error,options)
integer dcb(*), error, options
```

dcb

An integer array containing the DCB for the file.

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

options

An integer that returns the 16-bit I/O options word.

Once changed, the new options remain in effect until another FmpSetIoOptions call (or an FmpOpen call). The options word is described in the Standard I/O chapter of this manual. All of the options except the Z-bit can be set, because the FmpSetIoOptions call does not permit a Z buffer to be sent.

The call is ignored if the DCB is not open to a device file. FmpSetIoOptions should not be called under normal operation; in most cases, you should allow the file system to set the I/O option word.

FmpSetOwner

FmpSetOwner changes the owner of a directory to the specified user. You must be the current owner or a superuser.

```
error = FmpSetOwner(dir,err1,owner,err2)
character*(*) dir, owner
integer err1, err2
```

dir

A character string that specifies the name of the directory whose owner is being changed.

err1

An integer that returns errors associated with dir.

owner

A character string that specifies the name of new owner of the directory.

err2

An integer that returns errors associated with owner.

If either err1 or err2 contains an error code, the same code is returned in error. If error = 0, then neither err1 nor err2 contains an error code.

FmpSetPosition

FmpSetPosition sets or changes the current file position. The position can be set either to a record number or to an internal file position.

```
error = FmpSetPosition(dcb,error,record,position)
integer dcb(*), error
integer*4 record, position
```

dcb

An integer array containing the DCB for the file.

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

record

A double integer that specifies the desired record number.

position

A double integer that specifies the desired internal file position.

All files can be positioned to a particular record number. All disc files can be positioned to an internal file position as returned by an FmpPosition call. For fixed record length files, the record number and internal file positions are related by the function ((record_number-1) * record_size). For sequential files there is no such correlation because the records are variable in length.

Positioning sequential and device files by record number is very slow because it requires starting at the first record and stepping through to the desired record. Positioning by internal position is much faster for sequential files, but the position must be at the start of a record because read and write calls depend upon being at the beginning of a record. FmpPosition can be called to return the position of the start of a record to pass it to FmpSetPosition.

If the position parameter is positive, FmpSetPosition interprets it as the desired internal file position. The passed record number is saved as the current record number for later use.

If the position parameter is negative, positioning occurs by record. Device files are always positioned by record number only, regardless of the internal position value. Double integer variables should be used for the record number and internal position for device files, because they are often large numbers.

Although FmpSetPosition is usually called to position a file to a location already in the file, it can be used to create extents in a file opened for writing. Positioning a type 1 or 2 file can create an extent, but it can create a sparse file, which has missing extents between the file and a full extent. If a read request tries to access a record in one of the missing extents, an error occurs. Positioning a file of type 3 or above creates an extent without skipping extents, even if the file is forced to type 1 by the F option in the FmpOpen call.

FmpSetProtection

FmpSetProctection allows the owner of a file or directory to change the access rights to the file or directory.

```
error = FmpSetProtection(filedescriptor,owneraccess,othersaccess)
character*(*) filedescriptor, owneraccess, othersaccess
```

filedescriptor

A character string specifying the name of the file.

owneraccess

A character string specifying the access rights of the owner of the file or directory.

othersaccess

A character string specifying the access rights of other users of the file or directory.

The access rights are specified as ASCII "R" for read access, "W" for write access, or "RW" for both. The suggested setting is "RW" for owner, "R" for others.

When the access rights to a directory are changed, the access rights to files or subdirectories already in it are not changed, but new files or subdirectories created in it receive the new access rights.

The owner of a directory is the user who creates it or is assigned ownership via the FmpSetOwner routine. The owner of a directory owns all the files in it.

To prevent owners from being locked out of their own directories, owners do not need write access to a directory to change its protection. A superuser can change protection on any file or directory. A file's protection status can be changed while it is open, because protection status is only checked when the file is opened. Files that already have the file open are not affected by the protection change.

FmpSetWord

FmpSetWord positions a disc file to a specified internal position in the file.

```
error = FmpSetWord(dcb,error,position,how)
integer dcb(*), error, how
integer*4 position
```

dcb

An integer array containing the DCB for the file.

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

position

An integer specifying the desired file position. A positive value indicates an internal file position. A negative value indicates a record number.

how

An integer that specifies whether the file system should create an extent to contain the new position if it is outside the existing file area. How can be set to the following values:

1 Extent creation is not permitted; the usual setting for read operations which must only access existing file areas.

2 Extent creation is permitted.

FmpSetWord is a special case of the FmpSetPosition routine, and should be used only to minimize code size. FmpSetPosition is the general purpose positioning routine, and uses more code space.

FmpSetWord works exactly as FmpSetPosition does when it is called to position a file by internal file position, rather than by record. FmpSetWord does not update the record number in the DCB, so once it has been called, positioning by records must not be attempted. It also does not record the end-of-file position when a position beyond the existing end-of-file is selected without extent creation enabled. Its only advantage is that it does not add to the code size of the calling program, because it is used by FmpRead and FmpWrite, so it is already part of the code.

FmpSetWorkingDir

FmpSetWorkingDir changes or sets the working directory for you. The working directory can be a global directory or a subdirectory. Setting the working directory changes the working directory for all programs in the current session. It should be used with caution.

```
error = FmpSetWorkingDir(directory)
character*(*) directory
integer error
```

directory

An integer that returns a negative code if an error occurs or zero if no error occurs.

name

A character string that specifies the working directory.

If the name is specified as a zero, then you have no working directory until another call is made to establish one. This is useful in changing the search behavior for files when no directory is specified: if there is no working directory, the FMP calls can search FMGR discs for a specified file.

If name is longer than 63 characters, error -15 is returned.

FmpShortName

FmpShortName returns the file descriptor for the file associated with the specified DCB.

```
error = FmpShortName(dcb,error,filedescriptor)
character*(*) filedescriptor
integer dcb(*), error
```

dcb

An integer array containing the DCB for the file.

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

filedescriptor

A character string that returns the name of the file.

The returned file descriptor is not a full file descriptor; it does not include the file type, size, or record length. FmpShortName is similar to FmpFileName, described in this chapter, except that it returns a truncated file descriptor.

FmpSize

FmpSize returns the physical size of the file in blocks.

```
error = FmpSize(filedescriptor,size)
character*(*) filedescriptor
integer*4 size
```

filedescriptor

A character string specifying the name of the file.

size

A double integer that returns the physical size of the file in blocks.

The physical size of a file is the number of blocks of disc space it occupies, including extents.

FmpStandardName

FmpStandardName converts a file descriptor to the standard format.

```
call FmpStandardName(filedescriptor)
character*(*) filedescriptor
```

filedescriptor

A character string that specifies the name of the file.

The standard format uses the trailing directory notation, as in FILE.FTN::DIR. If the specified file descriptor includes subdirectories, it uses the hierarchical format, with a leading directory path, as in /DIR/SUB/FILE.FTN. If the file descriptor refers to a global directory, it also uses the hierarchical format, as in /GLB.DIR.

The standard is convenient for users familiar with FMGR files because the "::" notation is used whenever the file descriptor does not include a hierarchical directory structure.

FmpTruncate

FmpTruncate releases some of the disc space allocated to a file. The file must be opened for writing.

```
error = FmpTruncate(dcb,error,blocks)
integer dcb(*), error
integer*4 blocks
```

dcb

An integer array containing the DCB for the file.

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

blocks

A double integer specifying the minimum number of blocks to which file is to be truncated.

The file specified by DCB is truncated to no less than the specified double integer number of blocks. More blocks than this may actually remain, depending on internal considerations. Files will never be truncated to less than one block. It is the responsibility of the calling program to make sure that valid data is not truncated. The EOF mark should be in the area that remains after truncation. You should close file after it is truncated.

For example, if after performing sequential writes to a variable length record file (type 3 and above) you want to truncate the space beyond the current EOF mark, you can use the following (assuming the file is positioned at EOF mark):

```
Call FmpPosition (dcb,error,record,position)
if (error.lt.0) ...
blocks = (position + 128)/128
Call FmpTruncate (dcb,error,blocks)
if (error.lt.0) ...
Call FmpClose (dcb,error)
```

The calculation "position + 128" includes one word for the EOF mark, and rounds up the position so that all words in the current block are included. Dividing by 128 converts the number of words to number of blocks.

FmpUdspEntry

FmpUdspEntry returns the directory name for the specified entry and User-Definable Directory Search Path (UDSP).

```
error = FmpUdspEntry(udspnum,entnum,dirname,error)
integer udspnum, entnum, error
character*(*) dirname
```

udspnum

An integer that specifies the UDSP number.

entnum

An integer that specifies the entry for the UDSP number.

dirname

A character string that returns the directory name for the specified entry in the specified UDSP.

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

If the entry is undefined, or if udspnum and entnum are out of bounds with the definition for the session, error -247 is returned.

FmpUdspInfo

FmpUdspInfo returns the current User-Definable Directory Search Path (UDSP) information for your session.

```
error = FmpUdspInfo(udsps,depth,next,error)
integer error, udsps, depth, next
```

udsps

An integer that returns the number of UDSPs defined for the current session.

depth

An integer that returns the UDSP depth defined for the current session.

next

An integer that returns the next available UDSP. Next is set to zero if all UDSPs are defined.

error

An integer that returns one of the following values:

0	No error occurred
-1	Not under session control
-2	UDSP tables not set up correctly

FmpUniqueName

FmpUniqueName creates a 16-character file name that should be unique within a system that does not contain files from another system.

```
call FmpUniqueName(prefix,uniquename)
character*(*) prefix, uniquename
```

prefix

A character string specifying a prefix for the file name.

uniquename

A character string that returns the generated file name.

The name is created by appending a reading from the system clock to a user-supplied prefix. The clock reading is expressed as a string of hex digits. A typical uniquename is "TEMP7C43E20FF21".

If the file may be transferred to an FMGR directory, the prefix should be chosen to minimize the chance of a duplicate filename when the uniquename is truncated to six characters.

FmpUnPurge

FmpUnPurge restores a purged file. The file must have existed and been purged, and its disc space must not have have been allocated to another file.

```
error = FmpUnPurge(filedescriptor)
character*(*) filedescriptor
```

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

filedescriptor

A character string specifying the name of the file to be unpurged.

FmpUnPurge verifies the directory entry for the file and any extents, and makes sure that none of its disc space has been allocated to any other file. If it passes both tests, FmpUnPurge reallocates all of its space and converts its directory entries back to the normal status. The file's protection, time stamps, and other attributes are restored exactly as they were at the time that the file was purged.

Directories cannot be unpurged.

If several purged files have the same name, it is difficult to determine which is to be unpurged. The result of an FmpUnPurge call is not defined.

Files cannot be unpurged if a file already exists with the same name; the existing file must be renamed first.

FmpUpdateTime

FmpUpdateTime returns the time of the last update for the named file. The file is not opened in the process.

```
error = FmpUpdateTime(filedescriptor,time)
character*(*) filedescriptor
integer*4 time
```

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

filedescriptor

A character string specifying the name of the file.

time

A double integer that returns the time of the last update expressed in seconds since Jan 1, 1970.

The update time is set when a file is closed, but only if the file was changed while it was open.

Routines are available to convert the time value to an ASCII string. Usually, however, the returned time is compared to times in the same format, so the calling program may not have to convert the format.

FmpWorkingDir

FmpWorkingDir returns the name of your current working directory. The current working directory can be either a global directory or a subdirectory.

```
error = FmpWorkingDir(directory)
character*(*) directory
integer error
```

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

directory

A character string that returns the name of the current working directory.

The returned name is in a format suitable for passing to other routines, such as. FmpSetWorkingDir.

If the name contains more than 63 characters, the name is truncated to 63 characters and an error is returned.

If there is no working directory, then an error is returned and the name is undefined.

FmpWrite

FmpWrite writes data to a file of any type. The file must be opened for write access.

```
length = FmpWrite(dcb,error,buffer,maxlength)
integer length, dcb(*), error, buffer(*), maxlength
```

length

An integer that returns the number of bytes actually transferred, or a negative error code. If more than 32767 bytes are transferred, the returned length is a negative number. If this negative number is equal to the value of the error parameter, an error has probably occurred.

dcb

An integer array contianing the DCB of the file.

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

buffer

The name of a word-aligned buffer that contains the data to be transferred.

maxlength

The maximum number of bytes to write; it is interpreted as an unsigned one-word integer from 0 to 65534. For values larger than 32767, set maxlength to the desired maximum number of bytes minus 65536; for example, 40000 bytes is expressed as -25534 (40000 - 65536 = -25536).

FmpWrite writes data at the current position of the file. The file position can be set by other FMP routines, such as FmpSetPosition and FmpAppend.

For sequential (type 3 or above) files, one record is written. The DCB buffer is used during the transfer. If the file is not opened in update mode, the entire record is transferred and an end-of-file mark is written after it. If the file is opened in update mode, then the length transferred will be the shorter of the existing and supplied record lengths. No end-of-file mark is written.

For type 2 files, one record is written, using the shorter of the defined and supplied record lengths. The DCB buffer is used for the transfer.

For type 1 files (and files forced to type 1), multiple records may be written, depending on the supplied record length. The data is transferred directly from the user buffer to the disc. The return length is rounded up to an even number if necessary.

For type zero (device) files, one record is transferred. The data is written directly from the user buffer to the device. No more than 32767 bytes can be transferred with one call.

FmpWriteString

FmpWriteString is similar to FmpWrite, except that the data to be transferred is supplied in the string parameter.

```
length = FmpWriteString(dcb,error,string)
integer length, dcb(*), error
character*(*) string
```

length

An integer that returns the length of the record written to the file, or a negative error code. It may be less than the actual string length, but never longer.

dcb

An integer array containing the DCB for the file.

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

string

A character string of up to 256 bytes from which data is transferred. String cannot be greater than 256 bytes because the data must pass through an internal buffer of 256 bytes. If string is longer than this limit, an error is returned.

MaskMatchLevel

MaskMatchLevel is an integer function that returns the number of the directory level in which the last file was matched.

```
matchlevel = MaskMatchLevel(dirdcb)
integer matchlevel, dirdcb(*)
```

matchlevel

An integer set to the number of the directory level containing the last file that was matched.

dircb

An integer arrary initialized by FmpInitMask.

For example, if the search mask is /GLOBAL.DIR.D and matched file is /GLOBAL/SUBDIR/FILE, then matchlevel returns 2, to indicate that the file is nested two levels below the global directory. This value can help in creating new names for copy or rename operations, although Calc_Dest_Name is more commonly used for that function.

MaskOldFile

MaskOldFile is a boolean function that checks if the last file returned by FmpNextMask is a FMGR file.

```
bool = MaskOldFile(dirdcb)
integer dirdcb(*)
boolean bool
```

bool

A boolean variable that is set to TRUE if the last file returned by FmpNextMask is a FMGR file; otherwise, bool is set to FALSE.

dircb

An integer arrary initialized by FmpInitMask.

MaskOpenId

MaskOpenId is an integer function that returns the D.RTR open flag of the last file returned by FmpNextMask.

```
openid = MaskOpenId(dirdcb)
integer openid, dirdcb(*)
```

openid

An integer that returns the ID number of the file that has the file open. If the file is not open, openid is set to zero. If the file is open, the ID number of a program that has the file open is returned in bits 0-7, and the value of the exclusive bit is returned in bit 15.

dircb

An integer array initialized by FmpInitMask.

The returned program may not be the only program that has the file open. Refer to the FmpOpenFiles routine description for more information on the format of the open flag.

MaskSecurity

MaskSecurity is an integer function that returns the security code of the last file returned by FmpNextMask, if the file is an FMGR file. For FMP files, it returns zero.

```
seccode = MaskSecurity(dirdcb)
integer seccode, dirdcb(*)
```

seccode

An integer that returns the security code of the last file returned by FmpNextMask, if the file is a FMGR file. For FMP files, seccode is set to zero.

dirdcb

An integer array initialized by FmpInitMask.

WildCardMask

WildCardMask checks the mask for wildcard characters.

```
wild = WildCardMask(mask)
boolean wild
character*(*) mask
```

mask

A character string that contains the mask to be checked.

wild

A boolean indicating the presence of a wildcard character. Wild returns one of the following values:

TRUE The mask contains a wildcard character ("@" or "-"), or the mask qualifier contains any of the search directives ("d", "e", or "s"), or the specified mask can refer to more than one file for another reason.

FALSE The mask cannot refer to more than one file.

If WildCardMask returns FALSE, there is no need to use the mask search routines to find a specific file; it is faster to use the specified mask to open and access the file directly.

Using the FMP Routines with DS

All of the FMP calls that use a filedescriptor parameter can access files over DS, except FmpRunProgram, FmpSetWorkingDir, and FmpSetOwner because they perform system functions that should not be performed from a remote system.

The file descriptor must contain 63 or fewer characters, including the remote user account name and node specifications. As a result there may be some files that cannot be accessed over DS because they have a long filename or directory path that cannot fit with the DS information into the 63-character filedescriptor.

The name-building and parsing routines return the DS field as their last parameter. The returned DS field contains the DS delimiters. If a file is located in a remote system, the name returned by FmpFileName includes the node name.

Some of the FMP routines do not perform exactly the same over DS as they do on a single system. The limitations as follows:

- FmpOpen does not use a DCB buffer larger than 8 blocks (1024 words), even if a larger buffer is specified.
- FmpOpen cannot open an LU at a remote system. It returns an error if such an attempt is made.
- FmpOpenFiles can only identify the program that has a file open if the program and the file are on the same system. If a file is open via DS, FmpOpenFiles reports that it is open, but can not report the name of the program that has it open, because all files opened via DS are opened by the TRFAS program.
- Files opened exclusively via DS are honored, except for FMGR files.

Special Purpose DS Communication Routines

The following calls permit your programs to perform special functions, all with DS transparency. They allow you to establish connections to accounts at remote systems.

CAUTION

The following routines are internal FMP routines, so they should be used with some caution. For example, it is possible to inadvertently close the wrong file by passing an incorrect connection number.

All of the variables used by the special purpose routines are single integers, except as noted.

DsCloseCon

DsCloseCon closes a connection opened by DsOpenCon.

```
error = DsCloseCon(conn)
integer error, conn
```

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

conn

An integer that specifies the connection number.

It is important to close connections when the DS operations are completed, because only 63 connections are available, and they are not automatically released when the calling program terminates or when the DS operations complete.

DsDcbWord

DsDcbWord returns the first word of the DCB as it would appear if the file associated with it was not opened through DS.

```
error = DsDcbWord(conn,word)
integer conn, word
```

conn

An integer that specifies the connection number.

word

An integer that returns the first word of the DCB.

DS transparency is implemented by replacing the first word of the DCB with the negative connection number. A DCB associated with a file over DS is detected by examining bit 6 of the first word of the DCB, but that practice is not recommended.

DsDiscInfo

DsDiscInfo returns the number of tracks and blocks per track of the specified disc volume on the system associated with the connection number.

```
error = DsDiscInfo(conn,lu,ntracks,bpert)
integer error, conn, lu, ntracks, bpert
```

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

conn

An integer that specifies the connection number.

lu

An integer that specifies the LU of the disc volume about which the track and blocks per track information is wanted.

ntracks

An integer that returns the number of tracks for the specified disc volume.

bpert

An integer that returns the number of blocks per track of the specified disc volume.

DsDiscRead

DsDiscRead reads the disc on the system specified by the connection number.

```
error = DsDiscRead(conn,buf,len,track,sector)
integer buf(*), error, conn, len, track, sector
```

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

conn

An integer that specifies the connection number.

buf

An integer array that returns data from the disc.

len

An integer that specifies the amount of data to be read. A maximum of 4096 characters can be read.

track

An integer that specifies the track from which to read.

sector

An integer that specifies the sector from which to read (63 words per sector).

The first word of the DCB that contains conn must first be set by DsSetDcbWord.

This routine should be used only by users with a detailed knowledge of DCBs and their contents.

DsFstat

DsFstat performs an FSTAT call for the system associated with the specified connection number.

```
error = DsFstat(conn,buffer,len[,iform[,iop]])
integer buffer(256), error, len, iform, iop
```

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

conn

An integer that specifies the connection number.

buffer

An integer array that returns the status of the cartridges.

len

An integer that specifies the length of the buffer in words.

The iform and iop parmeters are optional parameters that are used only when the remote node is an RTE-6/VM system. These parameters are identical to the iform and iop parameters in the FSTAT call for RTE-6/VM (see the RTE-6/VM Programmer's Reference Manual for a description).

DsNodeNumber

DsNodeNumber returns the node number associated with the specified file.

```
node = DsNodeNumber(filedescriptor)
character*(*) filedescriptor
integer node
```

node

An integer that returns the number of the node associated with the specified file. A zero is returned if the file is not remote.

filedescriptor

A 63-character string that specifies the name of a file.

DsOpenCon

DsOpenCon opens a connection to the remote user account/node specified.

```
error = DsOpenCon(string,conn)
integer error, conn
character*(*) string
```

string

A character string that specifies the remote user account name, node name, or both, along with the required delimiters, as in ">27", ">SYS3", "[USER]", and ">SYS3[USER/PASSWORD]". String must not contain a filename, only DS information.

conn

An integer that returns the connection number.

error

An integer that returns a negative code if an error occurs or zero if no error occurs.

The connection number returned by DsOpenCon is used in the other DS communication routines to identify the connection.

DsSetDcbWord

DsSetDcbWord changes the first word of the DCB to make the DsDiscRead routine work.

```
error = DsSetDcbWord(conn,word)
integer error, conn, word
```

error

An integer that returns a negative code if an error occurs or a zero if no error occurs.

conn

An integer that specifies the connection number.

word

An integer that specifies the word to be changed.

This routine should be used only by users with a detailed knowledge of DCBs and their contents.

Example Programs for FMP Routines

Three sample programs follow. The first program simply demonstrates the use of the simplest (open, close, read, write) FMP routines. The second shows how file masking, a somewhat more advanced FMP function, is used. The third combines many of the FMP routines in a advanced application.

Read/Write Example

The following program copies one file into another, one record at a time. It illustrates the use of FmpOpen, FmpRead, FmpWrite, and FmpClose, as well as FmpReportError.

```
ftn7x,s
      program copy
      implicit integer(a-z)

c  Program to copy a file to another file.

      integer dcb1(528), dcb2(528), buffer(128)
      character file1*30,file2*30

c  Open the source and destination files;
c  use big DCB's to go fast.

      call fparm(file1,file2)
      type1 = FmpOpen(dcb1,err,file1,'ros',4)
      if (err .lt. 0) goto 10

      type2 = FmpOpen(dcb2,err,file2,'woc',4)
      if (err .lt. 0) goto 20

c  copy the data

      do while (.true.)
         len = FmpRead(dcb1,err,buffer,256)

c  look for errors and end-of-file

         if (err .lt. 0) goto 10
         if (len .eq. -1) goto 30

c  none of those, so write the record.

         call FmpWrite(dcb2,err,buffer,len)
         if (err .lt. 0) goto 20
      enddo

c  come here to report errors

 10   call FmpReportError(err,file1)
      goto 30
 20   call FmpReportError(err,file2)

c  come here to close files and quit

 30   call FmpClose(dcb1,err)
      call FmpClose(dcb2,err)
      stop
      end
```

Mask Example

The following program shows how FmpInitMask, FmpNextMask and FmpMaskName can be used to generate a list of files which match a mask.

```
ftn7x,l,s
      program files
      implicit integer (a-z)

c files lists the names of files which match the mask

      integer dirdcb(356), entry(32)
      character curpath*(63), newname*(63), mask*(63)
      logical FmpNextMask

c get the mask

      call fparm(mask)

c initialize the directory dcb, report errors

      if (FmpInitMask(dirdcb,err,mask,curpath,356) .lt. 0) then
         call FmpReportError(err,mask)
         stop
      endif

c while errors are nonfatal, print name of file

      do while (FmpNextMask(dirdcb,err,curpath,entry))
         if (err .lt. 0) then
            call FmpReportError(err,curpath)
         else
            call FmpMaskName(dirdcb,newname,entry,curpath)
            write(1,*) newname
         endif
      enddo

c if search ended with error, print error

      if (err .lt. 0) then
         call FmpReportError(err,curpath)
      endif
c
c close down mask search
c
      call FmpEndMask(dirdcb)
      stop
      end
```

Advanced FMP Example

The following is a much larger program that builds a data base and writes records to it.

In the example, a FmpUniqueName is called to create a unique file name for the data base in the directory "CRDB" with a file type extension of "DAT". The program illustrates name building, file positioning, and many other less-frequently used FMP routines. The database built here is simply a type 2 file, it should not be confused with an Image data base.

```
ftn7x,s
      program crdb
      implicit integer(a-z)

c  Program to create a database in a type 2 file

      parameter (recordlen=30)
      parameter (recordbytes=2*recordlen)
      parameter (filesize=24)

      integer dcb(144), buffer(recordlen)
      character name*63, asciitime*28, charbuffer*(recordbytes)
      character tempname*16

c  Note use of double integers for times, record numbers

      integer*4 time, currec

c  Allow "charbuffer" as the string version of "buffer"

      equivalence (buffer,charbuffer)

c  Make up the name

      call FmpUniqueName('D',tempname)
      call FmpBuildName(name,tempname,'DAT',0,'CRDB',2,
     *                  filesize, recordlen,' ')
      namelen = trimlen(name)

c  Open the database for read, write; create it; update is implicit.

      call FmpOpen(dcb,err,name,'RWC',1)
      if (err .lt. 0) goto 20
```

```
c  Print the file name, and when it was created

     err = FmpCreateTime(name,time)
     if (err .lt. 0) goto 20
     call daytime(time,asciitime)
     write(1,*) 'File ',name(1:namelen),' created ',asciitime

c  Loop on adding records

     do while (.true.)

c  See what record number to change

   5 write(1,*) 'Record to add? _'
     read(1,*,end=10,err=10) currec

c  Position to this record (let FMP trap bad record number)

     call FmpSetPosition(dcb,err,currec,-1J)
     if (err .eq. -12) then
        write(1,*) 'That record doesn't exist'
        goto 5
     endif
     if (err .lt. 0) goto 20

c  Get a value for the record

     write(1,*) 'Enter record contents: _'
     read(1,'(a)') charbuffer

c  Put it in the file

     call FmpSetPosition(dcb,err,currec,-1J)
     if (err .lt. 0) goto 20
     call FmpWrite(dcb,err,buffer,recordbytes)
     if (err .lt. 0) goto 20

c  Post the file to show what to do if this is shared access

     call FmpPost(dcb,err)
     if (err .lt. 0) goto 20
   enddo

c  Come here when the last record is entered

  10 write(1,*) 'All done'
     goto 30
```

```
c  Come here to report errors

  20 call FmpReportError(err,name)

c  Come here to close file, purge it, and quit

  30 call FmpClose(dcb,err)
     err = FmpPurge(name)
     if (err .lt. 0) then
        call FmpReportError(err,name)
     endif
     stop
     end
```

Chapter 7
File System Utilities

This chapter describes the disc management utilities provided for use in the CI file system: FSCON, FPACK, FREES, FOWN and FVERI. FSCON converts the directory structure of an FMGR cartridge to that of the CI hierarchical directory structure. FPACK packs a CI disc volume to increase disc free space. FREES and FOWN report the amount of free space on the disc and the amount of disc space used by file owners. FVERI verifies the validity of the disc volume directories and tables.

FSCON File System Conversion

FSCON converts the directory structure of an FMGR cartridge, creating a hierarchical directory entry for each file on the cartridge. After conversion, the files have all the characteristics of the CI file system (time stamps, file type extensions, etc.)

Note that programs not converted to the hierarchical directory structure may have difficulty accessing the converted files.

Requirements for Successful Conversion

Before beginning the conversion, FSCON checks to see if the following prerequisites are satisfied. If any one of the conditions is not met, FSCON terminates with the appropriate error message.

There must be sufficient free space between the last file and the FMGR directory at the end of the cartridge to create the new directory and free space table. The amount of space required depends on such factors as the number of files in the directory and the number of extents. You can most likely meet this requirement by purging unneeded files and then using the FMGR PK command to pack the disc before calling FSCON. If there is not enough disc space for the conversion, FSCON exits with the message "Not enough free space on disc".

The total size of the cartridge cannot exceed 128k blocks. This is due to a limitation on the size of the free space table. If the FMGR disc cartridge exceeds this size, FSCON terminates with the message "Disc too big to convert".

The disc must be dismounted. This is to preclude the possibility of any open files, swap files, or active type 6 files. If the disc cartridge to be converted is mounted, FSCON terminates with the message "Cartridge must be dismounted".

The disc must be an FMGR cartridge. FSCON will only convert FMGR cartridges. Otherwise, it terminates with the message "Doesn't look like an FMGR disc".

Operating Instructions

To call FSCON, enter the runstring:

```
RU,FSCON,LU
```

LU is the LU of the FMGR cartridge to be converted. If the LU is omitted, FSCON issues a message defining the correct runstring, and then terminates.

FSCON scans the directory on the given LU and builds a new directory in the unused space at the end of the disc, before the FMGR directory. At the same time, it builds a free space table. The FMGR directory is scanned once to determine if it will be possible to do the conversion, and then scanned again to actually build the CI directory.

File data is never moved during the conversion process; only the directory structure changes. The new directory is built in unused space, and the entire conversion is done before any change is made to the FMGR directory structure. Thus, if the conversion fails at any point short of completion, the FMGR directory will still be in place and all of the files will be unchanged.

The final step in the conversion is to overwrite the FMGR directory with the CI, hierarchical structure, directory. This is accomplished in a single disc write operation. The new directory name will be the CRN of the FMGR cartridge. In this way, the name of a file will be unchanged (except as noted in the following section). That is, a reference to &SORC::DB will access the same file both before and after the conversion.

After the successful conversion FSCON issues the message "Cartridge converted". If an error occurs during the conversion process, FSCON issues the appropriate error message followed by "Cartridge not converted" then terminates.

File Renaming

If the name of a file includes the character "." or "/", these characters are changed to "~" and "|", respectively. This change is necessary because the CI file system attaches special significance to these characters: the "/" delimits directories and subdirectories and the "." delimits the type extension and mask qualifier. If these names were not changed, the file would never be found. As each file name is changed, the the message

```
Renaming <name> to <name>
```

appears on your screen. After conversion, the files can be listed using the directory list (DL) command:

```
DL,@~@::dir      *list all files with "." changed to "~"

DL,@|@::dir      *list all files with "/" changed to "|"
```

Converted CI Directory Entries

In building the new CI directory, information required for the directory entries is obtained from the FMGR cartridge directory, from scanning the files, or from the default values. The entries of a newly converted CI directory contain the following information (refer to the System Manager's Manual for the format of the file directory entry):

word		
1	flag:	protection set to rw/rw for all files and for directory; backup bit (bit 8) set
2	type:	taken from FMGR file directory entry
5	size:	taken from FMGR file directory entry
6	recln:	type 1,2 files - taken from FMGR file directory entry; all other files - calculated by scanning file
9-16	name:	taken from FMGR file directory entry as modified to change special characters
17-18	ext:	extent type, blank
19-24	time:	create, access, update - set to current time
25-26	nblk:	type 1,2 files - taken from FMGR file directory entry; all other files - calculated by scanning file
27-28	eof:	type 1,2 files - taken from FMGR file directory entry; all other files - calculated by scanning file
29-30	nrec:	type 1,2 files - taken from FMGR file directory entry; all other files - calculated by scanning file

Error Messages

If any of the following errors occur, FSCON issues the related message and terminates:

Cartridge not converted

This message will appear with a definitive message if an error occurs during conversion. If the cartridge is not converted, the files are all intact and the FMGR directory structure is unaltered.

Bad lu parameter

The lu parameter is not a disc lu.

Cartridge must be dismounted

The cartridge to be converted must be dismounted. This guarantees there will be no open files, active type 6 files or the swap file on this cartridge.

Disc too big to convert

Disc has more than 128K blocks (2^17).

Doesn't look like an FMGR disc

FSCON will only convert FMGR discs into CI file system disc volumes.

Insufficient memory, size up program

FSCON uses free space for tables, and will run faster with more memory. Sufficient memory for at least one track of the disc is required. If the LU contains many type 1 or type 2 files with extents, more free space will be required. Resize the program.

Not enough free space on disc

FSCON requires sufficient free tracks between the last file and the FMGR directory to build a new directory and other tables. Correct this condition by purging some files and packing the disc, using the FMGR PK command, before trying the conversion again.

Open file: xxxxxx

This message should never appear, because the cartridge must be dismounted before the conversion begins. However FSCON checks for open flags on files as it converts them.

FPACK File System Pack

FPACK rearranges the files on a disc volume, packing the files together more tightly to increase the size of the largest free space on the volume. When the operation is complete, there usually will be free space at the high end of the volume. After the disc volume has been packed, you can run the utility FREES to determine the amount of free space and the largest area of free space obtained by FPACK.

Operating Instructions

To run FPACK, enter the runstring:

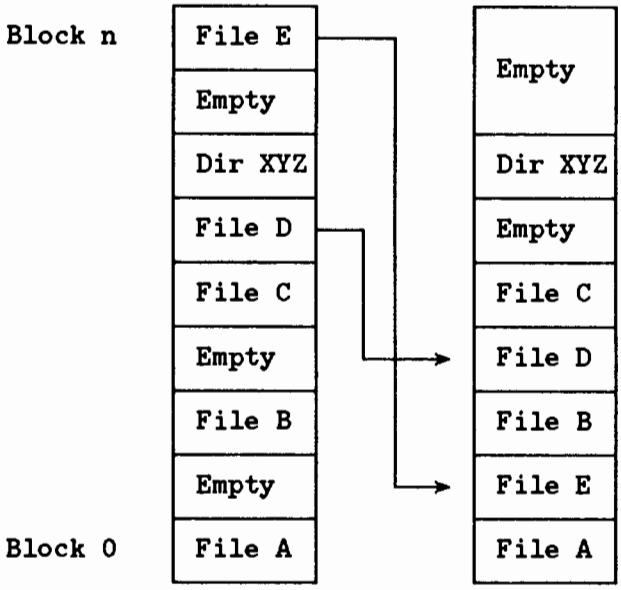
```
RU,FPACK,LU
```

LU is the LU of the volume to be packed. If no LU is given, FPACK issues a message defining the correct runstring, then terminates. Note that to copy a file, you must have read/write access to both the file and its directory; generally, only a System Manager has access to all files and directories.

FPACK scans the directories and generates a list of files in the order of their location on the disc volume. FPACK then copies files from higher numbered blocks to free spaces in lower numbered blocks on the disc, then purges the original files and marks the original file blocks as free space. A file is copied only if there is an area of free space below it that is large enough to contain the file and its extents. When a file is copied, any extents are copied into the main. All other attributes of the file (time stamps, protection, etc.) are not changed.

Because the integrity of a directory cannot be guaranteed if it is moved, FPACK does not copy directories. Open files, type 6 files, and the swap file also are not copied.

The process is illustrated below. Assuming that all files are the same size, FPACK would convert the disc volume structure on the left into the one on the right.



File E, the file in the highest numbered block, is copied into the free space in the lowest numbered block. Since FPACK does not copy the directory, the next file to be copied is File D. This process frees one area of space at the top, thus enlarging the available free space area on the volume. An area of free space still remains below directory XYZ.

After FPACK has moved as many files as possible, the LU is scanned again and FPACK issues an ordered listing of the files that can be moved, beginning at the highest disc volume address (a maximum of ten files are listed). The files listed are generally those that could not be copied -- type 6 files, open files, directories, the swap file. The entry for each file includes name, directory, file type, size, and record length. After printing the list, FPACK exits and you can run FREES to see the amount of free space now existing on the disc volume and the size of the largest area of free space. (Refer to the description of FREES in this chapter for the utility output format.)

If there still is not enough free space, you can increase the size of the free space by moving the appropriate files. If directory XYZ (in the preceding example) can be moved down, the largest free space will be enlarged. A procedure for moving the directory is given in the next section. If file C can then be moved, there will again be more space in the largest free space. If file C cannot be moved, moving file D will not help.

An alternate method to increase free space is to clear some existing files from the disc volume. After purging some existing files, run FPACK again to move files into the space left by the purged files.

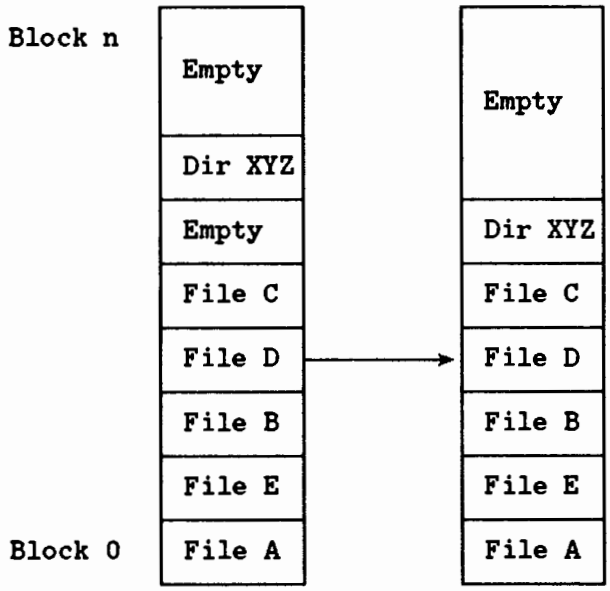
If the pack operations still have not created enough space, back up the entire volume on tape using the utility TF (refer to the RTE-6/VM Utility Programs Reference Manual), then reinitialize and restore the volume from tape. The CI file system automatically restores files beginning at the lowest numbered block on the disc volume. Note that if this is the system volume, the system must be capable of being booted from a volume other than the one being reinitialized.

Moving Directories

The following sequence of CI commands will move the global directory XYZ to a different disc location. The leading slash in the command argument is required to indicate that the named directory is a global directory.

WD /XYZ	Change the working directory to XYZ. This simplifies the command sequence.
CRDIR /TEMP	Create the temporary directory TEMP. The directory will be created on the same disc volume as the working directory, and at the lowest numbered block available.
MO @.@ /TEMP/@.@	Move all of the file entries from XYZ to TEMP. Only the directory entries are moved; the file data is unaffected. As each directory entry is moved, a message is issued to your terminal to define the entry being moved. A successful move is indicated with the notation [ok]. If an entry cannot be moved, the notation [failed] is given, followed by the reason for the failure. (A common reason is an open file on XYZ.) If you are unable to correct the failure and successfully move the file, move all files back to directory XYZ (mo /temp/@.@ /xyz/@.@) and stop the attempt.
WD /TEMP	Change the working directory to TEMP.
PU /XYZ.DIR	Purge the old directory, freeing the disc space. (Note that you must include the type extension .DIR when purging a directory.) If the directory is not empty, it cannot be purged. Move the files back to XYZ, purge TEMP and terminate the action.
RN /TEMP /XYZ	Restore the original name. The files now have the same name as before, but the directory has moved.

After this operation, the example disc volume structure will have been converted as follows:



Subdirectories can be moved using the same sequence; however you must specify the hierarchical path, as:

```
CRDIR /XYZ/TEMP.DIR
MO /XYZ/SUB/@.@ /XYZ/TEMP/@.@
PU /XYZ/SUB.DIR
RN /XYZ/TEMP.DIR /XYZ/SUB.DIR
```

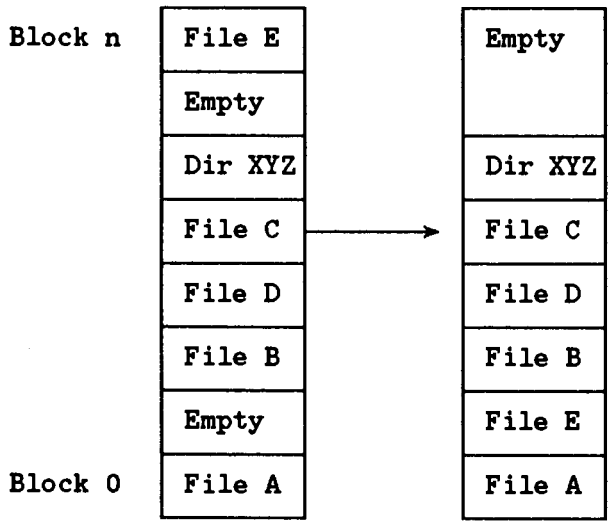
The concept is the same: create another subdirectory, move all of the files into it, purge the original and rename the new subdirectory to the original name.

Moving Files

When a file is copied with the CO command to a disc volume, it is placed in the lowest free space large enough for the file. This allows moving files on a disc volume to recover free space. If a file is copied to a different name on the same directory, the new copy of the file will be moved to a lower free space. For instance, using the original example in this section, file E can be manually moved to the lowest free space using the following command sequence (assuming file E is on directory XYZ):

WD /XYZ	Change the working directory to XYZ to simplify the command sequence.
CO E X	Copy file E to file X which does not exist. File X will be placed in the lowest free space available.
PU E	Purge the original file.
RN X E	Restore original file name.

After this operation, the example disc volume structure is left with a larger free space.



FREES (Report Disc Free Space)

FREES scans the free space table on CI disc volumes and reports the amount of free space and the size of the largest area of free space. The amount of free space on a volume is an indication of how much more can be stored on that volume. The largest free space defines the largest file that can be created on that volume.

FREES reports total free space in blocks and as a percentage of total space. The size of the largest free space is reported in blocks and as a percentage of the total available free space. This gives an indication of the amount of fragmentation of the volume. The lower the percentage, the more the volume is fragmented, and the more effective the FPACK utility will be at packing the volume.

Operating Instructions

To run FREES, enter the runstring:

```
CI.65> frees[ LU]
```

LU is the LU number of the CI file system disc volume to be scanned. If no LU is given, FREES scans all of the mounted volumes, reporting on each one individually.

Examples

The default case:

```
CIxx> frees
Analyzing LU 54
Total Blocks: 12992  (Reserved blocks: 0)
Free Blocks: 8324
Free space is 64% of total space
Largest Free Space: 8051 Blocks
Largest Free Space is 96% of Total Free Space
      :
Analyzing LU 27
Total Blocks: ...
      :
```

Specifying LU to be scanned:

```
RU,FREES,54
Analyzing LU 54
Total Blocks: 12992 (Reserved blocks: 0)
Free Blocks: 8324
Free space is 64% of total space
Largest Free Space: 8051 Blocks
Largest Free Space is 96% of Total Free Space
```

FOWN (Report File Space by Owner)

FOWN scans specified files and identifies, by owners, the total disc space used by files that match the mask given in the utility runstring.

Operating Instructions

To call FOWN, enter the runstring:

```
RU,FOWN[,mask]
```

The mask parameter specifies the class of files to be scanned. Refer to Chapter 3 for a description of file masks. The default mask is /@.@.s, which matches all files in the file system. This will tell how much disc space is being used by all users who own files.

In some instances, FOWN will be unable to identify the owner of a file. When this happens, FOWN displays the system number that corresponds to the owner. Usually this means that the owner of those files is no longer a user on this system. FOWN also cannot determine the owners of remote files, so all owners are referred to only by number.

Examples

The following example uses the default mask. Note that user number 7 is no longer known to the system. Note also the message "FMGR files not scanned". This indicates that some FMGR files matched the mask but were not counted. There is no ownership information available for FMGR files.

```
CI.10> fown
Scanning...   Mask = /@.@.s

   Owner          Disc Blocks
DOUG.LAB              6715
DON.TRAINING          4032
MANAGER.SYS           2761
DOUGL.SYS              328
NAOMI.GENERAL           69
Unknown (#7)           198
                      -----
Total                14103

FMGR files not scanned
```

In the following example, all type 6 files are counted and their owners identified.

```
CI.10> fown,@.@.e:::6
Scanning...   Mask = @.@.E:::6

  Owner          Disc Blocks
DOUG                  2943
DON                    433
DOUGL                  148
                      -----
Total                 3524

FMGR files not scanned
```

In the following example, ownership ID requested for DON only, thus the format of the output is changed.

```
CI.10> fown,/don/
Scanning...   Mask = /DON/
Owner: DON Total Blocks: 641
```

FVERI (File System Verification)

FVERI scans the directory and table structures of a CI file system disc LU and reports on areas where the data is inconsistent. Two kinds of checks are done.

1. Internal consistency of the directories and correctness of the bit map (the "bit map" is a table on the volume which keeps a record of used and free space on that volume; it is also referred to as the "free space table").

   - FVERI checks directory consistency by looking for legal values within the entries: extent pointers pointing to valid extents, data pointers having legal disc addresses, non-negative file types, and so on.

   - FVERI checks the bit map by building its own version of the bit map (based on the files it finds on the volume) and then comparing the two bit maps.

2. Consistency between directory entries and their associated files (number of records in the file, number of words in the file, etc.), and internal consistency of the files (valid record length marks, valid EOF mark, etc.)

   - FVERI checks both the consistency between directory entries and their associated files, and the internal consistency of the files, by reading through each file it finds (using normal FMP calls). It collects appropriate data as it is reading (number of records in the files, number of words, etc.), then compares what it found with the directory entry for that file.

If an LU is specified in the runstring, and the +B option is not given, FVERI will perform both types of checks. With the +B option, FVERI will only perform the first type of check. With a given mask, FVERI will verify all files described by the mask, performing only the second type of check. Because the bit map covers the entire volume, the bit map check cannot be done if a specific mask is specified, because FVERI may not be looking at all the files on the volume.

Note that because the second type of verification requires that FVERI read through every file, it will be slower than the first check. For this reason, if you want to verify the overall integrity of a disc volume, but do not need to check each individual file, it will be faster to use the +B option. On the other hand, if you are concerned about file integrity on a certain area of the volume, specifying a mask can be relatively fast because you are not asking FVERI to check the whole volume.

This utility can be used after a system crash to verify that the file system is still intact. It should also be run from time to time to verify the integrity of the file system. In order to gain access to all read-protected files on the disc, FVERI is best executed by a System Manager.

FVERI is most efficient when verifying a disc not in use. If other programs are creating, purging, or modifying files while FVERI runs, the tables will appear inconsistent due to synchronization errors. For instance, if a file is purged after being verified but before the free space table is verified, the free space table can appear to be invalid.

Operating Instructions

To run FVERI, enter the runstring

```
RU FVERI[ lu/mask][ options]
```

where:

lu = LU of volume to be verified (FVERI does not verify FMGR cartridges). If an LU is specified, the entire volume is verified.

mask = mask describing a set of files to be verified. If a mask is specified, only the files described by the mask are verified.

options = one or more options in any order; legal options are:

+B = verify bit map only; illegal if a mask is also specified.

+L,file/lu = list file or device for errors.

Running FVERI with a ? or ?? (RU,FVERI,?[?]) causes FVERI to display usage information as shown above.

If neither an LU or a mask is given, the default is to verify all mounted volumes. For each volume, the message:

```
Verifying LU xx
```

is issued while the verification is taking place. FVERI can be halted before completion with a BR,FVERI. The utility quits immediately without completing a verification in process. When performing a complete verification of a volume, FVERI can take several minutes to run, because it reads every variable length record file on the LU.

Error Messages

FVERI is susceptible to all FMP errors, which are reported if they occur. It also reports any errors detected in the table structures. Each error message is preceded with a number indicating the relative importance of the error. Most FMP errors are displayed as severity level four; some, usually protection violations, are reported with a severity of zero. The higher the number, the worse the problem. Continued use of an LU with a reported level 9 error can cause loss of data.

The error message usually includes a block number indicating the block on the disc where the bad data was found. This is either the block number of the directory entry for some file, or the block on the disc represented by the invalid data in the free space table.

Where possible, the error output will indicate the file whose directory entry is corrupt. This gives some clue to the logical part of the disc that is corrupt, to complement the physical location information provided by the block number.

As an example, the following message defines a level 6 error in the directory entry for file PG1.FTN::JAN. This directory entry is at block 1435 of the disc.

```
(6) Total blocks in file less than main size at block 1435
On file PG1.FTN::JAN
```

The following list of error messages is arranged in ascending order of severity, from zero to nine. The first four 0-level errors cause FVERI to terminate; all other error conditions are not fatal.

(0) Break detected; Verification terminated.

(0) Internal buffer too small, size up program.

(0) Not a hierarchical file system disc.

This error will be produced if the volume is not a disc, or if the volume header is corrupt.

(0) Disc volume not mounted.

The volume must be mounted to be verified. If you cannot mount the disc, FVERI cannot give you any further information.

(0) Record Length exceeds 512 bytes.

FVERI has an internal buffer of 512 bytes for reading type 3 and above files. A record of more than 512 bytes was read, and further verification of this file's contents will not be done.

(2) Directory Tag field is incorrect.

A special 32-bit tag is set for directories as a redundancy check to verify that this is in fact a directory. This directory does not have the proper tag value.

(2) Access time earlier than Update time.

The file access date and time must be equal to or later than the update time.

(3) Record Length Incorrect at block <nnnnn>

The record length field in the directory does not reflect the length of the longest record in the file. Some programs may have the file open, or the file may have been created in a non-standard way, or created with a program that misuses the FMP routines.

(3) Number of words in file incorrect at block <nnnnn>

The end-of-file pointer in the directory does not match the end-of-file mark in the file. Some program may have the file open, the file may have been created in a non-standard way, or created with a program that misuses the FMP routines. (FMPAPPEND calls on this file will fail.)

(3) Number of records in file incorrect at block <nnnnn>

The record count in the directory does not match the number of records in the file. The file may have been created in a non-standard way or with a program that misuses the FMP routines.

(3) Directory header flag incorrect at block <nnnnn>

The directory header and trailer (the two ends of a directory extent) should have a particular identifying flag. This directory doesn't have one.

(4) EOF pointer is beyond last block at block <nnnnn>

The last word of the file is reported to be beyond the last block in the file.

(4) Free space is marked as used space at block <nnnnn>

The free space table (the bitmap) has some disc space marked as used. However the file system does not have any files or directories in that space. The space is wasted and unrecoverable until the file system is completely backed up and restored. This can be caused by having read-protected directories FVERI could not verify. FVERI will work better if run by a System Manager.

(5) Unidentifiable directory entry at block <nnnnn>

There is an entry in the directory that is not a file main, an extent, a purged file, or anything else defined for the file system.

(5) Directory main size does not equal extent size at block <nnnnn>

Each directory extent should be the same size as the main. This is not true for this directory.

(6) Actual blocks in file entry is wrong at block <nnnnn>

The sum of the size of the main and all of the extents is not the same as the recorded total number of blocks used by this file.

(6) Extent entry back pointer is wrong at block <nnnnn>

Extent entries have a pointer that points back to the previous extent entry or the main (extent entries are in a doubly linked list). The return pointer in this extent entry does not point back to the right place.

(6) Total blocks in file less than main size at block <nnnnn>

The total number of blocks used by this file is less than the number of blocks in the main.

(6) Illegal file type at block <nnnnn>

File types must be greater than zero.

(7) Invalid directory extent pointer at block <nnnnn>

The pointer to the next or previous directory extent points to a disc address beyond the valid address space on this disc.

(7) Extent entry flag is wrong at block <nnnnn>

Extent entries in the directory should have a particular identifier flag. This extent does not have the right flag, yet is pointed to as an extent of this file.

(8) Invalid extent data pointer at block <nnnnn>

The pointer to the extent data points to a block address outside the legal address range of this disc.

(8) Invalid extent forward pointer at block <nnnnn>

The pointer to the next extent entry points to a block address outside the legal address range of this disc.

(8) Invalid extent pointer in main at block <nnnnn>

The pointer to the first extent of this file points to a disc address beyond the valid address space of this disc.

(8) Illegal directory size at block <nnnnn>

Each directory extent must be from 1 to 64 blocks long. This directory has a size not in that range.

(9) Duplicate use of disc block <nnnnn>

Two or more files are stored on the same section of the disc. At least one of the files must be corrupt. This message can occur as a result of other file activity on this disc while FVERI is running. Run FVERI again. The same block should not be reported on two consecutive runs of FVERI unless the error is real.

(9) Blocks per bit value is illegal at block <nnnnn>

The free space table represents up to 128 blocks of disc data per bit in the free space table. There can be no more than 8192 words in the free space table, or the free space table overflows. If this value is wrong, the allocation of space on the disc can be corrupt.

(9) Used space is marked as free space at block <nnnnn>

There is a space on the disc pointed to by a directory entry, however it is not marked used in the free space table. This space is liable to be reclaimed at any time by the file system for some other file. This message can occur as a result of other file activity on this disc while FVERI is running. Run FVERI when the volume is not in use by other programs.

Error Recovery

There are several possible responses to inconsistencies detected in the CI file system. Minor errors can be ignored. The error can sometimes be corrected by copying the affected files to another disc or tape, purging them from the verified disc, then restoring the files. As the files are restored, the file system will be properly recreated.

In the case of inconsistencies in the free space table (the bitmap), it will be necessary to save the entire disc to tape using TF. Then initialize the disc and restore the disc from tape. This procedure should always clear errors.

The disc save/restore procedure should be used for all errors reported at a severity of 7 or higher. It should also be used if any other errors are detected that cannot be cleared by copying the file to a new location.

Appendix A
Error Messages and Codes

Most of the error messages caused by an operator action are simple self explanatory messages. But there are some that are displayed in the form of an error code or in a particular format where a number of variables may be displayed. The common error formats are described below and the operator error messages are listed in alphabetic order with the explanation, and corrective active if not obvious, given under the heading Error Messages in this appendix. Error codes are given in numerical order following the error messages.

Error Formats

Error messages have different formats depending upon the operation being performed. Errors reported by CI in response to commands, such as AS, RU, SZ, etc, are in the form of brief descriptive messages. For example:

```
Parameter is not proper for this command
Input is not proper for this command/program
```

There may be occasions (although rare) when error messages are reported by the system. For example:

```
No SAM available at this time to perform the request
The specified LU is not assigned on this system
```

System program D.ERR is used to generate the text of FMP error messages. If an FMP error occurs and D.ERR cannot be found by the system, the following message will be generated:

```
(warning -250) FMP error xxx
```

where "xxx" is the FMP error that occurred. The error code -250 is a warning indicating D.ERR not available.

Some errors are reported in the form of an error code. These are reported in the form:

```
FMP error -59
```

The error codes are listed and described under the heading Error Codes in this appendix.

Error Messages

Active working directory
: Reported by purge or dismount when you try to purge another user's working directory, or when you try to dismount a disc containing a working directory.

Bad record length
: Attempt to read or position to a record not written, or on update to write an illegal record length; check position or size parameters.

Bad password
: The correct password was not supplied.

Break flag detected
: User entered BR command, stopping the operation.

Can only run unshared
: Reported when a shared program cannot be run because there is no room in the shared program table. Use LINK to make it non-shareable, or make room in the shared table by OFF'ing programs or by regenerating the system.

Cannot access account
: Logon error occurred other than the normal user, account, and session limit errors.

Cannot change that property
: Rename operations cannot change whether the file is a directory, nor can they change the file type, size or record length.

Changed RPL checksum
: The program file was linked with a snap that specified different microcode instructions (RPLs), or the same ones in a different order. Your program may or may not work, depending on whether it uses any instructions that are not there. If it does not work, you will receive a UI (unimplemented instruction) abort. Whenever you get this message, the program file is changed to make it so that this error will not be reported again with the current system. It does not fix the problem, if any, but it keeps you from getting repeated error messages if the program works anyway.

Connection broken
: The remote system monitor TRFAS was restarted since the connection was opened.

DCB is not open
Attempt to access an unopened DCB.
Check error code on open attempt.

Device I/O failed
Illegal read/write on type 0 file Attempt to read/write or position type 0 file that does not support the operation; check file parameters.

Did not ask to read
Specify the 'R' option in the open request.

Did not ask to write
Specify the 'W' option in the open request.

Directory is corrupt
During a directory lock done by MC, DC, IN, PK, CR, or PU, the directory is scanned for internal consistency. If this occurs, copy the files to another disc, or just store the ones you need. Normally occurs when a disc is mounted before being initialized.

Directory is empty
No files are in the specified directory.

Directory is full
No more room in file directory; purge files and pack directory with FMGR PK command if possible, or try another cartridge.

Directory not empty
Directories can only be purged when they are empty. Purge the remaining files or move them to another directory; you may want to use a wildcard purge for this.

Directories not on the same LU
Rename operations do not move data, and data must be on the same LU as the directory, so rename operations can only rename a file into a directory on the same LU as the source file.

Directory read protected
You are not allowed to read one of the directories needed to access the file. You must gain access to the directory by changing the protection status.

Directory write protected
You are not allowed to write the directory containing the file, so you cannot change, purge it, or rename it.

Disc error
The disc is down; try again and then report it to the system manager of facility.

Disc I/O failed
Reported when D.RTR tries to access an LU that is down, or when any EXEC error occurs on an FMP disc access.

Disc is locked
Cartridge is locked; initialize cartridge if not initialized, otherwise, keep trying.

Disc is not mounted
The indicated volume was not mounted, so it cannot be dismounted and directories cannot be created on it. Try mounting it.

DSRTR not available
The DS transparency source monitor is not RP'd, so DS transparency does not work. Try to RP DSRTR. This may also indicate invalid characters in a file name (> or [ ).

Duplicate directory name
Duplicate name. Check destination of directory being created.

D.RTR EXEC request aborted
D.RTR has tried something unreasonable, probably because of a corrupted cartridge list or a disc error

D.RTR not available
D.RTR is not RP'd or has been removed. Check the BOOTEX file for "RP,D.RTR,D.RTR"; then reboot.

DS error DS08 (0), node 4
Generic error message when a strange DS error occurs. These are in the DS manual.

DS is not initialized
DS has not been started with DINIT.

DS link is not connected
Hardware problem.

File already exists
A file already exists with specified name; repeat with new name or purge existing file.

Files are open on LU
This LU cannot be dismounted because one or more files are open. The name of the first open file is printed by D.RTR.

File is already open
Attempt to open a file already open exclusively, or an FMGR file open to 7 programs.

File read protected
You are not allowed to read this file because of protection, or because it is a write-only device. Try changing the protection on the file.

File write protected
You are not allowed to write this file because of protection, or because it is a read-only device. Try changing the protection on the file.

Illegal file position
Attempt to read or write or position beyond the file boundaries; check record position parameters, result depends on file type and call.

Illegal interrupt from SC20
Reported when an interrupt occurs from a select code which is not in the system select-code table. Most likely a generation error in select code specification. Reported by the RTE system.

Illegal LU.
Attempt to access an undefined LU.

Illegal name
File name does not conform to syntax rules; correct name.

Illegal program file
Reported if a program file is not a type 6 file, or it was not loaded with a snap file compatible with the current system, or it is a non-transportable file from another system.

Illegal remote access
Usually indicates an internal error. Either an invalid connection number was specified, or an invalid request was routed to the DS transparency software.

Illegal use of directory
Directory used when it should not be, such as in creating a file with a DIR file type extension.

Incorrect security code
Attempt to access a file without the correct security code. Use the correct code or do not access file.

Input is not proper for this command
There may be a parameter missing or a bad number specified.

Invalid reserved partition request
The partition being assigned a program must exist, must be big enough for the program, and must not be downed due to a parity error.

LU has old directory
This LU has an old directory, and you did not tell FmpMount to re-initialize old directories.

Missing extent
A request was made for a file extent which was missing from the file. The file is probably corrupt. Purge the file.

More than 255 extents
An attempt to create more than 255 extents was made. Use a file with a larger initial size.

Must not be remote
The specified file or directory must be on the local system.

Must specify an LU
FmpCreateDir could not figure out where to create this directory. Either supply an LU, or set your working directory to a directory on the LU where the new directory is to be created.

No free ID segments
Cannot restore the program due to lack of ID segments. Try removing programs no longer needed.

No files selected
Nothing matches the mask supplied.

No such account
No user has name specified.

No such directory
One of the directories needed to find the file does not exist. It may be misspelled, or you may be using the wrong working directory.

No such cartridge
Specified cartridge is not mounted. Check disc specification.

No such file
Attempt to access a file that cannot be found. Check the file name or cartridge number.

No such node
The local system does not recognize the node number or name specified. It may not be in the NRV.

No such user
Reported by OWNER command when the user does not exist.

No TRFAS at remote system
Remote monitor TRFAS is not RP'd at the remote system.

No working directory
Returned by FmpWorkingDir when there is no working directory established, and by some other calls when a file name is specified with no directory but no working directory exists.

Program is active
A request to purge an active type 6 file was requested by FmpPurge. The program must be OF'd before the file can be purged. The swap file cannot be purged if swapping is enabled.

Program is busy
(Attributes cannot be modified when running/owning a partition) Program size and partition assignment can only be changed when the program is dormant and not saving resources. Also returned by XQ command when the program is running and cannot be cloned with a new name.

Parameter is not proper for this command
Some parameter value entered is out of range.

Program aborted
The program was OF'd or aborted before it ran to completion.

Program name exists
Cannot RP program with that name because there already is one. OF the old name with the ID parameter, or choose another name.

Ran out of disc space
Disc specified for a disc file has insufficient room for file create. Could occur when an extent is being created.

Remote system does not respond
The remote system is probably down, or not running DS.

Session limit exceeded
Cannot log on the remote system because too many sessions are already logged on.

String is too long
Returned by FmpReadString and FmpWriteString when the passed string is longer than 256 bytes.

System common changed
Reported when you try to RP a program with system common defined differently from the current system.

System programs can only be changed by superusers
Return when a general user uses the system manager capabilities in commands such as OF, PR, or SS; also returned if action is taken by a general user that may affect the operating system.

That LU is not assigned on this system
Invalid LU for a status request; usually not seen from CI. Reported by the RTE system.

The disc where the program resides is down
Hardware problem with the disc drive preventing execution of the program.

Too many directories
D.RTR has no room to record this global directory; may occur on mount or create directory. Closing some files or dismounting a volume should provide temporary relief; a long-term solution is to size D.RTR bigger or open fewer files or have fewer global directories. You may be able to rename some global directories to be subdirectories.

Too many open files
D.RTR has no room to record the open flag for this file. Closing some files or dismounting a volume should provide temporary relief; a long-term solution is to size D.RTR bigger or open fewer files or have fewer global directories.

Too many remote connections
No more than 64 files can be open at remote systems at any one time. Each open file requires a connection. Connections can be reclaimed by closing files.

Unable to schedule program PROGN on interrupt to driver
RTE system interrupt table specifies a program to run on interrupt, but that program was not dormant when the interrupt occurred. Reported by the RTE system.

Unknown for FMGR file
Returned when you ask for unavailable information about a FMGR file, such as time stamps.

Unpurge failed
Disc space or a directory entry occupied by the purged file has been reclaimed, so the file cannot be recovered.

Value is too big
Check command syntax for valid range.

Value is too small
Check command syntax for valid range.

Value must be positive
Enter proper positive value.

You do not own
You must be the owner of file to change its protection information, and you must own a directory to change it owner. A system manager is required to performed the desired task.

Error Codes

FMP error -3
Backspace illegal
Attempt was made to backspace a device (or type 0 file) that cannot be backspaced; check device type.

FMP error -4
Type 2 record length is zero.
Type 2 file record length must be a positive number.

FMP error -9
Attempt to position or force to 1 a type 0 file.
Type 0 files cannot be positioned or be forced to type 1; check file type.

FMP error -10
Not enough parameters Required parameters omitted from call; enter the parameters.

FMP error -16
Illegal type or size=0
Wrong type code supplied; attempt to create or purge type 0 file or create 0-length file; check size and type parameters.

FMP error -36
Lock error on device
A call to OPEN or OPENF specified exclusive use of a device which was already locked or no resource numbers were available. Try again or request non-exclusive use.

FMP error -38
Illegal scratch file number
The legal range of scratch file numbers is 0-99. Check your program.

FMP error -53
Program assigned to bad reserved partition.
The program is assigned to a reserved partition which is "bad" due to a parity error in a reserved partition which is undefined. Use the AS command to re-assign the program to a good partition.

FMP error -54
Program is assigned to a reserved partition which is too small
The program is assigned to a reserved partition which is not large enough to hold the program. The program must be assigned to a larger reserved partition.

FMP error -55
No room in shareable EMA table
The shareable EMA table already contains 15 entries. If possible, OF,,ID any programs not in use that access shareable EMA. Note that all programs that access a certain shareable EMA area must be OF'd for the shareable EMA table entry to be deleted.

FMP error -56
Shareable EMA assigned to a non-existent reserved partition
The shareable EMA area used by the program is assigned to a reserved partition which was not defined (by the AS command) at system bootup time. The program must be reloaded to change the shareable EMA assign number or the system must be rebooted to define the partition. (Remember that the first program RP'ed that uses a shareable EMA area determines where it is allocated. Perhaps another program that uses the shareable EMA area could be RP'd first.)

FMP error -57
Shareable EMA assigned to (or will be allocated in due to a previously RP'd program) a reserved partition which is too small
The shareable EMA area used by the program is assigned to a reserved partition which is not large enough to hold it. If all the programs that access the shareable EMA area do not specify the same shareable EMA size, this error could result.

FMP error -58
Program assigned to same reserved partition as shareable EMA area.
The program is assigned to the same reserved partition as the shareable EMA area the program accesses. Both must be in memory for the program to run, so one must be re-assigned to a different reserved partition. This error could result if the first program that uses that shareable EMA area assigns it to a reserved partition in which a second program that accesses it is assigned to run.

FMP error -59
Already 63 programs using shareable EMA area
There are already 63 programs RP'd that access the shareable EMA area specified by the program. No more programs may be RP'd (or run).

FMP error -60 through -100 NOT USED

FMP error -101
Illegal parameter in D.RTR call
Possible operator error; recheck previous entries for illegal or misplaced parameters.

FMP error -217
Bad directory block
Tag fields in the directory do not match, indicating a corrupt disc or working directory pointer. Try changing working directories. If that fails, use the file system status utility to check the situation.

FMP error -223
Illegal DCB buffer size
DCB buffer sizes must be in the range 1-127 blocks, except for type 0 and 1 files which ignore the size.

FMP error -227
Program does not fit in partition (SC08/09)
The program is too big for the partition it is assigned to, or for available memory. Try unassigning the program or assigning it to a bigger partition.

FMP error -228
No SAM to pass string (SC10)
System does not have enough SAM. Try shortening the string. If unsuccessful, regenerate the system with more SAM.

FMP error -243
Parameter error
One of the parameters specified is not a reasonable value.

FMP error -244
Mapping error
One of the VMA file routines got an error mapping VMA.

FMP Error Codes

-001 Disc error
The disc is down; try again and then report it to the system manager of facility.

-002 File already exists
A file already exists with specified name; repeat with new name or purge existing file.

-003 Backspace illegal
Attempt was made to backspace a device (or type 0 file) that cannot be backspaced, check device type.

-004 Illegal record length
Attempt to create a type 2 file with a zero record length.

-005 Bad record length
Attempt to read or position to a record not written, or on update to write an illegal record length; check position or size parameters.

-006 No such file
Attempt to access a file that cannot be found. Check the file name or cartridge number.

-007 Bad file security code
Attempt to access a file without the correct security code. Use the correct code or do not access file.

-008 File is already open
Attempt to open file already open exclusively or open to eight programs or cartridge containing file is locked; use CL or DL to locate lock.

-009 Attempt to position or force to 1 a type 0 file
Type 0 files cannot be positioned or be forced to type 1; check file type.

-010 Not enough parameters
Required parameters omitted from call; enter the parameters.

-011 DCB not open
Attempt to access an unopened DCB. Check error code on open attempt.

-012 Illegal file position
Attempt to read or write or position beyond the file boundaries; check record position parameters, result depends on file type & call.

-013 Disc locked
Cartridge is locked; initialize cartridge if not initialized, otherwise, keep trying.

-014 Directory is full
No more room in file directory; purge files and pack directory if possible, or try another cartridge.

-015 Illegal name
File name does not conform to syntax rules; correct name.

-016 Illegal type or size=0
Wrong type code supplied; attempt to create or purge type 0 file or create 0-length file; check size and type parameters.

-017 Illegal read/write on type 0 file
Attempt to read/write or position type 0 file that does not support the operation; check file parameters, namr.

-018 Illegal LU. LU not assigned to system
Attempt to access an undefined LU.

-030 Value too large for parameter
Value is greater than legal maximum.

-032 No such cartridge
Specified cartridge is not mounted. Check disc specification in call.

-033 Ran out of disc space
Disc specified for a disc file has insufficient room for file create. Could occur during a WRITF if an extent is being created.

-034 Disc already mounted
Disc is mounted as an FMGR or hierarchical volume.

-036 Lock error on device
A call to OPEN or OPENF specified exclusive use of a device which was already locked or no resource numbers were available. Try again or request nonexclusive use.

-037 Program is active
A request to purge an active type 6 file was requested by PURGE. The program must be off'd before the file can be purged. The swap file cannot be purged if swapping is enabled.

-038 Illegal scratch file number
The legal range of scratch file numbers is 0-99. Check your program.

-046 Greater than 255 extents
An attempt to create more than 255 extents was made. Use a file with a larger initial size.

-049 Copy verify failed
The verify option of the COPYF routine detected a discrepancy while verifying a transfer of data. Check the file for correctness.

-050 No files found
A "-" was specified in a namr, but there were no files matching the mask. Check the mask for correctness.

-053 through 98 NOT USED.

-099 D.RTR EXEC request aborted
D.RTR has tried something unreasonable, probably because the cartridge list has been corrupted.

-101 Illegal parameter in D.RTR call
Possible operator error; recheck previous entries for illegal or misplaced parameters.

-102 D.RTR not available
D.RTR is not RP'd or has been off'd; system should be rebooted.

-103 Directory is corrupt
During a directory lock done by MC, DC, IN, PK, CR, PU, the directory is scanned for internal consistency. If this occurs, copy the files to another disc, or just store the ones you need.

-104 Extent not found
A request was made for a file extent which was missing from the file. The file is probably corrupt. Purge the file.

-105 through 199 NOT USED.

-200 No working directory
Returned by FmpWorkingDir when there is no working directory established, and by some other calls when a file name is specified with no directory but no working directory exists.

-201 Directory not empty
Directories can only be purged when they are empty. To purge the directory, purge the remaining files (use a wildcard purge).

-202 Did not ask to read
This file is read-protected. Specify the 'R' option in the open request.

-203 Did not ask to write
This file is write-protected. Specify the 'W' option in the open request.

-204 File read protected
This file is read-protected or is a write-only device. Change the protection on the file.

-205 File write protected
This file is write-protected or is a read-only device. Either the file has write protection set or it has a positive security code which must be specified correctly in the open call. Change the protection on the file.

-206 Directory read protected
One of the directories needed to access the file is read-protected. Change its protection.

-207 Directory write protected
The directory containing the file is write-protected, so you cannot change its properties, purge it, or rename it.

-208 Directory already exists
That name already being used. Be sure the directory is being created where you expect it to be.

-209 No such directory
One directory needed to find the file does not exist. Its name may be misspelled, or the working directory may be wrong.

-210 Unpurge failed
Disc space or a directory entry occupied by the purged file has been reclaimed, so the file cannot be unpurged. Not repairable.

-211 Directories are not on the same LU
Rename operations do not move data, and data must be on the same LU as the directory, so rename operations can only rename a file into a directory on the same LU as it was originally.

-212 Cannot change that attribute
Rename operations cannot change whether the file is a directory, nor can they change the file type, size, or record length.

-213 Too many open files
D.RTR has no room to record the open flag for this file. Close some files or dismount a volume for temporary relief; a long-term solution is to size D.RTR larger, open fewer files, or have fewer global directories.

-214 Disc not mounted
The indicated volume was not mounted, so it cannot be dismounted and directories cannot be created on it.

-215 Too many directories
D.RTR has no room to record this global directory; this error can occur on mount or directory create. Close some files or dismount a volume for temporary relief; a long-term solution is to size D.RTR larger, open fewer files, or have fewer global directories. Perhaps some global directories can be renamed as subdirectories.

-216 You do not own
Only the file owner can change its protection information, and only the directory owner can change the file owner. The system manager does not get this error.

-217 Bad directory block
Tag fields in the directory do not match, indicating a corrupt disc or working directory pointer. Change working directories. If that fails, investigate the situation with the file system status utility.

-218 Must specify an LU
FmpCreateDir could not determine where to create this directory. Either supply an LU, or set the working directory to a directory on the LU where the new directory is to be created.

-219 No remote access
The passed name or DCB indicates that this file is located on a [possibly] remote system, so it must be routed through the DS transparency software before it is usable.

-220 DSRTR not available
The DS transparency source monitor is not RP'd, so DS transparency does not work. RP DSRTR.

-221 File are open on LU
This LU cannot be dismounted because one or more files are open. The name of the first open file is printed by D.RTR.

-222 LU has old directory
This LU has an old directory, and FmpMount was not told to re-initialize old directories.

-223 Illegal DCB buffer size
DCB buffer sizes must be in the range one to 127 blocks, except for type zero and one files, which ignore the size. This error is also returned by routines such as FmpCopy when the passed buffer is too small.

-224 No free ID segments
Cannot restore the program, due to lack of ID segments. Remove programs that are no longer needed.

-225 Program busy
FmpRunProgram reports that the program named in the XQ command is busy.

-226 Program was aborted
The program was OF'd or aborted before it ran to completion.

-227 Program doesn't fit in partition (SC08/09)
The program is too big for available memory or the partition to which it is assigned. Unassign the program or assign it to a bigger partition.

-228 No SAM to pass string (SC10)
The system does not have enough SAM to pass runstrings. If a shorter string does not work (it probably won't), rebooting may help is SAM is fragmented or may need to regenerate the system to get more SAM.

-229 Active working directory
Tried to purge a working directory or dismount a disc containing a working directory.

-230 Illegal use of directory
A directory was used illegally (e.g., to create a file).

-231 String is too long
A string longer than 256 bytes was passed to FmpReadString or FmpWriteString.

-232 Unknown for old file
Requested unavailable information (e.g., time stamp) about an old file.

-233 No such user
User name not found by FmpSetOwner.

-234 Size mismatch on copy
Source and destination file sizes for FmpCopy are incompatible.

-235 Breakflag detected
An FMP routine detected a break sent by the BR command.

-236 Reserved for superuser
Normal user used a command reserved for the superuser.

-237 Must not be remote
A file was specified with a remote system name or account in a situation where such names are illegal. This error is reported even if the node specifies (or defaults to) the local system.

-238 Illegal program file
The file named is illegal because:

- It is not a program file.
- It accesses system entry points outside the table in %VCTR and is being RP'd to a system other than the one for which it is linked.
- It was linked with an incompatible version of %VCTR.

-239 Program name exists
Cannot RP program with that name because another program already has it. OF the old program with the ID parameter, or choose another name for the new program.

-240 through 241 NOT USED.

-242 Disc I/O failed
D.RTR tried to access an LU that was down, or an EXEC error occurred on an FMP disc access.

-243 Parameter error
An actual parameter has an unreasonable value.

-244 Mapping error
An error occurred while a VMA file routine was mapping VMA.

-245 NOT USED.

-246 System common changed
Tried to RP a program that defines system common differently than it is defined on the current system.

-250 D.ERR not available
The system program D.ERR (used to generate FMP error messages) cannot be scheduled. It was not RP'd or it has been OF'd. RP D.ERR.

The following error codes reflect errors in DS transparency software.

-300 illegal remote access
Usually means an internal error. Either an invalid connection number was specified, or an invalid request was routed to the DS transparency software.

-301 too many remote connections
No more than 64 files can be open at remote systems at any one time. Each open file requires a connection. You can reclaim connections by closing files.

-302 no such node
The local system does not know anything about the node number or the name specified. It may not be in the NRV.

-303 session limit exceeded
Can't log on the remote system because too many other sessions are already logged on.

-304 no such account
No user has that name.

-305 bad password
The correct password was not supplied.

-306 cannot access account
Some logon error occurred that was not one of the above three.

-308 connection broken
The remote system monitor TRFAS was restarted since the connection was open.

The following errors are reported by DS software; see the DS manuals for more details.

-310 DS is not initialized [DS00]
DS has not been started with DINIT

-311 DS link is not connected [DS01]
Hardware problem.

-312 Remote system doesn't respond [DS05]
Other system is probably down, or not running DS.

-313 No TRFAS at remote system [DS06]
Remote monitor TRFAS is not RP'd at remote system.

-315 DS error DSXX(X), node YY
Something happened not included in the above. The DS error code is reported.

Appendix B
Exception Condition Handling

There are situations that do not occur often but when they do, the cause and the corrective action may not be apparent. These are described in this appendix.

Unusual File Access Errors

A number of unusual problems can occur when using CI files. The following paragraphs discuss some of these problems.

Non-Standard File Names

Non-standard characters are those other than A-Z, a-z, 0-9 (not as the first character), and underscore. The CI file system assigns specific meanings to the following characters: period, slash, and colon. (However, FMGR files have different file name requirements as discussed below.) In addition, the commands which allow masks assign meaning to the minus sign and at sign (@), and the DS transparency routines assign meaning to the characters left bracket ([) and greater-than (>) sign (except as the first character of a name). Do not use any of these characters in file names; a file called @.@ will not be recognized as such by DL, CO, and PU.

Often, FMGR files have special characters in them. This was a naming convention, since only six characters were allowed in names. CI can accommodate FMGR files from the normal CI file commands if the files do not have any special characters. It is recommended that all FMGR files with special characters be changed to conform to the CI file system convention as mentioned above, by renaming them from FMGR.

File Not Found

The CI file system follows a specific file search sequence if the directory (or subdirectory) is not given. If a file is not within the default search path, it may not be found. It is easy to forget the directory (subdirectory) search sequence. It is best to specify the subdirectory/directory name with the file to prevent accessing the wrong file or receive the "No such file" message. A quick reference to the CI file system search sequence is given below:

1. When a directory name is specified with a file, such as FILE::XX, the named directory is searched. If the program can also accommodate FMGR files, then FMGR discs will also be searched to find such a directory. If the program only understands FMGR files (such as FMGR, FC and other subsystems), then only FMGR disc directories will be searched.

2. If no name is specified, then the working directory will be searched, if there is one. If there is no working directory, all FMGR files are searched in the order they appear in the cartridge list. The directory name 0, as in FILE::0, is used to search all FMGR cartridges regardless of whether there is a working directory.

3. Some programs, such as CI and LINK, will look in special directories if the file is specified without a directory name. Programs using FMGR subroutines will receive the "No such file" message when accessing files in the CI file system.

Directory Name and FMGR Cartridge Reference

An untimely CRDIR or MC command or tape restore can cause the same name to be used simultaneously as a global directory and as a FMGR cartridge name. This situation seldom occurs but can be confusing when it does happen.

Files on a FMGR cartridge called XX are referred to as <filename>::XX. The same is true for global directories, those directories (not subdirectories) that are at the top of the CI file structure. It is possible for XX to refer to both a FMGR cartridge and a global directory at the same time. In this case, if you are running FMGR or FC, the FMGR cartridge XX is used; CI and other programs will use directory XX. It is recommended that you use unique names for global directories and FMGR cartridge references.

Unable to Open File or Create Directory

If you try to open a file or create a global directory and receive a message "Too many open files" or "Too many directories", it is caused by program D.RTR running out of table space. D.RTR keeps information about open files and mounted global directories in the table space at the end of its code space. The amount of this table space is determined by the SZ command, and is fixed while D.RTR is running. If you have a large system, this space may be used up; D.RTR will be unable to open any file, create any global directory, or mount any volume until the situation is cleared. You can clear this situation by closing files, or by removing global directories via purge, rename or dismount commands; but it is best to prevent it by keeping D.RTR sized to 32 pages and by limiting the number of open files and global directories to no more than 600 total. (300 of each, 100/500, etc.)

OWNER, PROT or WD Command Failures

If an account is purged and subsequently recreated, the directory/file ownership for that user must be reassigned. Purging an account removes the user ID number used in the session environment and this number is used for directory/file ownership. This also applies to moving a disc volume from one system to another system with a different set of user accounts, causing incorrect ownership. This problem can also be corrected with the OWNER command.

Disc Volume Full

There are limits to how much data will fit on a disc. Flexible discs, such as mini- and micro-floppies, have limited storage space; other discs have much more space available. When there is no room available to create or extend a file on a particular volume, the message "Ran out of disc space" is displayed.

This message means that there is not enough contiguous, free disc space on a particular volume. There may adequate free space available on other volumes; or there may be enough free space in the disc but they are smaller than the needed size on that volume. Files (and directories) never cross volumes, and they are not broken up into smaller pieces to fit into smaller areas.

There are several courses of action available. You may be able to create the file in a directory on another volume which has space. The FREES program will show how much space is available on a volume. If you have several volumes, you may be able to choose one with more free space. You may also be able to create the file with a smaller number of contiguous blocks; this will help if the free space is fragmented into smaller sections. (The CI CO command tries to create contiguous sequential files if you do not specify a size.)

If the problem still exists, you may purge the unneeded files to reclaim disc space. The free space is automatically reclaimed (except on FMGR cartridges which require packing). Purging a large file may free enough space, or purging several small files may be sufficient. Some helpful hints are given below.

1. Archive and purge files that have not been used in a long time. The command DL @.@ +A will sort by time of last access to help you decide.

2. Purge files which are close to each other; "DL @.@ +L" can be used to sort by disc location of the main file.

3. Purge larger files to get more space; "DL @.@ -S" reverse sorts by size.

4. Purge any temporary files, left by LINK, Macro, or other program, with a PU @.@.T command. (Open files are not purged.)

The E qualifier can be used with these commands to search everywhere; this may or may not be useful, depending on how many volumes and files you have.

If there is not enough contiguous free space available, the FPACK program can be used to rearrange the files on the disc to produce more contiguous space. Program FPACK can be run concurrently with other operations.

For a permanent solution, the system must be regenerated to change the way disc space is allocated. Having fewer, larger volumes will reduce the number of times that a volume will be full, though it will make the problem worse if and when it does happen. There are no disc space limits for individual users, only for volumes; you may want to place directories on particular volumes to keep them from competing with each other for disc space.

For FMGR cartridges, disc space is generally not reused when files are purged. The FMGR PK command can be used to reclaim space; files on that cartridge cannot be used while it is being packed. FPACK cannot be used on FMGR volumes.

Disc Volume Dismounted

One or more disc volumes may be dismounted or not mounted during system initialization. All directories on a dismounted volume are inaccessible until it is mounted again. Accessing files in this case will display the message "No such directory".

If you get such errors, do not compound the problem by creating the missing directory on another volume. When you do mount the dismounted volume, you will get duplicate directory errors (see the MC command for details). You should mount the missing volume to fix the problem.

A disc volume cannot be dismounted if it has files that are open, restored programs, or working directories. Thus your more important volumes will probably not be easily dismountable, while less important volumes will be.

Parity Errors

High-speed computer memory is subject to intermittent errors. They are not very common, but can be bad when they occur. The CI system detects these errors through parity checking, or corrects these errors with Error Correcting Circuitry (ECC). If an error occurs which cannot be corrected by hardware, the CI system handles the error.

If the error occurs in memory occupied by the operating system, by operating system tables, or by the program being swapped, the system is halted. This is an unrecoverable error. If the error occurs in a program, the system aborts the program and reports an error in much the same way as it does for other types of aborting errors, such as memory protect violations. Other programs continue to run, and the offending program can be rerun later, usually without error.

Most parity errors are not reproducible and they are known as soft errors. Some errors are repeatable and when one of these occurs, the system blocks off the offending page (or partition if it is a reserved partition), making that page unavailable for use. This will keep other programs from using the bad memory location.

Power-Fail

If your system is equipped with a battery backup card, it will be able to recover from power failures lasting up to several minutes. Battery backup maintains the state of memory, allowing the system to recover where it left off when power failed. The following paragraphs explain what happens during power-fail handling.

The status of programs is maintained during power failures. All data in memory will be intact, although the program will not run while power is off. Power to the I/O cards and peripherals is not maintained, so they may be in an indeterminate state. The operating system notifies all I/O drivers when a power failure occurs. This allows the drivers to reset the I/O cards, along with any other processing the peripheral may require.

Appendix C
Converting FMGR File Calls

This appendix describes a step-by-step procedure to convert FMP calls used in the FMGR file system environment to CI calls for use in the CI file system environment. The conversion procedures have been written to assist in converting programs that may be unfamiliar to the user.

General Considerations

File system calls usually make up a small percentage of a program, so the conversion effort is minimal since in most cases the program logic should not have to be changed.

Many of the FMGR calls will still work although it is recommended that programs be converted to allow full usage of the enhancements available with the FMP calls.

NOTE

The CI calls do not have any optional parameters. All parameters in these calls must be supplied.

File and Directory Names

File and directory names can contain up to 63 characters, allowing for a full name including all directories and the ASCII versions of type, size, etc. A sample file descriptor is shown below.

```
/population/cities/california/sanjose.txt:::4:24  (48 characters)
```

File names should be stored in 32-word character buffers if they are supplied as input to the program. This ensures consistency between programs. Since names are passed as character strings, it is possible to use a smaller buffer for file names which are embedded in the program. CI calls work with unparsed names, so the 32-word buffer replaces the 10-word namr used by FMGR.

Global directory names contain up to 16 characters, and can be stored in 8-word character buffers. A subdirectory is treated as part of the file name by the supplied parsing routines. The global directory name can be specified as a prefix, as in the following example:

```
SOURCE/CMDS::USER:3   or   /USER/SOURCE/CMDS:::3
```

Constructs such as /FILE::DIR produce undefined results.

The directory name can appear in either of two places: to the left of any subdirectories or after two colons to the right of the file name. Use the following conventions to determine where to print the directory name:

- If no subdirectories are specified, print the directory name after the two colons, as in GRIDLOCK.RUN::PROGRAMS.

- If one or more subdirectories are specified, print the directory name as a prefix to the subdirectory name, as in FAMILY/GENUS/SPECIES.TXT.

Use file names in FMP calls after the file is opened since many of the FMP calls work with file names. File names are also useful in reporting errors.

Namr Calls and Strings

Namr calls that parse file names need to be replaced, but be careful not to change namr calls used for different purposes. Namr calls which are used only to set up calls to open, create and purge can be removed, as the new equivalents of these calls do not require parsed file names. Calls which break apart file names for purposes of examining individual components can be replaced with a call to FmpParseName in most cases. FmpParseName does not distinquish subfield types and does not parse up to a comma the way that Namr does.

FmpParseName does not completely replace Namr. Other useful routines include: SplitString, which divides a character string at a blank or comma, and DecimalToInt, which converts a character string to a single integer. Fparm does runstring parsing, returning the file name in a runstring as separate character variables. (Fparm is not available to Pascal users.) Descriptions of these routines can be found in the RTE-6/VM Relocatable Libraries Manual.

Examples:

Here is an example of code which opens two files whose names are passed in the runstring:

```
call getst(buffer,-80,len)
start = 1
if (namr(pbuf,buffer,len,start) .lt. 0) goto 900
type1 = open(dcb1,err,pbuf,0,pbuf(5),pbuf(6))
if (err .lt. 0) goto 920
if (namr(pbuf,buffer,len,start) .lt. 0) goto 900
type2 = open(dcb2,err,pbuf,0,pbuf(5),pbuf(6))
if (err .lt. 0) goto 920
```

This can be replaced by:

```
file1 = ' '
file2 = ' '
call fparm(file1,file2)
if (file1 .eq. ' ' .or. file2 .eq ' ') goto 900
type1 = fmpopen(dcb1,err,file1,'ro',1)
if (err .lt. 0) goto 920
type2 = fmpopen(dcb2,err,file2,'ro',1)
if (err .lt. 0) goto 920
```

Note that Namr was not used.

The next example shows a sequence without character strings.
It illustrates constructing string descriptors, which are the
double integer (integer*4) variables in the following example.
The function STRDSC takes parameters of buffer, starting character
and number of characters, and returns a string descriptor.
Here it is used to create string descriptors for the file name
and option strings (a constant 'ROS'):

```
integer*4 strdsc,string,file1,file2,options
call getst(buffer,-80,len)
string=strdsc(buffer,1,len)
file1=strdsc(buffer1,1,64)
file2=strdsc(buffer2,1,64)
call splitstring(string,file1,string)
call splitstring(string,file2,string)
if (blankstring(file1) .ne. 0 .or. blankstring(file2) .ne. 0)) goto 900
options = strdsc(3hROS,1,3)
type1 = fmpopen(dcb1,err,file1,options,1)
if (err .lt. 0) goto 920
type2 = fmpopen(dcb2,err,file2,options,1)
if (err .lt. 0) goto 920
```

String descriptors describe strings by identifying where they can be found and how big they are. Once a string descriptor is set up, it can be used indefinitely. The buffer it points to can be changed through the string descriptor or through direct changes. In the above example, 'splitstring' changes the referenced buffer, and 'blankstring' tests for an all blank string.

The file system assigns default values for type and size when the file is created. The following example shows how the type and size values can be changed. In the example, the code sequence constructs the name of a debug file from the name of a type 6 file according to the following rule: if the type 6 file name has a .RUN type extension, then create a file with the same name and a .DBG extension; otherwise create a file with the same name but insert an 'at' sign (@) in front of it, because this is a FMGR file. Make the file type 1, block size 96:

```
character pname*64, name*64, dir*16, typex*4, ds*64

call fmpparsename(pname,name,typex,sc,dir,d,d,d,ds)
if (typex .eq. 'RUN') then
    call fmpbuildname(pname,name,'DBG',sc,dir,1,96,0,ds)
else
    call fmpbuildname(pname,'@'//name,typex,sc,dir,1,96,0,ds)
endif
```

Open and Openf Calls

All OPEN and OPENF calls are replaced by FmpOpen calls. Handling of file name parsing and character string is the same as previously described. In addition, be aware of how options and buffer sizes are specified. For example, the FMGR call:

```
type = open(dcb,err,pbuf,0,pbuf(5),pbuf(6),256)
```

specifies exclusive open for reading and writing (assuming the security code matches), with no other unusual options. It uses a 256-word DCB buffer, so the DCB should be declared as 256+16=272 words.

To get the same effect with CI calls, the call would be:

```
type = fmpopen(dcb,err,name,'rwo',2)
```

Note that character options 'rwo' have been specified. Reading and writing are specified by 'rw'. The 'o' option means it is allowed to open an FMGR file, but not to create a new one. (This is discussed more under the section describing the CREAT call.) Other options available and their octal equivalents in the option word of the OPEN call are:

```
  1: shared access: 's'
  2: update mode: 'u'
  4: force to type 1: 'f'
 10: supply subfunction: no equivalent, see FmpSetIoOptions
 20: (not defined)
 40: permit extents: 'x'
```

The option word must be specified in a CI call. In converting an FMGR call, start with 'rwo', then add the other options to match the FMGR call options. For example, an option word 45B (open, permitting type 1 and type 2 extents, forced to type 1 and shared) would be option word 'rwoxfs'. The option characters can come in any order. If it is known that the file will be used only for reading or only for writing, omit the 'w' or 'r' respectively. Use the shared option only for reading files; do not use it for writing without providing synchronization.

Note that the buffer size is specified in blocks, rather than in words. The buffer size needed is the OPEN buffer size divided by 128:

```
type = fmpopen(dcb,err,name,options,256/128)
```

The buffer size parameter must be supplied; if the FMGR call didn't supply a buffer size, use a value of 1.

FmpOpen call accepts a logical unit number as in the OPENF call, but the logical unit number must be a string. For example,

```
type = fmpopen(dcb,err,'6','wo',1)
```

is correct, but

```
type = fmpopen(dcb,err,6,'wo',1)
```

does not work, because the logical unit number is an integer, not an ASCII string. If the logical unit is non-interactive, FmpOpen will try a logical unit lock with wait unless the file open is shared.

Note that CI files can be opened to a large number of programs (more than 7), but there must be room in D.RTR's internal table for the open flag. If there is not enough room, the open will fail. One program can have the same file open several times (if file is shared); this is different from how it used to be when a file is only open to a program once.

READF and WRITF Calls

For sequential files, (type 3 and above, and type 0), READF calls are replaced by FmpRead calls, and WRITF calls are replaced by FmpWrite calls. They are similar to the FMGR calls, except that lengths are passed in and returned as byte lengths, not word lengths. The length read is returned only as a function value, so calling FmpRead as a subroutine will probably not produce the desired results.

For example, the following FMGR call sequence

```
call readf(dcb1,err,buffer,128,len)
if (err .lt. 0) goto 900
call writf(dcb2,err,buffer,len)
if (err .lt. 0) goto 910
```

is replaced by:

```
len = fmpread(dcb1,err,buffer,256)
if (err .lt. 0) goto 900
call fmpwrite(dcb2,err,buffer,len)
if (err .lt. 0) goto 910
```

Now len is in bytes. If the program is expecting to use words, you can either change the program to deal with byte lengths (including odd byte lengths), or you can convert len to words:

```
if (len .ne. -1) len = (len+1)/2
```

End of file is reported as err = 0, len = -1. Do not try to use FmpWrite with a length of -1 to write an explicit end of file, as this will write 0 bytes (see below).

For random access files (type 1 and 2), READF and WRITF calls are converted to an FmpPosition call followed by an FmpRead or FmpWrite call. The straightforward way to do this is to position via (double integer) record number; this is requested by using an internal position parameter of (double integer) -1. (Refer to the FmpSetPosition description for details.)

For example, the following code

```
call readf(dcb1,err,buffer,len,dummy,rrec)
if (err .lt. 0) goto 900
call writf(dcb2,err,buffer,len,wrec)
if (err .lt. 0) goto 910
```

is replaced by:

```
integer*4 drec

drec = rrec
call fmpsetposition(dcb1,err,drec,-1J)
if (err .lt. 0) goto 900
call fmpread(dcb1,err,buffer,len*2)
if (err .lt. 0) goto 900
drec = wrec
call fmpsetposition(dcb2,err,drec,-1J)
if (err .lt. 0) goto 910
call fmpwrite(dcb2,err,buffer,len*2)
if (err .lt. 0) goto 910
```

Be careful not to pass single integers to FmpSetPosition. A called subroutine cannot determine what kind of integer was passed, so FmpSetPosition will use the single integer as the upper half of a double integer.

CLOSE Calls

Non-truncating calls to CLOSE can be replaced by calls to FmpClose:

```
call close(dcb)   =>    call fmpclose(dcb,err)
```

Pass the error parameter, even if no error can occur. FmpClose stores a value through the error parameter.

Truncating CLOSE requires two or three calls, depending on whether or not the user knows the truncation size. The sequence to truncate a file at the current position used to be:

```
call locf(dcb,err,rec,block,offset,size)
if (err .lt. 0) goto 900
tblocks = (size/2) - (block+1)
call close(dcb,err,tblocks)
if (err .lt. 0) goto 900
```

Now it is:

```
integer*4 record, position, newsize
call fmpposition(dcb,err,record,position)
if (err .lt. 0) goto 900
newsize = (position+127)/128  for type 3, (position/128)+1
call fmptruncate(dcb,err,newsize)
if (err .lt. 0) goto 900
call fmpclose(dcb,err)
if (err .lt. 0) goto 900
```

Note that the CLOSE call specified the number of blocks to truncate, while the converted code specifies the desired file size. The new sequence will truncate extra extents, which was not possible before. All sizes are double integers. There is no call provided for this sequence because it is not common.

Note that truncating to zero size does not purge the file. It leaves a one block file.

CREAT and CRETS Calls

All CREAT and CRETS calls are replaced by FmpOpen calls which specify the 'c' option, meaning it is allowed to create the file. These calls are similar to the OPEN and OPENF call. Refer to the description of OPEN and OPENF for conversion details. Additional size, type and record length information is passed as ASCII, appended to the name; FmpBuildName is useful for creating ASCII strings.

Any options used in an OPEN call can be specified when creating a file. CREAT sets up default options of nonshared update mode, so to create the equivalent code sequence, use the string 'rwcu.'

For example,

```
call creat(dcb,err,pbuf,pbuf(8),pbuf(7),pbuf(5),pbuf(6))
if (err .lt. 0) goto 900
```

is replaced by:

```
call fmpopen(dcb,err,name,'rwcu',1)
```

This will give an error -2 if the file exists. Specifying both 'o' and 'c' will open the existing file, or create a new one if necessary. Note that this sequence followed by opens can be replaced by a single FmpOpen call:

```
call creat(dcb,err,pbuf,24,3,pbuf(5),pbuf(6),256)
if (err .eq. -2) then
   call open(dcb,err,pbuf,0,pbuf(5),pbuf(6),256)
endif
if (err .lt. 0) goto 900
```

is replaced by:

```
call fmpopen(dcb,err,name,'rwoc',2)
if (err .lt. 0) goto 900
```

Handling of scratch files consists of creating a name which is unique and a bit which indicates that this file is not important. The program that creates the scratch file must purge it before exiting. The file system does not automatically purge scratch files, although a wildcard purge of all scratch files can be specified. This eases the problem of having scratch files disappear when they are closed briefly.

To create an extendible type 1 scratch file with a starting size of 24 blocks, the FMGR calling sequence

```
call crets(dcb,err,0,name,24J,1,sc,cr)
if (err .lt. 0) goto 900
call open(dcb,err,name,40b,sc,cr)
if (err .lt. 0) goto 900
```

is replaced by:

```
call fmpuniquename('TEMP',name)
call fmpopen(dcb,err,name//':::1:24','rwctx',1)
if (err .lt. 0) goto 900
```

The 't' option specifies this is a scratch file. Note that this file goes on the working directory. This only causes a problem if the working directory is currently on a small or slow disc, when a larger or faster disc is available elsewhere. One possible solution is to create the file on directory SCRATCH or some such special name, then try again on the working directory if the special directory does not exist.

In this example the unique name has a prefix 'TEMP'. This is of no special significance, except that some prefix must be supplied to differentiate the name from a number. If there is a chance the scratch file will go on an FMGR cartridge, then the prefix should be short (one character) to keep from getting duplicate six-character names. In any case, the name must be known in order to purge the file.

APOSN, LOCF and POSNT Calls

File positioning is also discussed in the section covering random access READF and WRITF calls. APOSN and LOCF use internal file pointers, while POSNT positions by record number. These functions are performed with FmpPosition and FmpSetPosition for CI files.

Two position pointers are maintained for open disc files, a record number and an internal file position. The internal file position is the word offset from the first word of the file. To record the current record number and internal file position, use FmpPosition. Note that it always returns double integer values, even if single integers were passed. For example, the LOCF call

```
call locf(dcb,err,record,block,offset)
if (err .lt. 0) goto 900
```

is replaced with

```
integer*4 drecord, dposition
call fmpposition(dcb,err,drecord,dposition)
if (err .lt. 0) goto 900
```

The new internal position value is related to the previous value: position = block * 128 + offset.

Use caution when changing LOCF calls. They contain much information, and it is not always easy to tell what is used and what is not. FmpPosition only returns file position. Other LOCF information can be obtained using the FmpSize and FmpEof calls. The FmpSize call returns the total size of the file in blocks, not the size of the main part of the file in sectors. The FmpEof call tells how much of the file is being used. There is no CI call to return the logical unit of a file, because the logical unit cannot be used in place of the directory name. The FmpRecordLength call returns file record length; FmpOpen returns file type when it opens the file.

To restore file position to a place recorded with APOSN, use FmpSetPosition. For example:

```
call aposn(dcb,err,record,block,offset)
if (err .lt. 0) goto 900
```

is replaced by:

```
integer*4 drecord, dposition
call fmpsetposition(dcb,err,drecord,dposition)
if (err .lt. 0) goto 900
```

This works for any type disc file. FmpSetPosition knows to use the internal position recorded by FmpPosition because the passed position is a non-negative value. If the position is negative, it is ignored and positioning is done by record number (see below.) The record number parameter is only used to set up the record number in the DCB for use later by calls which position by record number.

FmpSetPosition is also used to position files by record number. Positioning type 1 and 2 files has already been discussed under READF and WRITF. Positioning type 0 and type 3 and above files has been described in the FmpPosition description in Chapter 6. POSNT can position to an absolute record number, or to a record number relative to the current position. FmpSetPosition always positions to an absolute record number; however, relative positioning can be achieved by first using FmpPosition to see where you are, then adding the offset to get the absolute record number. (FmpSetPosition always positions relative to the current record number in the DCB, so if this is wrong you will not end up at the right absolute record number.) Remember that positioning sequential files by record number can be very slow.

For example, to position to absolute record 100, then skip backward 10 records, using POSNT:

```
     call posnt(dcb,err,100,1)
     if (err .lt. 0) goto 900
              .
              .
              .
     call posnt(dcb,err,-10,0)
     if (err .lt. 0) goto 900
```

The above sequence can be replaced by:

```
     integer*4 drecord, dposition
     call fmpsetposition(dcb,err,100J,-1J)
     if (err .lt. 0) goto 900
              .
              .
              .
     call fmpposition(dcb,err,drecord,dposition)
     if (err .lt. 0) goto 900
     call fmpsetposition(dcb,err,drecord-10,-1J)
     if (err .lt. 0) goto 900
```

The -1J parameter passed as the file position indicates that only the record number is to be used for positioning, as with type 1 and 2 files.

PURGE and NAMF Calls

PURGE calls are replaced by FmpPurge calls, and NAMF calls are replaced by FmpRename calls. The CI calls do not work if the file is open to anyone, including the caller, so the file should be closed first. These calls do not require the caller to pass in a DCB.

The following PURGE call,

```
    call purge(dcb,err,pbuf,pbuf(5),pbuf(6))
    if (err .lt. 0) goto 900
```

is replaced by:

```
    call fmpclose(dcb,err)
    err = fmppurge(name)
    if (err .lt. 0) goto 900
```

The following NAMF call,

```
    call namf(dcb,err,pbuf,newname,pbuf(5),pbuf(6))
    if (err .lt. 0) goto 900
```

is replaced by:

```
    call fmpclose(dcb,err)
    err = fmprename(oldname,err1,newname,err2)
    if (err .lt. 0) goto 900
```

Extended Calls

Extended calls (ones that start with E, i.e., EREAD, EWRIT, ECREA) are replaced in the same way as their non-extended equivalents. These calls work with large files as a standard feature.

The creation of a file larger than 32767 blocks is slightly complicated. The user must pass in an ASCII file size which is the negative number of 128-block "chunks" in the file, so that a 50000 block file would be represented as -(50000+127)/128 = FOO:::-391. This will really create a 50048 block file. Maximum file size is 32767 * 128 blocks, which is about 4 million blocks or 1 billion bytes.

Other Calls

CI calls that perform the functions done by Rwndf, Post and Fcont are FmpRewind, FmpPost and FmpControl, respectively. Their use is not illustrated here, but is described in Chapter 6.

Accessing FMGR Files

This section describes what happens when a CI call refers to a FMGR file, which provides the same level of service as is obtained with FMGR calls referring to FMGR files. The caller can open, create, or purge files on FMGR disc cartridges. This is straightforward if the cartridge is specified, and if there is no new directory with this name. The cartridge can be specified as +CRN or -LU. The following paragraphs discuss the cases where the cartridge is not specified.

If there is a CI directory with the same name as an FMGR cartridge CRN, then that cartridge cannot be accessed via the CI calls, although it can be with FMGR calls. (CI calls first check CI directories, while FMGR calls first check disc cartridges.) In general, it is confusing to have a CI directory with the same name as a disc cartridge, so it is not recommended (although it is allowed).

If the directory is not specified, e.g., FOO or FOO:::3, and the user has a working directory, only that directory is searched. If the directory is explicitly specified as 0, FOO::0, or FOO::0:3, then all of the FMGR disc cartridges mounted to this user will be searched. This also applies to the case where there is no working directory and a directory is not specified. This is one way to get a multiple disc search with the CI calls, and it only searches FMGR disc cartridges.

Calls which specify a file name only work with FMGR files if the information is available in the FMGR directory. Thus, a user can get the name of an FMGR file, but cannot get the timestamps or position of end-of-file. In the latter cases, the FMGR cartridges are not even searched, even if a disc cartridge name is specified. A summary is given below.

Calls that pass file names and work with FMGR files:

FmpOpen, FmpPurge, FmpRename, FmpSize, FmpAccess

Calls that pass file names but do not work with FMGR files:

FmpCreateTime, FmpAccessTime, FmpUpdateTime, FmpEof,
FmpCreateDir, FmpSetWorkingDir, FmpUnpurge,
FmpSetOwner, FmpSetAccess

Calls that do not pass file names and do not work with FMGR files:

FmpSetDirInfo, FmpOpenFiles

Other calls that do not pass file names work with FMGR files.

NOTE

If the directory name is found on a disc cartridge, then the FMGR rules for parsing names apply. Dots and slashes in names are not significant on FMGR directories. The name is truncated to six characters.

Accesses to FMGR directories follow all rules for the FMGR file system, such as that for open flags and extent creation. The same protection checks (security code, etc.) are made, although it is not guaranteed that all invalid requests will be caught (such as illegal characters in file names.) FMGR file system error codes are returned when appropriate.

Calls which specify a DCB work regardless of whether the file is FMGR or CI, including read, write, position, etc. This includes files with extents, and files with odd byte length records.

Standard Type Extensions

File type extensions are used to replace the special characters used in FMGR to designate a group of files, e.g., % for relocatables and & for source, etc. The following is a list of the standard file type extensions used in the CI file system.

.ABS	absolute file
.CMD	transfer file
.DAT	data
.DBG	debug information file
.DIR	directory
.FTN	FORTRAN source
.LIB	indexed library
.LOD	loader command file
.LST	list file
.MAC	Macro source
.MAP	load map
.PAS	Pascal source
.REL	relocatable
.RUN	loaded program
.STK	command stack file
.TXT	text (reports, etc.)

INDEX

A

B

C

D

E

F

G

H

I - J - K

L

M

N

O

P

Q

R

S

T

U

V

W

X - Y - Z

Part No. 92084-90036
Printed in U.S.A. January, 1985
E0185

HEWLETT PACKARD