HEWLETT PACKARD

RTE-A

User's Manual

Software Services and Technology Division
11000 Wolfe Road
Cupertino, CA 95014-9804

Manual Part No. 92077-90002
E0495

Printed in U.S.A. April 1995
Eighth Edition

NOTICE

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

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THE 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.

RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARs 252.227.7013.

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 Manual Numbering File. (The Manual Numbering File is included with your software. It consists of an "M" followed by a five digit product number.)

Second Edition	Jun 1983	
Update 1	Dec 1983	Command stack enhancement
Reprint	Dec 1983	Update 1 incorporated
Third Edition	Jan 1985	CI enhancements
Update 1	Jan 1986	
Reprint	Jan 1986	Update 1 incorporated
Fourth Edition	Aug 1987	Rev. 5000 (Software Update 5.0)
Fifth Edition	Jan 1989	Rev. 5010 (Software Update 5.1)
Update 1	Jul 1990	Rev. 5020 (Software Update 5.2)
Sixth Edition	Dec 1992	Rev. 6000 (Software Update 6.0)
Seventh Edition	Nov 1993	Rev. 6100 (Software Update 6.1)
Eighth Edition	Apr 1995	Rev. 6200 (Software Update 6.2)

Preface

This manual introduces the interactive features of the RTE-A Operating System and shows how to communicate with the system from the terminal keyboard. The commands described are those provided by the Command Interpreter (CI) program, which is designed to simplify interactive operations for RTE-A users. To run other programs such as the editor or LINK interactively, consult the manuals that describe those programs.

RTE-A also provides an interactive user program called FMGR, which maintains files previously created with FMGR. Appendix B of this manual contains a full description of the FMGR program.

This manual also includes information on the HP 92078A Virtual Code+ System Extension Package (VC+). Information related to this product is indicated as optional; it is only needed if your system uses VC+.

You should be familiar with the *RTE-A Getting Started Manual,* part number 92077-90039*,* supplied with the RTE-A Operating System. It is assumed that RTE-A has been successfully booted up and is operating. All users of this manual must be familiar with the display terminal being used in the system.

How This Manual Is Organized

Chapter 1	Describes the features of the RTE-A Operating System and introduces the basic concepts and terms used throughout this manual.
Chapter 2	Describes how to obtain system information and how to control certain user-level system operations.
Chapter 3	Describes files, their associated attributes, and how to manipulate files.
Chapter 4	Describes programs, their associated attributes, and how to control programs.
Chapter 5	Describes the multiuser operations provided with VC+.
Chapter 6	Contains descriptions of all CI commands and utility programs. The descriptions are given in alphabetical order.
Chapter 7	Describes the methods of editing CI command strings.
Chapter 8	Describes exception condition handling.
Appendix A	Contains error messages and error codes that may be encountered during interactive operations. Operator error message formats are also described in this appendix.
Appendix B	Describes the FMGR commands for disk/file manipulation, transferring files, and FMGR scheduling.
Appendix C	Describes the CS/80 Exerciser Utility (EXER) used to diagnose and troubleshoot CS/80 disk drives on HP 1000 systems.

Conventions Used in This Manual

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

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

Certain commands and command parameters are applicable only to systems using VC+. These are indicated by "(VC+ Only)" in the heading or in the descriptive comment column.

The command syntax conventions are as follows:

Convention	Meaning

Uppercase characters	Capital letters indicate the exact characters required. However, CI accepts lowercase input. For example, the command syntax for the AS command is:
	
	`AS` *prog* <*partition number*> `[C|D]`
	
	and the actual sample entry can be:
	
	`CI>as testprogram 2 c`

[ ]	Optional parameters are shown in square brackets. If you omit a parameter, use a comma as a placeholder before specifying additional parameters, unless the omitted parameter(s) begins with a + or − character.
|	Separate alternate choices by a vertical bar (|). The "C|D" shown above indicates either C or D can be entered.
, or blank	Use a comma or a space as a delimiter between commands and parameters. Blank spaces are used throughout this manual in all syntax strings.
lowercase and < >	Italicized lowercase letters represent user-supplied variables. In long descriptive phrases, the variable may be enclosed in angle brackets for clarity, for example, <*partition number*>.

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. For VC+ systems, session identification is optional. Examples of program names:

```
A
PROGA
ADVEN
TIMER
LRUN/3    (Session ID is 3)
```

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, for example, in the TO command description in Chapter 6. LU 0 is the bit bucket, a non-existent device to which unwanted data can be dumped.

file — File descriptor. Refer to the CR command description in Chapter 6 for a full description of the file descriptor. It can be accepted in any of the following formats:

Standard:	*/dir/subdir/filename*:::*type*:*size*:*rlen*
Combined:	*subdir/filename*::*dir*:*type*:*size*:*rlen*
FMGR:	*filename*:*sc*:*crn*:*type*:*size*:*rlen*

mask — Mask field. May be the wildcard characters in the filename parameter (- and @) or a mask qualifier appended to the filename 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. Refer to the RU command description in Chapter 6 for more information.

pram — One parameter is allowed.

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

Table of Contents

Chapter 1
Introduction

Using RTE-A 1-1
Command Interpreter Features 1-2
Introduction to Programs 1-4
Introduction to Files 1-4
System Information 1-5
CI Termination 1-5
When CI is Not Available 1-5

Chapter 2
Using the System

Getting Help 2-3
Using the Command Stack 2-4
Obtaining System Status 2-6
    Display Program Status 2-6
    Display Memory Usage 2-8
    Display I/O Configuration 2-8
Controlling Devices 2-10
Changing I/O Device Attributes 2-11
    How to Bring Up a Device 2-11
    Changing Timeout Values 2-12
Displaying System Time 2-13
Executing a Command File 2-13
Positional Variables 2-14
User-Defined Variables 2-15
Environment Variables (VC+ Only) 2-16
Predefined and Other Variables 2-16
Tilde (~) Substitution 2-21
Nesting Command Files 2-21
String Processing and Positional Variables Parsing 2-22
Quoting 2-23
Multiple Commands per Line 2-24
Return Status 2-24
Execution Control Structures 2-25
Timeout/Logoff Function 2-25
Spooling (VC+ Only) 2-26
Error Logging (VC+ Only) 2-26
System Setup Utilities 2-27

Chapter 3
Manipulating Files

File Properties 3-1
    File Names 3-1
    Temporary Files 3-2
    I/O Devices Referenced as Files 3-2

File Type Extensions 3-4
File Descriptors 3-5
Directories 3-6
Subdirectories 3-8
Directory Specifiers "." and ".." 3-9
Directory Specifier "#n" 3-10
File Type 3-11
File Size 3-12
Record Length 3-12
File Ownership and Associated Group (VC+ Only) 3-13
Protection (VC+ Only) 3-13
Time Stamps 3-14
File Masks 3-14
Masking and FMGR Files 3-20
Destination File Masks 3-20
File Operations 3-22
Directory Listings 3-22
Listing Files 3-23
Copying Files 3-24
Renaming Files 3-24
Moving Files 3-25
Purging Files 3-25
Unpurging Files 3-26
Creating Empty Files 3-26
Change File Protection (VC+ Only) 3-27
Creating Symbolic Links (VC+ Only) 3-28
Manipulating Directories 3-29
Creating a Directory 3-29
Creating a Subdirectory 3-30
Display/Setup Working Directory 3-30
Moving Directories 3-31
Displaying Directory Owner (VC+ Only) 3-31
Changing Directory Owner and Associated Group (VC+ Only) 3-32
Purging Directories 3-33
Display/Change Directory Protection 3-33
Searching for Files 3-34
Default Search Sequence 3-34
Defining UDSPs (VC+ Only) 3-34
Specifying UDSPs in File Descriptors (VC+ Only) 3-35
Manipulating Volumes 3-36
Mount/Dismount Volumes 3-36
Volume Ownership and Protection 3-37
Listing Volumes 3-37
Initializing Volumes 3-38
File Manipulation Utilities 3-38
File System Utilities 3-38
Transferring Data to and from Devices 3-39
FMGR Files 3-40
DS File Access (DS Only) 3-41
Specifying Remote Files 3-41
Multiuser Remote File Access 3-42
DS File Access Considerations 3-43
Remote File Access Limitations 3-44

Chapter 4
Controlling Programs

Program Identification 4-2
Program Priorities 4-2
Running a Program 4-2
Program Execution 4-3
    Running Programs with Wait 4-3
    Running Programs without Wait 4-4
    Time Scheduling Programs 4-5
Restoring Programs 4-5
    Program ID Segments 4-5
    Prototype ID Segments 4-6
    Program ID Duplicating 4-6
Removing Programs 4-7
Breaking Program Execution 4-7
Suspending a Program 4-8
Resuming Program Execution 4-8
Restarting a Program 4-8
Displaying Program Status 4-8
Changing Program Priorities 4-9
Changing Memory Requirements 4-9
Assigning Partitions 4-10
Changing Virtual Memory Area 4-11
Shared Programs (VC+ Only) 4-11
Changing CDS Program Memory Requirement
(VC+ Only) 4-12
Models of EMA/VMA 4-13

Chapter 5
Multiuser Session Operation (VC+ Only)

Logon and Logoff 5-1
Group Accounts 5-4
User Accounts 5-4
Creating and Modifying Accounts 5-5
Superusers 5-5
Capability Levels 5-5
    Command Capability Levels 5-6
    Program Protection through Capability Levels 5-6
Session Handling 5-7
Identifying Programs 5-8
Running Out of SAM 5-8

Chapter 6
Command Descriptions

Capability Levels for CI Commands 6-1
Precedence Within CI 6-2
? (Help) 6-3
* (Comment) 6-3
AB2MI (Absolute Binary to Memory Image) 6-4
    Break Detection 6-4
    AB2MI Error Messages 6-5

ALIAS (Define/Display Aliases; VC+ Only) 6-6
AS (Assign Partition)* 6-8
ASK (Display a Prompt and Read a Response) 6-9
AT (Set Program Run Time) 6-11
BR (Break Program Execution)* 6-13
CALLM (Merge Text Files for CALLS Utility) 6-14
.Include Directive 6-15
CALLS (Online Help Facility) 6-16
CALLS Catalog File 6-17
CALLS Directives 6-17
Relating Topics to Other Topics 6-18
Index File 6-19
CD (Change Working Directory) 6-20
CL (List Mounted Disks) 6-22
CLOCK (Access A990 Clock Chip) 6-23
CN (Control Device) 6-24
CO (Copy Files) 6-26
CP (Copy Files and Directory Subtrees; VC+ Only) 6-29
CR (Create File) 6-32
CRDIR (Create Directory/Subdirectory) 6-35
CRON (Clock Daemon; VC+ only) 6-37
CRONTAB (User CRONTAB File; VC+ only) 6-38
CSYS (Copy System) 6-40
Operation 6-41
Examples 6-41
Loading CSYS 6-42
Error Messages 6-43
CZ (Display/Modify Code Partition Size; VC+ Only)* 6-44
DC (Dismount Disk Volume) 6-45
DL (Directory List) 6-46
DT (Display/Modify Data Partition Size; VC+ Only)* 6-50
ECHO (Display Parameters at Terminal) 6-51
EX (Exit) 6-52
FOWN (Report File Space by Owner) 6-54
Return Values 6-55
Examples 6-55
FPACK (File System Pack) 6-57
The Packing Process 6-57
Moving Directories 6-59
Moving Subdirectories 6-60
Moving Files 6-61
FPUT (Bootable System Installation) 6-62
FPUT Operation 6-63
FREES (Indicate Free Space on a Volume) 6-64
FSCON (File System Conversion) 6-67
Requirements for Successful Conversion 6-67
The Conversion Process 6-67
File Renaming 6-68
Converted CI Directory Entries 6-68
Error Messages 6-69
FUNCTION (Define a Function; VC+ Only) 6-70
FUNCTIONS (Display Functions; VC+ Only) 6-72
FVERI (File System Verification) 6-73
Error Recovery 6-75
Error Messages 6-76

GO (Resume Suspended Program)* 6-79
GREP, FGREP (Search a File for a Pattern) 6-80
IF-THEN-ELSE-FI (Control Structure) 6-83
IN (Initialize Disk Volume) 6-85
INSTL (Initialize BOOTEX File) 6-86
INSTL Operation 6-88
Error Messages 6-89
IO (Display I/O Configuration and Status) 6-90
IS (Compare Strings or Numbers) 6-95
KTEST (ksh-style Condition Evaluation Command) 6-96
LI (List Files) 6-98
LI Flags 6-98
LI Commands 6-100
LNS (Create Symbolic Link; VC+ Only) 6-103
LS, LL, LSF, LSX (List Directory Contents) 6-105
MC (Mount Disk Volume) 6-108
MERGE (Concatenate Many Files into One) 6-109
METER (Display CPU Usage) 6-111
Sorting and Displaying Process Information 6-112
Example of METER Output 6-113
Loading METER 6-113
MI2AB (Memory Image to Absolute Binary) 6-114
Break Detection 6-114
Error Messages 6-115
MO (Move Files) 6-116
MPACK (File Compacting and Disk Pack) 6-117
MPACK Options 6-117
Compacting and Reporting Options 6-117
Remove Extents from Files (+R) 6-118
Packing Options 6-120
OK to Overlay Data (+OK) 6-121
Enable Visual Mode (+V) 6-121
Logging Option 6-122
Examples 6-123
MV (Move/Rename Files/Directories; VC+ Only) 6-124
NOTIFY (Send a Message to a Terminal) 6-125
Defining Aliases for Notification 6-126
Alternate Logons for a User 6-126
Group Distribution Lists 6-126
Hosts in a DS Network 6-127
Alias Definition Format 6-127
OF (Stop/Remove Program)* 6-128
OLDRE (Extended Record Converter) 6-129
OLDRE Extended Records 6-129
OLDRE Operation 6-129
Translation Results 6-130
Program Restrictions 6-131
Macro 6-131
Pascal 6-131
FORTRAN 6-132
Error Messages 6-132
OWNER (Display/Change Owner; VC+ Only) 6-133
PATH (Display/Modify UDSP; VC+ Only) 6-134
POLL (Polling Function) 6-137
PR (Change Program Priority)* 6-138

PROT (Display/Change Protection; VC+ Only) 6-139
PS (Display Program Status)* 6-141
PU (Purge Files) 6-143
PWD (Display Working Directory) 6-145
RESIZE (Set $LINES/$COLUMNS Variables; VC+ Only) 6-146
RETURN (Return from Command File) 6-147
RM (Remove Files or Directories; VC+ Only) 6-148
RN (Rename File, Directory, or Subdirectory) 6-149
RP (Restore Program File) 6-150
RS (Restart Program) 6-152
RU (Run Program)* 6-153
Base Set Version of the RU Command 6-155
SAM (Show the Status of System Available Memory) 6-156
Running SAM without the AL Parameter 6-156
Running SAM with the AL Parameter 6-157
Returned Values 6-157
Loading SAM 6-158
SCOM (File Comparison) 6-159
F1, F2, BO 6-160
NN 6-160
NH 6-160
NT 6-160
TB 6-160
IB 6-160
Dx 6-160
Cx 6-161
IT 6-161
IC 6-161
ET, ER 6-161
BR, BB 6-161
TC 6-162
The Compare Operation 6-162
Returned Values 6-162
Status Interrogation 6-163
SCOM Examples 6-163
SCOM Error Messages 6-170
SET (Display/Define Variables) 6-171
SPORT (Serial Port Analyzer) 6-173
Including SPORT in a User Program 6-174
Loading SPORT 6-175
SS (Suspend Program)* 6-176
SYSTZ (Set Time Zone and Daylight Savings Time) 6-177
SZ (Display or Modify Program Size)* 6-179
TM (Display or Set System Clock)* 6-180
Base Set Version of the TM Command 6-180
TO (Display or Modify Device Timeout) 6-181
TOUCH (Update File Times; VC+ Only) 6-182
TR (Transfer to Command File) 6-184
UL (Unlock Shareable EMA Partition; VC+ Only)* 6-186
UNALIAS (Delete Alias; VC+ Only) 6-187
UNPU (Unpurge Files) 6-188
UNSET (Delete User-Defined Variable) 6-189
UP (Up a Device)* 6-190
VS (Display or Change VMA Size)* 6-191
WC (Line, Word, Character Count) 6-192

WD (Display or Change Working Directory) 6-194
WH (System Status Reporting) 6-195
WH Options 6-197
WH Option Examples 6-197
WH, CL (Class Table Information) 6-198
WH, D (Show Prototype ID Segments) 6-199
WH, SC (Show Scheduled Programs) 6-199
WH, PA (Memory Partition Listing) 6-199
WH, RN (Display Resource Number Information) 6-200
WH, SH (Display Shareable EMA) 6-201
WH, ST (System Status Information) 6-202
Locking WH in Memory 6-202
WHILE-DO-DONE (Control Structure) 6-203
WHOSD (Report Users; VC+ Only) 6-204
WS (Display or Modify VMA Working Set Size)* 6-206
XQ (Run Program Without Wait)* 6-207
Base Set Version of the XQ Command 6-207

Chapter 7
Command Editing

/ (Command Stack Editor) 7-1
$VISUAL Mode Command Line Editing (VC+ Only) 7-8
Performance Considerations 7-8
EMACS/GMACS Visual Editing Mode 7-9
Cursor Motion Commands 7-10
Deleting, Killing, and Restoring Commands 7-10
Formatting Commands 7-11
Marking Commands 7-11
Searching Commands 7-11
Other EMACS Editing Commands 7-12
VI Visual Editing Mode 7-13
Input Mode Edit Commands 7-13
Control Mode Edit Commands 7-13
Cursor Motion Commands 7-13
Search Commands 7-14
Text Modification Commands 7-15
Other Edit Commands 7-16
CSH Visual Editing Mode 7-17
CMNDO Monitor 7-17

Chapter 8
Exception Condition Handling

Unusual File Access Errors 8-1
Non-Standard File Names 8-1
File Not Found 8-2
Directory Name and FMGR Cartridge Reference 8-2
Unable to Open File or Create Directory 8-2
OWNER, PROT, or WD Command Failures 8-3
Unusual Logon Errors 8-3
Disk Volume Full 8-3
Disk Volume Dismounted 8-5
Clearing Open Files 8-5
Parity Errors 8-6

VCP Interrupt 8-6
Missing System Programs 8-7
The RTE Prompt 8-8
The System Prompt 8-8
Power Failure 8-9

Appendix A
Error Messages

Error Formats A-1
Error Messages and Codes A-1
I/O Errors A-2
Program Abort Errors A-3
Parity Errors A-4
Error Messages A-4
FMP Error Codes A-13
DS Transparency Software A-23
DS/1000 Software Errors A-24
Native Language Support Utilities Errors A-24

Appendix B
File Manager (FMGR)

FMGR Control B-2
List Device B-2
Log Device B-2
Severity Code B-2
FMGR Errors B-2
FMGR Control Commands B-2
Explain Error Codes (??) B-3
?? Command Examples B-3
Display or Change List Device (LL) B-4
LL Command Examples B-4
Change Log Device (LO) B-5
Display or Set Severity Code (SV) B-5
SV Command Examples B-6
Disk Manipulation B-7
Logical vs Physical B-7
Disk Logical Units and LU Numbers B-7
Cartridges and Cartridge Reference Numbers B-8
Configuration of Logical Units/Cartridges B-8
Mounting and Dismounting Cartridges B-9
Cartridge Directory B-10
Cartridge File Directory B-10
Cartridge Initialization B-11
Master Security Code B-12
Re-initializing a Cartridge B-12
Purging Files on a Cartridge During Re-initialization B-13
Packing a File Cartridge B-14
Transferring Files Between Disk Cartridges B-15
Disk Manipulation Commands B-15
List Cartridge Directory (CL) B-16
CL Command Example B-17
Copy Files (CO) B-17
CO Command Options B-18

CO Command Option Examples B-19
CO Command Examples B-20
CO Command Termination B-22
Dismount File Cartridge (DC) B-23
DC Command Error Handling B-23
List File Directory (DL) B-24
DL Command Examples B-25
Initialize File Cartridge (IN) B-27
IN Command Error Handling B-30
Mount File Cartridge (MC) B-30
MC Command Error Handling B-31
Pack File Cartridge (PK) B-31
PK Command Error Handling B-32
File Manipulation B-33
Records and File Types B-33
Scratch Files B-34
Accessing a Disk File B-34
Creating a File B-35
Purging Files B-35
Storing Data on a Device or New File B-36
Listing the Contents of a File B-36
Renaming Files B-36
File Manipulation Commands B-36
Create a File (CR) B-37
CR Command Examples B-39
CR Command Error Handling B-39
Dump Data to a Device or Existing File (DU) B-40
DU Command Examples B-41
List Contents of a File (LI) B-42
LI Command Example B-43
Purge a File (PU) B-44
PU Command Examples B-44
Rename a File (RN) B-45
RN Command Examples B-45
Store Data on a Device or New File (ST) B-45
ST Command Examples B-48
Transfer Files B-51
Creating a Transfer File B-51
G-Type Global Parameters B-51
P-Type Global Parameters B-53
TR and Related Commands B-55
Calculate Global Parameter (CA) B-56
CA Command Examples B-57
Display Parameters (DP) B-58
DP Command Examples B-58
Conditional Skip (IF) B-59
IF Command Examples B-60
Pause and Send Message (PA) B-61
PA Command Example B-61
Set Global Parameter (SE) B-61
SE Command Examples B-62
Transfer Control to a File or Device (TR) B-62
TR Command Examples B-63
Device Manipulation B-64
Calling FMGR and COMND B-64

Device Manipulation Commands B-66
Display/Modify Buffer Limits (BL) B-66
BL Command Examples B-67
Make Device Unavailable (DN) B-68
DN Command Example B-68
Display Device Status B-69
DS Command B-69
Other FMGR Commands B-70
COMND B-71

Appendix C
CS/80 Exerciser Utility (EXER)

Introduction C-1
Getting Started C-1
Loading the Program C-2
Using the Exerciser C-2
Selected Command Descriptions C-5
Error Handling C-8
EXER and CS/80 Tape Drives C-8
EXER and Cache Disks (HP 793xXP) C-8
EXER and SubSet 80 Disks C-8

List of Illustrations

Figure 3-1 Sample Hierarchical File Organization 3-7
Figure B-1 Cartridge Structure B-11
Figure B-2 Disk Structure Before and After Packing B-14
Figure B-3 Initialized Cartridge Structure B-29
Figure B-4 Global Parameters B-53

Tables

Table 1-1 Typical RTE-A Interactive Programs 1-2
Table 2-1 System Commands 2-1
Table 2-1 System Commands (continued) 2-2
Table 2-2 System Status Commands 2-6
Table 3-1 File Manipulating Commands 3-2
Table 3-2 Standard File Type Extensions 3-4
Table 3-3 Mask Qualifiers 3-16
Table 4-1 Program Control Commands 4-1
Table 4-2 Three Models of EMA/VMA 4-13
Table 6-1 LI Flags Summary 6-99
Table 6-2 LI Commands Summary 6-100
Table 6-3 LI Responses 6-102
Table 6-4 Expressions Summary 6-102
Table 6-5 MPACK Options Summary 6-118
Table 6-6 PU Responses 6-144
Table 6-7 WH Options Summary 6-197
Table 7-1 Stack Mode Commands 7-2
Table 7-2 Editing Commands 7-3

Table B-1 FMGR Control Commands B-3
Table B-2 Disk Manipulation Commands Summary B-16
Table B-3 CO Command Options Summary B-18
Table B-4 Record and File Type Equivalences B-33
Table B-5 File Manipulation Commands Summary B-37
Table B-6 Transfer Commands Summary B-55
Table B-7 FMGR/CI Commands B-70
Table B-8 COMND Commands B-71

1

Introduction

This chapter presents an overview of the RTE-A features available through the display terminal. Basic concepts used throughout this manual are also included here.

It is assumed that the RTE-A Operating System is booted and operating so that interactive operations can be performed. System bootup information can be found in the *RTE-A Primary System Software Installation Manual*, part number 92077-90038, and the *RTE-A System Generation and Installation Manual,* part number 92077-90034. Interactive operations are accomplished through the Command Interpreter (CI), a system program that allows effective use of the computer system through a set of user commands. There are other interactive programs in the RTE-A system, described in separate manuals. Refer to the *RTE-A Index and Glossary Manual,* part number 92077-90036, for a documentation map and description of each manual.

Using RTE-A

The RTE-A Operating System is the foundation for program development, file management, and support of user application programs. It includes many programs that help you perform these functions. The typical interactive programs are shown in Table 1-1. Many HP products can also be ordered with the RTE-A system to add other features. For example, ordering the HP 92078A Virtual Code (VC+) Package provides the following capabilities: outspooling, I/O redirection, large program and shared program support, and multiuser operations.

In program development, an editor (EDIT) program can be used to create and modify source programs and files. You can then compile the programs with the appropriate language compiler (or assembler) and link them with the LINK program. These programs can be debugged with the HP 92860A Symbolic Debug/1000 program. Control of the executable program is accomplished through the CI program.

File management tasks can be done through system programs such as EDIT and CI. EDIT creates and modifies files, and CI provides many file manipulating functions. You can merge files with the MERGE program and print hard copies through the system printer with the PRINT utility. Files can be transferred between different media with the LIF, TF, and FST utilities. In addition, you can use the file management program (FMGR) and the file copy utility (FC) to perform tasks similar to those provided by CI and TF. These are used primarily to handle existing FMGR files. The limitations of using FMGR are discussed in Chapter 3 under the FMGR Files heading.

Special application support is accomplished through the CI program, which provides the capabilities to monitor system status and the control of the interactive system operations.

Table 1-1. Typical RTE-A Interactive Programs

Name	Prompt	Description	Manual
CI	CI>	User interface; simplifies interactive system operations	User's Manual (92077-90002)
EDIT	/	Text editor with screen mode	EDIT/1000 User's Guide (92074-90001)
DEBUG	Debug>	Symbolic Debugger with profiler	Symbolic Debug/1000 User's Manual (92860-90001)
LINK	link:	Linkage Editor	LINK User's Manual (92077-90035)
FST	FST>	High performance logical (File-by-File) backup utitlity	Backup and Disk Formatting Utilities Manual (92077-90249)
TF	TF:	File copy utility	Backup and Disk Formatting Utilities Manual (92077-90249)
FMGR	FMGR :	Alternate user interface used for handling FMGR files	User's Manual (92077-90002)
FC	FC:	Alternate file copy utility for handling FMGR files	Backup and Disk Formatting Utilities Manual (92077-90249)

Command Interpreter Features

The Command Interpreter (CI) is usually scheduled and run automatically as soon as the RTE-A operating system is booted. The program prompt is displayed, and CI is ready to accept a command. The command 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 placeholders. Following are examples of CI command entries.

```
CI> edit                    (Runs the Editor program to create a file.)

CI> wh                      (Displays system status.)

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

CI> dl /jones/              (Displays all files in directory JONES.)
```

Interactive operations available in RTE-A through CI include the following functions:

System Status Check
System Control
File Manipulation
Program Control
Multiuser Operations (VC+ Only)
Spooling (VC+ Only)

The Command Interpreter provides commands that start and stop programs and commands that change the way programs execute. Commands are available for the creation of directories and subdirectories that are used in file management. Files can be created, copied, stored in a directory or subdirectory, purged (destroyed), 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 update. System information can be displayed on the terminal screen; for example, program status and input/output device status. System behavior can be controlled through the system commands.

One of the features available in the optional VC+ Package is spooling capability. Spooling provides the means to control shared devices; its most common use is for shared access to the system line printer. A description of the spool system is contained in Chapter 2.

Interactive operations are performed through the display terminal keyboard. Each of the functions mentioned above includes a variety of tasks such as executing programs, creating files, and copying files. Each task may be executed with a single CI command. The commands are described in Chapters 6 and 7, and typical interactive tasks are described in Chapters 2 through 5.

In a VC+ system, CI provides another level of commands for the superuser, a person who is thoroughly familiar with the RTE-A system. Typically, a superuser manages the system. A superuser is designated at system installation time and can in turn designate any user as a superuser. In addition to using all the interactive capabilities described in Chapters 2 through 5, the superuser is allowed to do the following:

- Change the properties of any program (including all protected system programs). Applies to all program control commands.
- Override file protection. Applies to all file manipulating commands.
- Modify system attributes such as the system clock. Applies to all system control commands.
- Re-initialize disks.

If Security/1000 is on in a VC+ system, CI filters its commands using capability levels. The user must have a capability level equal to or greater than that defined for the command to use it. Users with capability level 0−30 are subject to capability level checking and file system protection. Users with capability level 31 are superusers and have full access to the system. That is, they can use all commands, run all programs, and override file protection.

Introduction to Programs

Programs are written for a specific task or set of tasks; they control the computer's execution. There are various properties associated with programs, including what they do, how much memory they use, and how important they are relative to other programs.

In order to reduce the amount of information maintained by the RTE-A Operating System at any time, many programs are kept in an "unrestored" state. The unrestored program is a file that contains a memory image of the program, along with information describing the applicable program properties. Such a file must be restored into a system program table before the program can be executed.

Restoration of a program file can be done automatically when you schedule a program for execution. You may also restore the file and then perform other program control tasks.

There are several kinds of programs, each with different system memory or data handling requirements. There are differences in the type of instructions that programs need, the type of users allowed to change the way the program runs (system programs), and the way the program uses memory (EMA/VMA programs). All of these items are discussed in Chapter 4. Note that the system does not differentiate programs on the basis of what type of computation they do or the language in which they are written.

Introduction to Files

Files contain various types of information organized in a specific manner to facilitate storage, access, and data transfer. The information can be text (usually called ASCII data), data collected from some experiments or tests (binary data), information used by programs, or even programs themselves.

Files are identified by file names. Additional information is added to the file name to describe the file's associated properties. The storage location of the file is recorded in a directory. 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 called 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)
`userManualchap2.txt`	(File name)
`progb.rel::directory`	(File descriptor)
`/directory/progb.rel`	(File descriptor)
`/directory/subdirectory/chapter6:::4:609`	(File descriptor)

In the RTE-A 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.

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

System Information

Certain system information can be displayed on the display terminal screen with CI commands. The system time can be obtained with the TM command. Other commands display information about I/O devices and the system program status. The IO command can be used to show the I/O devices in the system and the LU numbers associated with those devices. The WH command can be used to show program status. Detailed descriptions of these commands as well as others that control system operations are given in Chapter 6 of this manual. Chapter 2 of this manual describes how to use the common system commands.

CI Termination

The Command Interpreter can be terminated at any time with the EX command. If your system has multiuser capability, the EX command will also log you off (if you are using the startup copy of CI, not a secondary copy run later). However, if you are running programs without wait when you give the EX command, CI needs more information about what to do with these programs. It will ask whether you want to wait until they finish, stop them, or (in a multiuser environment) allow them to run as background programs.

When CI is Not Available

CI may be busy when you want to enter another CI command. This usually happens if the program you are running takes a long time or if it gets into trouble, such as running out of printer paper when running PRINT.

A copy of CI called CM is kept waiting to handle such situations. Interrupting the system while CI is busy will run a standby program. You can interrupt the system by pressing any key on your session terminal, unless the terminal is connected to a multiplexer port that has FIFO buffering or type-ahead turned on. In all cases, pressing the break key on your session terminal will interrupt the system. Note that special consideration must be given to using the break key on the system console. Refer to the *RTE-A System Manager's Manual,* part number 92077-90056, for information on the system console and VCP.

Once you get the system's attention, the following prompt is displayed:

```
CM>
```

This is the prompt for a copy of CI named CM, which differs from CI in two ways: CM always runs programs without wait, and it exits after completing a single command. It lets you obtain system status quickly. Besides getting a system status report, you can use CM to stop the program for which you are waiting. The TR, EX, FUNCTION, IF-THEN-ELSE-FI, and WHILE-DO-DONE commands (refer to Chapter 6), and the command stack cannot be used in CM. The RS command is available only in CM.

After executing one command, CM terminates and control is returned to CI. If the busy situation persists, you need to get the system's attention again to run CM and execute another command.

Be careful when running CM; it is designed as a backup user interface and not for tasks that take a long time, such as copying files. Doing such tasks keeps others from using CM because there is only one copy of CM. Use CM only for simple commands such as OF, RS, UP, or WH.

At times CM may be busy either with a normal command or because an operation is blocked. In these cases, you will encounter the system prompt:

```
System>
```

If your system does not have the optional VC+ Package, you may encounter another prompt while CM is busy:

```
RTE:
```

These prompts may occur if the operating system is interrupted or when there is a problem while running CM. If the prompt appears as a result of an interrupt such as pressing a key, the return key should return control to CM. Refer to Chapter 8 for a discussion of exception conditions and the RTE and system commands.

2

Using the System

This chapter discusses using the system to display system status, manipulate files, and control programs. There are system commands that display help information and system status, and control certain system operations. In a VC+ system, commands that affect system processes and control system operations are reserved for the superuser who manages the operating system. Superuser capabilities are described in Chapter 5. A summary of system commands is shown in NO TAG below.

System Commands

Command	Task
? (or HELP)	Display help summary
? *command* or HELP *command*	Display command description
/ [*parameters*]	Access command stack
ALIAS [−x|+x] [*alias_name* [[=] *string_value*]] ALIAS *alias_name*	Define CI alias or display defined aliases
CN *lu* [*function*[*pram*4*]]	Control I/O device
ECHO [*parameters*]	Display parameters to terminal
EX [B|C|L]	Exit CI

FUNCTION [−x|+x] *function_name* {
command-list
}
	Define a function

FUNCTIONS [−x|+x|*function_name*]	Display functions
IF *command-list* THEN *command-list* [ELSE *command-list*] FI	Execution control structure
IO [*parameters*]	Display I/O configuration
IS *strg1* <*rel op*> *strg2* [*opt*]	Compares two strings or numbers

Table 2-1. System Commands (continued)

Command	Task
SET [*variable* [= *string*]] SET [−x|+x] [*variable* [[=] *string*]]	Display/define positional, user-defined, and predefined variables
TM	Display system time
TO *lu*[*interval*]	Display/modify device timeout
[TR] *file*[*pram***9*]	Transfer to command file
UL *label*	Unlock shareable EMA partition
UNALIAS *alias1* [*alias2* ...]	Delete one or more aliases
UNSET *variable1* [*variable2* ...] UNSET −f *function1* [*function2* ...]	Delete a user-defined variable or function
UP *lu*	Up a device
WH [*parameters*]	System status reports
WHILE *command-list* DO *command-list* DONE	Execution control structure

Getting Help

The CI program provides an online 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. For example, here is the response from a ? command:

```
CI> ?
Help available on: (use ? <command> for help on <command>)
directory /HELP.DIR
??              ADVLINK         ALIAS           AS
ASK             AT              BR              BREVL
BRTRC           CALLM           CALLS           CD
CI              CL              CLOSE           CN
CO              COMPILE         CP              CR
CRDIR           CZ              DC              DL
DSCOPY          DSRU            DT              ECHO
ERROR           EVMON           EX              FMTRC
FOWN            FPACK           FREES           FREL
FTP             FUNCTION        FUNCTIONS       FVERI
GO              GREP            GRUMP           IF
IN              IO              IS              KILLSES
LC              LI              LINDX           LINK
LNS             LOGCHG          LS              MACRO
MAIL            MAKE            MAKEFILE        MASK
MC              MERGE           METER           MO
More...
```

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" displays the following:

```
CI> ? of
OF -- Terminate a program and optionally remove its ID segment or
remove a prototype ID segment

Usage: OF [prog[/session][opt]]

prog is either a program name with optional session qualifier or
the name of a prototype ID segment if the D option is given.  The
default is the last scheduled unprotected program on the primary
program (usually CI) chain.
    :
    :
More...
```

In addition to commands and explanations, other information is available with the help command, including such items 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 directory /HELP. You select the file it lists when you ask for help on a particular command. Entering the help command without a parameter displays the contents of directory /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 20 entries are saved; however, the average is approximately 300 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 home directory if one is defined; otherwise, it uses 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 6 for details.

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 section on "Predefined Variables" for details.

To display the command stack, enter a slash:

```
CI> /                                                (A screenful of commands,
 --020/320--   Commands: [CI.STK::DEXTER]             defaulting to 20, is displayed. See
                                                      SET command description in
                                                      Chapter 6 for changing the default.)
dl
   .
   .
   .
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/
```

The command stack window is preceded by a banner that contains the starting line number of the window and the total number of lines selected for this display in inverse video, separated by a slash, and the name of the current stack file in square brackets, for example, [newfile].

Note that the cursor is at the bottom of the stack. Pressing the return key returns to CI, and a new command can then 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 was 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> //
--001/320--  Commands: [CI.STK::DEXTER]
prot report5.sale rw/
```

In this example, pressing the carriage return key repeats 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 maximum of 80 slashes is allowed.

You can enter a slash followed by a number (/n). 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 enters the command line. If you do not wish to enter this line, you can move the cursor to a blank line and press the return key to return to the CI prompt, or enter a slash to repeat the whole command stack.

After you have displayed the command stack and before you return to the CI prompt, you can use stack mode commands in addition to terminal keys to manipulate the stack and mark groups of commands. Refer to the command stack descriptions in Chapter 7 for details. You can display all command lines containing a specific string by entering a slash followed by a period followed by the string (for example, /.string). If you insert a caret (^) between the period and the string, only the command lines starting with the string are displayed. Refer to Chapter 7 command stack descriptions for examples.

You can change the command stack display size by using the SET command. For example, the following command changes the command stack display size to 15 lines:

```
CI> set frame_size = 15
```

Obtaining System Status

You can display system information on your terminal screen with the appropriate CI command. A brief description of how to use some of these commands to get the desired information is explained in this section. Note that these are the most common uses. A full description of each command is given in Chapter 6.

System Status Commands

Command	Description
WH [*parameters*]	Display program status and system memory usage. *
IO [*parameters*]	Display system devices and attributes of those devices.
METER [*lu*] [*commands*]	Display CPU usage.
SAM [AL] [XS] [*lu*|QU]	Display status of System Available Memory (SAM).
SPORT [*port_lu* [*display_lu* [*waitflag*]]]	Display status of serial ports (using device driver DDC00/01).

* Note that although WH is extremely useful it does not provide a true snapshot of the system state. To restrict all other system activity while WH takes a snapshot could compromise the real-time characteristics of the system. WH can, therefore, occasionally report inconsistent information.

Display Program Status

To display the status of your programs, use the WH command as shown in the following example.

```
CI> wh
Program             DataPartition      CodePartition
Name    Prio    PC     Seg  Size Status   Size Status  Program Status
-------------------------------------------------------------------
Session  44  User CAROLYN
CI       51 14532          32 in                    waiting for WH
PROG2    90 54321          32 in                    scheduled
WH       51  5640          15 in          5         scheduled
-------------------------------------------------------------------
Fri Feb 14, 1992 3:49:35 pm
```

In this example, the user entering the WH command is logged on as CAROLYN to session 44 and has three programs active: CI, PROG2, and WH. Note that one of the programs is the WH program itself, which is run when the WH command is entered. CI is waiting for WH to finish. PROG2 is running at priority 90, but because 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 Chapter 6.

To display the status of all programs, enter:

```
CI> wh al
```

A sample display is shown below.

```
CI> wh al
Program                DataPartition CodePartition
Name   Prio   PC    Seg  Size Status   Size Status  Program Status
-------------------------------------------------------------------------------
Session   1  Superuser MARK
CI       51 14515          32 in                    waiting for WH
```

```
WH        5  6330          15 in                    scheduled
CM        2     0          32                       dormant

System Session
D.RTR     1 22574          32 in                    dormant saving resources
PROMT     3  3322          10 in                    dormant saving resources
LOGON     2  3056          13 in                    class susp on class #50
--------------------------------------------------------------------------------
Down lu's :  3
--------------------------------------------------------------------------------
Thu Feb 20, 1992 11:16 am
```

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.

The status column in this example includes the following categories:

Status	Meaning
scheduled	The program is scheduled to run or is 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 suspended, waiting for directives.
class susp on class #xx	The program is suspended waiting on a class number, usually waiting for I/O.

Display Memory Usage

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

```
CI> wh pa

 Ptn#  Page   Range  Size   Occupant     Status          Priority
-------------------------------------------------------------------
 No reserved partitions.

       48-    79    32     CI/1                           51
       80-   111    32     D.RTR        saving resources   1
      112-   121    10     PROMT        saving resources   3
      122-   134    13     LOGON                           2
      135-   149    15     WH/1                            5
      161-   255    95     free
-------------------------------------------------------------------
 Thu Feb 20, 1992 11:20 am
```

Display I/O Configuration

Systems differ widely in the number and types of peripherals such as disks, printers and tape drives. The 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. The LU number for a particular device can be used to specify that device. The I/O configuration information is displayed with the IO command.

The −cs option of the IO command displays on your terminal screen a listing of the devices on your system. For each LU, the IO command returns the type of device attached to that LU, a select code identifying the I/O card the device is attached to, and other configuration information, depending upon the parameters. Note that select codes are in octal.

The IO command can be used to list all devices in the system, a selected device specified by the LU number, or a range of devices.

Following is an example that displays information about LU 6.

```
CI> io -c 6

                    select       bus    dvt     dvt     interface
    lu device name  code         addr   nbr     addr    type

    6 printer (12)    25          5     63      51707     37

  Thu, Apr 28, 1988, 10:05 AM
```

All of the fields in the listing and possible values are explained in the IO command description in Chapter 6.

To display all devices grouped by select code, enter:

```
CI> io –sc
 select code 20b:   interface type 0b

       lu    device name

        1    terminal (5)
        2,3  cassette tape (20)


 select code 25b:   HPIB interface (37)

 bus
 addr   lu    device name

  1    40–43  MAC/ICD disk (32)
  2    59     printer (12)
  3     7     streaming tape drive (24)
  4     8     tape drive (23)
  5     6     printer (12)
  6    52,53  floppy disk (30)
 36     9     instrument (77)
```

To display a range of devices, for example, LUs 6 through 9, enter:

```
CI> io –c 6 9
                                  select  bus   dvt dvt   interface
 lu  device name                   code   addr  nbr addr  type

  6  printer (12)                    30     2   11  36460  37
  7  not assigned
  8  tape drive (23)                 26     4   9   36372  37
  9  CS/80 tape drive (26)           31     2   7   36273  37

Thu, Dec 10, 1992,  3:03 PM
```

Controlling Devices

There may be times when you need to control the operations of an I/O device from the terminal, such as to rewind tape or eject paper. 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 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 assumes that the command without any control parameter is rewinding tape or ejecting paper, respectively.

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

In each RTE-A computer system, there are many different types of devices that are 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 description in the *RTE-A Driver Reference Manual,* part number 92077-90011, for details.

Various other I/O control requests can be issued with the CN command. For example, the following command sets up multiplexer ports on the Revision C MUX (IDM00):

```
CI> 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. The equivalent for the Revision D MUX (IDM00) is:

```
CI> cn 1 30b 131b
```

The CN command treats LU 1 as the system LU 1, rather than as your terminal. Other control requests are described in the CN command description in Chapter 6.

Changing I/O Device Attributes

The CI program provides several commands that modify I/O operations. The RTE-A 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-A System Generation and Installation Manual*, part number 92077-90034. 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.

In addition to the attributes mentioned above, there are others that can be changed in special situations. These are the device HP-IB address, device priority (not to be confused with program priority), and the driver parameters specified at generation time. In VC+ systems, modifying these requires superuser capability. Details are given in Chapter 6.

How to Bring Up a Device

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

```
CI> up 8
```

This allows I/O requests to go to the device. If the original problem recurs, the device goes down again. This happens for various reasons. A device may be inadvertently taken offline, effectively disconnected from the system. Tape drives go offline because most tape save/restore utilities put the tape drive offline when they are finished to allow removal of the tape and to prevent another user from using the tape drive. Printers are taken offline for manual form feeds. Whenever a device is offline and you need to access it, you must first place it online and bring up the device. Otherwise, the device cannot complete the control request and the operating system marks the device as down.

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

```
CI> cn 8
I/O device error on LU 8 The reason is:
Device timed out
Device has been downed (use UP,lu to try recovery)
```

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

Changing Timeout Values

In most cases, there is something wrong if an I/O operation takes too long. A disk or printer should always respond within 1 second and a disk I/O operation should complete in 5 seconds. The RTE-A Operating System detects when a problem occurs through a mechanism called a timeout.

Each LU has a timeout value associated with it. This is a device attribute that tells the system how long to wait for a response from the device. When the system starts an I/O operation, it also starts a timeout timer. If the 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.

Timeout values can be specified either during system generation or with the TO command (by the superuser only). They are specified in units of 1/100th of a second. A timeout value of 100 means one second. This unit is chosen to match the resolution of the time base generator.

To set the timeout on LU 8 to 10 seconds, enter the following:

```
CI> to 8 1000
```

This sets the device on LU 8 (by normal convention a magnetic tape unit) timeout value to 10 seconds.

There are two other useful forms of the TO command. Specifying a timeout of zero really requests an infinite timeout. 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 timeout values to zero.

Entering the TO command with an LU but without any timeout parameter displays the timeout value currently in effect for that LU. To display the timeout value for your terminal, use the system LU number for the terminal, 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> tm
Wed Mar 2, 1989 7:39:34 am
```

It is important to maintain the correct system time. Otherwise, the RTE-A features such as time scheduling programs and time stamping files cannot be used effectively. Refer to the discussion of the TM command in Chapter 6 to learn how to set and reset the system time.

Executing a Command File

To execute a series of commands without user intervention, a command file can be created using the EDIT program. The file contains all the commands to be executed in the desired sequence.

To execute commands in the file, the command file name is entered with or without the TR command. At the end of the command sequence, CI returns to the source of the TR command, either another command file or the interactive mode. We recommend that you use the file type extension .CMD on all your command files. Note that the .CMD file type extension is required if the file is to be found via UDSP#2. Refer to Chapter 6 for a detailed description of the PATH and TR commands.

The following is a sample command file 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 REPORT.CMD, enter any of the following:

```
CI> tr report.cmd

CI> tr report

CI> report

CI> report.cmd
```

Functions are memory-resident command files that are defined via the FUNCTION command described in Chapter 6. To execute a function, simply enter its name:

```
CI> myfunc
```

Positional Variables

Positional variables are defined in the CI command string 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, either of the following commands sets the positional variables $1 through $4:

```
CI> ru ci myfile.cmd prog1 prog2 prog3 prog4

CI> tr myfile.cmd prog1 prog2 prog3 prog4
```

Positional variables can be separated by blanks or commas but commas must be used to specify non-consecutive positional variables. For example, to transfer to a command file and specify values for only $1 and $4, enter:

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

Three commas are required to ensure 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 more than 9 variables are specified, 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 variables cannot be deleted.

Once values are set for the positional variables, they are used until CI terminates, another TR command is executed, or you exit from a command file.

The values of positional variables are local. Before executing a TR command or a user-defined function, CI saves the current values of $1 through $9. While executing the command file or function, the values specified in the TR or function command string are used in the variable substitutions. When the command file or function is exited, the previous values of $1 through $9 are restored.

A command file must be specified to 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> 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> tr 1 R T E                          (3 parameters are set up)

CI> co $1$2$3AnswerFile SpoolA          (Copies file RTEAnswerFile)
```

User-Defined Variables

User-defined variables are defined using the SET command and deleted using the UNSET command. See Chapter 6 for a detailed description of the SET and UNSET commands.

User-defined variables are available to both CI and command files. A variable defined in response to a CI prompt can be used in a command file, and a variable defined in a command file can be used in response to a CI prompt. Note that a user-defined variable is referenced by preceding the name with a dollar sign ($).

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

Another example of concatenation is as follows:

```
CI> set dir = /system            (Define a directory)
CI> 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 thinks the variable name is $DIRANSWERS because the blank after ANSWERS is the first invalid character for a variable name.

To concatenate two words, use the backslash (\) or the single character quote described in the section on "Quoting". In the following example, variable NAME is set to "user", and the backslash is used to concatenate the variable with other ASCII strings.

```
CI> set name = user
CI> li $name\2                   (List file USER2)
CI> li $name\3                   (List file USER3)
CI> 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 PROG, you must type P instead of p.

You should delete unnecessary 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 affect user-defined and predefined variables; for example, CI may not have enough space to return a value to a variable. There is no set number of user-defined variables. The names and values of variables are hashed into two separate tables, so the number allowed depends on the lengths of the names and values.

Environment Variables (VC+ Only)

Environment variables allow programs within a session to share variables. This is an extension of the SET command in CI. The original SET command can be thought of as resulting in "local" (to one copy of CI) variables. The extension creates "session" variables that can be "exported" to (and "imported" from) the session environment. Exported variables are available to all programs in the session.

Note that it is possible to have a local and an environment variable by the same name. Access is always to the local variable.

The SET command by itself displays all defined variables, in alphabetical order. The local variables are listed first, then a blank line and then the environment or exported variables.

Environment variables can be used either interactively or in CI transfer files. Any program in the session can access the value of an environment variable. Variable access in CI command lines is done by preceding the variable name with a dollar sign ($) as in the preceding section.

The environment variables are stored in a space called the Environment Variable Block (EVB). The EVB is kept in sharable EMA and is identified by the session number; for example, Environment 90 for session 90. The block is allocated and initialized when a user logs on. The size of a user's EVB is indicated in the user file. If you need more space in your EVB, see your system manager.

The UNSET command is used to remove either type of variable.

```
CI> unset [-f] varname
```

When the UNSET command is given, first the local variable list is searched; if a local copy is found, it is unset. If no local variable is found, the exported variables are searched. Remember that unsetting an exported variable removes it for all programs in the session.

Refer to Chapter 6 for a detailed description of the SET command.

Predefined and Other Variables

When you begin a CI session, some of the variables are predefined. These variables are initialized to default values by CI. However, you can use the SET command to modify the values of all the variables except $MY_NAME. You can also modify $WD, but CI updates its value after each CD or WD command. Likewise, you can modify the $RETURN variables, however, CI does not recognize the new value when evaluating control structures. With VC+, some variables are automatically exported and cannot be imported.

The SET and ECHO commands can be used to display the values of predefined variables. Unless otherwise noted, the following variables are predefined and you cannot use the UNSET command to delete them.

$AUTO_LOGOFF
: Allows for automatic logoff if the session is inactive. CI initializes $AUTO_LOGOFF to zero, which means automatic logoff is not in effect. If you set $AUTO_LOGOFF to a non-zero value, CI times out after that many terminal timeouts. If CI is the only active program, after four CI timeouts an EX,B command is executed to terminate the session.
  Default: 0
  Allowed: ranges from 0 to 32767

$CMNDO (VC+ only)

The $CMNDO variable can be set to TRUE or FALSE to control CMNDO usage in programs that support the CMNDO monitor. This variable is not predefined. Refer to Chapter 7 for information regarding CMNDO.

$COLUMNS (VC+ only)

$COLUMNS is the number of columns in your display. This variable is used by LI, LS, and FmpPaginator. CI automatically exports it at startup. You may import, modify or unset $COLUMNS. If the value is invalid, CI uses the default. The RESIZE program can be executed to compute the size of your display and set $COLUMNS accordingly. $COLUMNS can be deleted by the UNSET command.
Default: 80

$DATC

The datecode revision of the operating system; for example, 6000 for Revision 6.0. This variable is for user information and can be deleted by the UNSET command.

$EVB_SIZE (VC+ only)

The size of the session's EVB in pages; zero if no EVB is available. This variable is for user information and can be deleted by the UNSET command.

$FRAME_SIZE

The size of the command stack display in lines. When you log on, the command stack display size is initialized to twenty lines. It can be set to any positive integer greater than zero.
Default: 20
Allowed: ranges from 1 to 32767

$HOME

For VC+ CI, $HOME is your home directory, as given by User Defined Search Path (UDSP) #0, set up by the PATH program or CI. If PATH is not run, $HOME is set to the directory in which the first copy of CI starts up. $HOME is automatically exported and cannot be imported or deleted by UNSET.

For non-VC+ CI, $HOME is set to the directory in which CI starts up.

$IFDVR (VC+ only)

The interface driver of your port; for example, IDZ00, ID100, ID400, or ID800. This variable is for user information and can be deleted by the UNSET command.

$KILLCHAR (VC+ only)

$KILLCHAR is used with the visual editing modes described in Chapter 7. When the kill character is entered, the contents of the current command line become invalid and the user is given another prompt. When not defined by the SET command, the DEL character is used.

$LINES (VC+ only)

$LINES is the number of lines in your display. This variable is used by LI, LS, and FmpPaginator. CI automatically exports $LINES at startup. You may import, modify, or unset it. If the value is invalid, CI uses the default. The RESIZE program can be used to set the value correctly for your display. $LINES can be deleted by the UNSET command.
Default: 24
Allowed: ranges from 0 to 32767

$LOG

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

$LOGON

The user and group name with which the user is associated during the session, in the form "*user*`.`*group*".

$MY_NAME

The true or scheduled name of CI. This variable can never be altered.

$OLDPWD

$OLDPWD is set to the previous working directory ($WD) whenever a WD or CD command is executed.

In VC+ CI, $OLDPWD is automatically exported and cannot be imported or deleted by UNSET.

$OPSY

The value of the system entry point $OPSY for the current system. For example, the value of $OPSY for RTE-A Revision 6000 is −125.

$POLL

This variable is set by the POLL command to be the command that is executed via the POLL command. $POLL can be altered by the SET command and can be deleted by the UNSET command. $POLL is not predefined.

$POLLINT

The approximate number of minutes between the successive executions of $POLL. $POLLINT cannot be altered by the SET command but can be deleted by the UNSET command. $POLLINT is not predefined.

$PROMPT

The prompt that is displayed when CI is waiting for input. CI initializes this variable based on the program name with which CI has been scheduled, for example, "`CI..A>`".
Default: `CI>`
Allowed: up to 78 characters in VC+; up to 16 characters in non-VC+

$QUIET_CMD (VC+ only)

A flag indicating whether or not certain CI commands should run in quiet mode. If this variable is set to ON, the commands CO, MO, PROT, PU, RN, and UNPU do not display any messages if there are no errors. If it is not set or is set to OFF, the commands display messages for each file acted upon. This variable is not preset.

Allowed:

ON	CI does not display file messages except when errors occur.
OFF	CI gives messages for each file that is being acted upon.

$REPROMPT (VC+ only)

If this variable is not set or is TRUE, CI automatically reissues the prompt upon a timeout. This can be inconvenient to workstation users using terminal emulators that de-iconify themselves when they receive output (that is, hpterm with mapOnOutput set to True). To correct this behavior, you can either set the terminal's timeout to 0, which is not recommended, or set REPROMPT to FALSE. If REPROMPT is set to FALSE and the terminal emulator still de-iconifies itself when a timeout occurs, change the interface driver protocol to not use DC1 handshakes (for example, use Xon/Xoff protocol). See the *RTE-A Driver Reference Manual*, part number 92077-90011, for details.) This variable is not predefined.
Allowed: TRUE or FALSE

$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

A character string returned from execution of the last command. The string is 256 characters in VC+, and 80 characters in non-VC+. CI updates the value as commands are executed.

$RU_FIRST

A flag indicating whether RU or TR is to be assumed if you enter only a file name in response to a CI prompt or as a line in a command file. $RU_FIRST can have the following meanings:

TRUE	CI first attempts to execute an RU command for the specified file name.
FALSE	CI first attempts to execute a TR command for the specified file name.

You should set the variable to FALSE if you execute more command files than program files.
Default: TRUE

$SAVE_STACK

A flag indicating if the command stack is saved when you exit CI or when the command stack file is changed with the WD command. $SAVE_STACK can have the following values:

TRUE	the stack should be saved.
FALSE	the stack should not be saved.

Default: TRUE

$SESSION

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

$VISUAL (VC+ Only)

Required to edit the command line in a manner similar to ksh or csh, command interpreters that are used in the HP-UX operating system. Values can be CSH, EMACS, GMACS, VI, or CI, with or without the NODUPES option. This variable is not predefined.

Example: "`set visual = emacs`" allows file name completion and emacs-type editing of the command line.

Setting $VISUAL to EMACS, GMACS, VI, or CSH also has an effect on the functionality of the RTE command stack. By default, the RTE command stack routines do not insert duplicate lines in the stack. When setting $VISUAL, duplicate lines are allowed in the stack. To override this behavior, a "`,NODUPES`" can be added to the visual mode. For example, to use the VI editing mode without saving duplicate lines in the command stack, set $VISUAL to "`vi,nodupes`".

Executing the command "`set visual ci,nodupes`" is equivalent to not setting the variable. Unsetting it accomplishes the same thing.

Allowed:

`emacs,[nodupes]`	ksh-style EMACS command editing
`gmacs,[nodupes]`	ksh-style GNU emacs command editing
`vi,[nodupes]`	ksh-style VI command editing
`csh,[nodupes]`	csh-style name completion and directory lists
`ci,[nodupes]`	disables command editing, except for "/" commands

See Chapter 7 for more information on the use of $VISUAL.

$WD

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

The following example changes the value of $PROMPT:

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

The following example displays the value of $OPSY:

```
CI> echo $opsy
-125
```

Tilde (~) Substitution

Each command parameter is checked to see if it begins with an unquoted tilde (~). If it does and the tilde is alone or before a slash (/), the tilde is replaced with the value of the $HOME variable. This is known as tilde substitution. A tilde followed by a plus (+) or minus (−) is replaced by the value of the variable PWD or OLDPWD, respectively. In addition, tilde substitution is attempted when the value of a variable assignment begins with a tilde. For example:

```
CI> echo foo=~/bar
FOO=~/BAR
CI> set foo=~/bar
CI> echo $home
/EAH
CI> echo $foo
/EAH/BAR
```

Note that in the case of:

```
CI> co ~/foo>remote_node @
```

"~" will not use the home directory on the remote_node. For example, assume your home directory on remote_node is /EAH. Your home directory on the local node is /ESTHER. The above command is equivalent to:

```
CI> co /esther/foo>remote_node @
```

not

```
CI> co /eah/foo>remote_node @
```

as might be expected.

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 one level of command file nesting to the previous level, these values are restored.

If CI is currently executing nested command files and you wish to stop, issue the BR command. CI will give you the following prompt:

```
Multiple levels of command files.  Do you wish to
(R)eturn to prior command file or (A)bort all command files [A]?
```

If you reply with either an "A" or a carriage return, CI returns to the interactive level. If you reply with an "R", CI exits from the command file currently executing and proceeds with the previous (upper) one.

String Processing and Positional Variables Parsing

CI processes a non-quoted command string by taking the following actions:

1. Shifts it to uppercase.

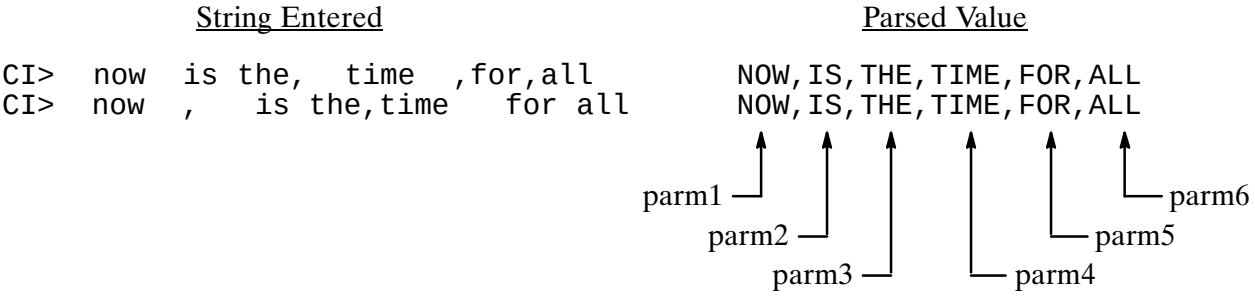
2. Strips any contiguous group of blanks that precedes or trails a comma.

3. Replaces each remaining contiguous group of blanks with a comma.

4. Delimits a command at a semicolon.

5. Performs variable substitution before executing the command.

Thus, the string is shifted to uppercase and parameters are delimited by spaces and/or commas. For example:



Strings substituted for dollar variables are processed in a slightly different manner. The case of the string is not altered and spaces are preserved in the substituted string.

Positional variable parsing occurs after the string processing for the commands that set the positional variables. When parsing the positional variables $1 through $9 of the substituted string, each contiguous group of blanks that precedes or trails a comma is stripped but all other blanks in the string are preserved. This has the effect that substituted string parameters are delimited by commas but not by blanks. Also, note that commas delimit positional variables even when the commas were originally quoted.

For example, if FIRST.CMD consists of:

```
set log = on
set params = ‘now  is the,     time    ,    for all,good‘
tr ech $params
```

and ECH.CMD consists of:

```
set log = off
   echo >$1<
   echo >$2<
   echo >$3<
   echo >$4<
```

executing FIRST.CMD results in the following:

```
CI> first
SET,PARAMS,=,now  is the,    time   ,  for all,good
TR,ECH,now  is the,    time   ,  for all,good
SET,LOG,=,OFF
>now  is the<
>time<
>for all<
>good<
```

If FIRST.CMD consists of:

```
set log = on
set params = ‘now, is,the,    time   ,  for all,good‘
tr ech $params
```

executing FIRST.CMD results in the following:

```
CI> first
SET,PARAMS,=,now,  is,the,    time   ,   for all,good
TR,ECH,now,  is,the,    time   ,   for all,good
SET,LOG,=,OFF
>now<
>is<
>the<
>time<
```

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 backquotes (‘).

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

Some examples of quoting are as follows:

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

```
CI> echo ru,savename,‘Jane Doe‘
RU,SAVENAME,Jane Doe
```
(Passes a blank in command string)

```
CI> echo ‘This is a backquote (‘‘).‘
This is a backquote (‘).
```
(Passes a backquote as part of the quoted string)

```
CI> echo peek\npoke
PEEKnPOKE
```
(Displays string with the “n” unaltered by CI.)

```
CI> li $file\x
```
(Uses the backslash to indicate that $file is the user-defined variable (not $filex). If $file is Foo, this will list FOOX.)

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 used to separate commands must not be enclosed in quotes.

Two examples are as follows:

```
CI> wh;dl
```

This executes the WH program followed immediately by DL.

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

This 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 the 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 you 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 the execution control structures of CI discussed later in this chapter. 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 ensure that you can access these values before they are modified. Note that SET can be used to modify a return variable, however, when done within a control structure, the new value is not used to evaluate the control structure.

The CZ, DT, OF, PR, PS, SP, SZ, UL, VS, and WS commands do not return status to CI; therefore, $RETURN1 always equals 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, which enable decision making during execution of the command file. (See Chapter 6 for detailed information.)

The following statements compile TEST. If no errors or warnings occur during the compile, TEST is linked; otherwise, EDIT is run on TEST.FTN so you can fix the errors.

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

In the next example, the file SOME_FILE is printed five 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 user written program that accepts two ASCII representations of integers, converts them to integers and performs the specified operation. The result, in ASCII, 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
```

Timeout/Logoff Function

To eliminate inactive sessions on a system, CI can log off a user. The variable $AUTO_LOGOFF can be defined to tell CI how many device timeouts can occur at your terminal before CI times out. Each time CI times out, a warning message is displayed on the terminal. After the fourth timeout, CI executes an EX command.

The next example begins the CI timeout process after CI waits 15 minutes for input. Set the terminal timeout to 30000 (see the TO command) and the $AUTO_LOGOFF variable to 3. If CI receives no input for 60 minutes, the session is terminated.

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

Spooling (VC+ Only)

A spool system manages concurrent access to system resources by multiple users and processes. The spooler establishes a queue of tasks for the resource and decides which of the queued tasks to next schedule for the resource.

The most common usage of spoolers is the sharing of printers and plotters among users. The spool system performs the actual output of the print data to the printer device. A number of services and options related to printing formats are provided, such as generating banner pages at the start of the output, downloading printer fonts, and so on.

There are three spoolers on RTE-A:

- The LP spool system
- The SP spool system
- The RTE-A PRINT and PRIN0 programs

Refer to the *RTE-A Print and Spooling Reference Manual*, part number 92077-90248, for detailed information on the above spooling systems.

Error Logging (VC+ Only)

Error logging is a feature included in the spool system provided in a VC+ system. Any user can initiate logging by entering the following:

```
CI> sp lo on ErrorFile
```

The parameter *ErrorFile* is a file descriptor of the error log file. After error logging is initiated, all subsequent serious system messages (abort, I/O error, and so on.) and all logon/logoff transactions are entered in the error log file. If the system resources are not available, these messages are not entered in the log file. The messages that are to be displayed on the user terminal still appear at the terminal. The log file also includes a time stamp of each error occurrence.

To check the error log file, error logging must be stopped. This can be done with the following entry:

```
CI> sp lo off
```

After error logging is terminated, the log file can be listed with the LI command. Note that the spool status report indicates error logging if enabled.

System Setup Utilities

The following utilities, INSTL and FPUT, are used during the system installation process which is described in the *RTE-A Generation and Installation Manual*, part number 92077-90034. Refer to that manual for details on when to use these utilities. A detailed description of the utilities themselves is contained in Chapter 6 of this manual.

Initialize BOOTEX File (INSTL) allows you to initialize the boot extension (BOOTEX) file for an RTE-A disk LU.

Bootable System Installation (FPUT) installs bootable systems and diagnostics in the space on the file system volume that was reserved by a CI command.

The **Copy System (CSYS)** utility copies memory image files from disk to CS/80 cartridge tape to create a bootable memory-based system. Creating memory-based systems is described in the *RTE-A Generation and Installation Manual*. Refer to Chapter 6 of this manual for a detailed description of CSYS.

The **Absolute Binary to Memory Image (AB2MI) and Memory Image to Absolute Binary (MI2AB)** utilities convert absolute binary files to memory image files and vice versa. Refer to Chapter 6 of this manual for a detailed description of these utilities.

3

Manipulating Files

This chapter describes how to use CI commands to manipulate files, directories, and subdirectories. Table 3-1 provides a summary of the commands. Chapter 6 contains the detailed descriptions for the commands.

Also included in this chapter are descriptions of file properties; this information will help you take full advantage of the file system.

File Properties

Each file has certain properties associated with it. Some properties describe the way information is organized within the file, and others contain information about the file, such as its location, ownership, protection, and time stamps. The file properties are listed below and described in the following paragraphs:

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

File Names

Each file in a directory has a unique name, consisting of up to 16 characters, which distinguishes it from other files in the directory. (Duplicate file names may be used for files that reside in different directories.) The first character of each file name must be a letter or a number but names "0" through "255" are not allowed in order to avoid confusion with LU numbers that represent I/O devices.

A file type extension, consisting of a period followed by up to four characters, can be appended to a file name. Thus, a full file name, including a file type extension, can contain up to 21 characters. The following file name includes a file type extension that indicates it is a text file:

```
CurrentManualCh1.txt
```

File names can be entered in upper or lowercase letters, and capitalization is optional. CI always shifts the input to uppercase. We recommend that you avoid using characters other than letters and numbers in a file name. "Reserved" characters are those that have special meaning to CI

and/or cannot be used. For example, the slash (/), "at" sign (@), minus sign (−), left bracket ([), greater than sign (>), period (.), and comma (,) are reserved characters. The use of other punctuation characters should also be avoided. Although FMGR file naming does not have the same restrictions, note that problems could occur if you tried to move FMGR files whose names contained reserved or other punctuation characters to a new file volume.

It is possible to accidentally create a file name that has the high order bit in a character set. Such a name will print as a normal file name, but cannot be manipulated as a normal name, nor purged from CI. In this case, you must use FMP routines with the appropriate ASCII and non-ASCII characters to manipulate or purge the file name. FMP routines are described in the *RTE-A Programmer's Reference Manual*, part number 92077-90007.

Temporary Files

A temporary file is one that is intended to be used while a program runs and is typically purged after the program finishes executing. A temporary CI file is identified by a bit in the file's directory entry. A temporary FMGR file is identified by having a number as the first character in its name.

Temporary CI and FMGR files can be created using FMP routines. See Appendix F of the *RTE-A Programmers's Reference Manual* for a detailed discussion of temporary files, and to the discussion of FMP routines in that manual.

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 assigned at generation time to all devices in the RTE-A system. The LU numbers for devices such as terminals and printers can be used in most cases where a file name appears. To try using LU numbers to indicate I/O devices, use the CO command to copy a file. Because your terminal is always LU 1, you can display a file to your terminal as follows:

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

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

Table 3-1. File Manipulating Commands

Command	Task
CD [*directory*]	Change working directory
CL	List mounted disk volumes
CO <*src file*> <*dest file*>[*pram*]	Copy file
CP *file1* [*file2*, ...] *dest*	Copy file(s)
CR *file*	Create file
CRDIR <*directory name*>[*lu*]	Create directory on specified LU

Table 3-1. File Manipulating Commands (continued)

Command	Task
DC *lu*	Dismount disk volume
DL [*mask* [*option* [*file*|*lu* [*msc*]]]]	Display directory contents
IN *lu* [*blocks* [ok]]	Initialize disk volume
LI [*flags*]*file*|*lu*	List file or LU
LNS *file1* [*file2*, ...] *dest*	Create symbolic link(s)
LS *mask1* [*mask2*, ...]	Display directory contents
MC *lu*	Mount disk volume
MO <*src file*|*dir*> <*dest file*|*dir*>	Move file or directory
MV *file1* [*file2*, ...] *dest*	Move files and/or directories
OWNER *directory*[*newOwner*]	Display/reassign directory owner
PATH	Display current UDSP information
PATH [-E] [-N:*n*] *udspnum* [*dirname1*[*dirnam2*[...]]]	Display/define specified UDSP or UDSP entry
PATH [-E] -F,*file*|*lu*	Display/define UDSP using commands from specified file or LU
PROT *file* [*newprot*]	Display/modify file protection
PU *file*|*dir* [ok]	Purge file or directory
PWD	Display current working directory
RM *file1* [*file2*, ...]	Purge files and/or directories
RN *file*|*dir newname*	Rename file or directory
TOUCH *file1* [*file2*, ...]	Update the access time of a file
UNPU *file*	Unpurge file
WD [<*directory name*> [*file*|+S]]	Display/set up working directory with option for posting or changing command stack file
WHOSD *file*|*directory*|*lu*	Display the users of a file, directory, or LU

File Type Extensions

A file name may include a file type extension that indicates the type of information in the file, for example, text, binary data, or listing. The file type extension consists of a period and up to four characters appended to the file name. For example, in the file name parameter EDIT.RUN, the file type extension is .RUN. A blank file type extension is allowed, and is the default for some programs and commands. If you do not use a file type extension, you need not include a period after the file name.

Standard file type extensions should be used when files contain standard information. For example, all executable program files should have file type extension .RUN, relocatable files should have file type extension .REL, and all CI transfer files should have file type extension .CMD. The standard file type extensions are listed in Table 3-2.

Table 3-2. Standard File Type Extensions

File Type Extension	File
.c	C source file
.cmd	CI command file
.dat	Data file
.dbg	Symbolic Debug/1000 file
.dir	Directory or subdirectory entry
.doc	Document file
.err	Error message file
.ftn	FORTRAN source file
.ftni	FORTRAN source include file
.h	C include file
.hlp or .help	Help file
.lib	Library of relocatables
.lod	LINK command file
.lst	Listing
.mac	Macro source file
.maci	Macro source include file
.map	Load map list
.mlb	Macro library file
.mnf	Manual numbering file
.mrg or .merg	Merge file
.pas	Pascal source file
.pasi	Pascal source include file
.rel	Relocatable (binary) file
.run	Program file
.snp	System snapshot file
.spl	Spool system file
.stk	Command stack file
.sys	System file
.txt	Text file

When you specify a file, you must include the file type extension if there is one. If you specify only REPORT for a file named REPORT.TXT, you are implying a blank default file type extension, which does not match REPORT.TXT.

Some programs and program commands assume different default 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 manuals appropriate for the programs you are using for any default file type extensions (for example, the manuals for EDIT and DEBUG/1000).

File Descriptors

A file descriptor is a term used to specify a file by means of its attributes, including file size, type, record length, subdirectories, and directory. Colons are used to separate the parameters, and slashes are used to separate subdirectories, directory, and file name. The following two examples show file descriptor formats:

filename`::`*directory*`:`*type*`:`*size*`:`*record_length*

or

`/`*directory*`/`*subdirectory*`/`*filename*`:::`*type*`:`*size*`:`*record_length*

Note that the *filename* parameter includes the file type extension, if there is one. You must use a colon as a placeholder for each default parameter that is followed by another parameter.

The maximum length of a file descriptor is 63 characters, including delimiters, and we recommend that you keep file descriptors in the range of 40 characters, because FMP routines expand them to include unspecified attributes, which may cause them to exceed the limit. It is possible, however, to create files with file descriptors longer than 63 characters by using working directories or by renaming directories.

The *filename* parameter in the file descriptor can contain a mask qualifier that you can use to access multiple files. In addition, two wildcard characters, @ and −, can be used in the *filename* parameter. Refer to the "File Masks" section later in this chapter for details.

In the following examples of file descriptors, the file names in the user entries are shown in uppercase letters for clarity only. Directories and subdirectories in the comments are shown in uppercase letters as they are 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)
`PROGRAMMERS:::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)
`@.@::-16`	(All files on disk LU 16)

Directories

Directories contain files and other directories, called subdirectories. Directories maintain information about files including their names, file type extensions, all the optional properties defined for the files, and their locations. Many directories can be on one system, and each directory can have multiple subdirectories.

Each directory has a unique name of up to 16 characters, subject to the same rules as file names except that a global directory name can be a single integer. The directory name can be specified along with the name of a file you want to access, but its use is optional.

Directories can be specified as follows:

::<*directory name*> (or ::–*disklu* to refer to all directories on a disk LU)

or

/<*directory name*>

In the "::<*directory name*>" format above, the directory name is preceded by two colons, which separate the file name from the directory name. This form is generally used with FMGR files and the system may display this form for compatibility with FMGR files. The field between the colons is used by FMGR files to define an optional file security code; for example, file:sc:crn.

The "/<*directory name*>" format is typically used if the files are organized in a hierarchical structure. Such a file structure may contain directories that have nested subdirectories. This form of specifying files is used to indicate the search path for the files in the CI, or hierarchical, file structure. Figure 3-1 illustrates the hierarchical file organization.

If the directory name is omitted in a file descriptor, a default directory called the working directory (WD) is used. The working directory can be defined or changed with a WD command. Once defined, the working directory remains in effect until changed by another WD command. You may display the name of the working directory by using the WD command without a parameter.

If the optional VC+ (HP 92078A) is used in your system, the multiuser logon system can set up your working directory automatically whenever you log on.

Certain programs contain a special feature that lets you schedule other programs without specifying the directory name. If you omit the directory in the program runstring, standard directories set up by the system are searched. For example, in executing the RU (Run) command, CI searches 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.

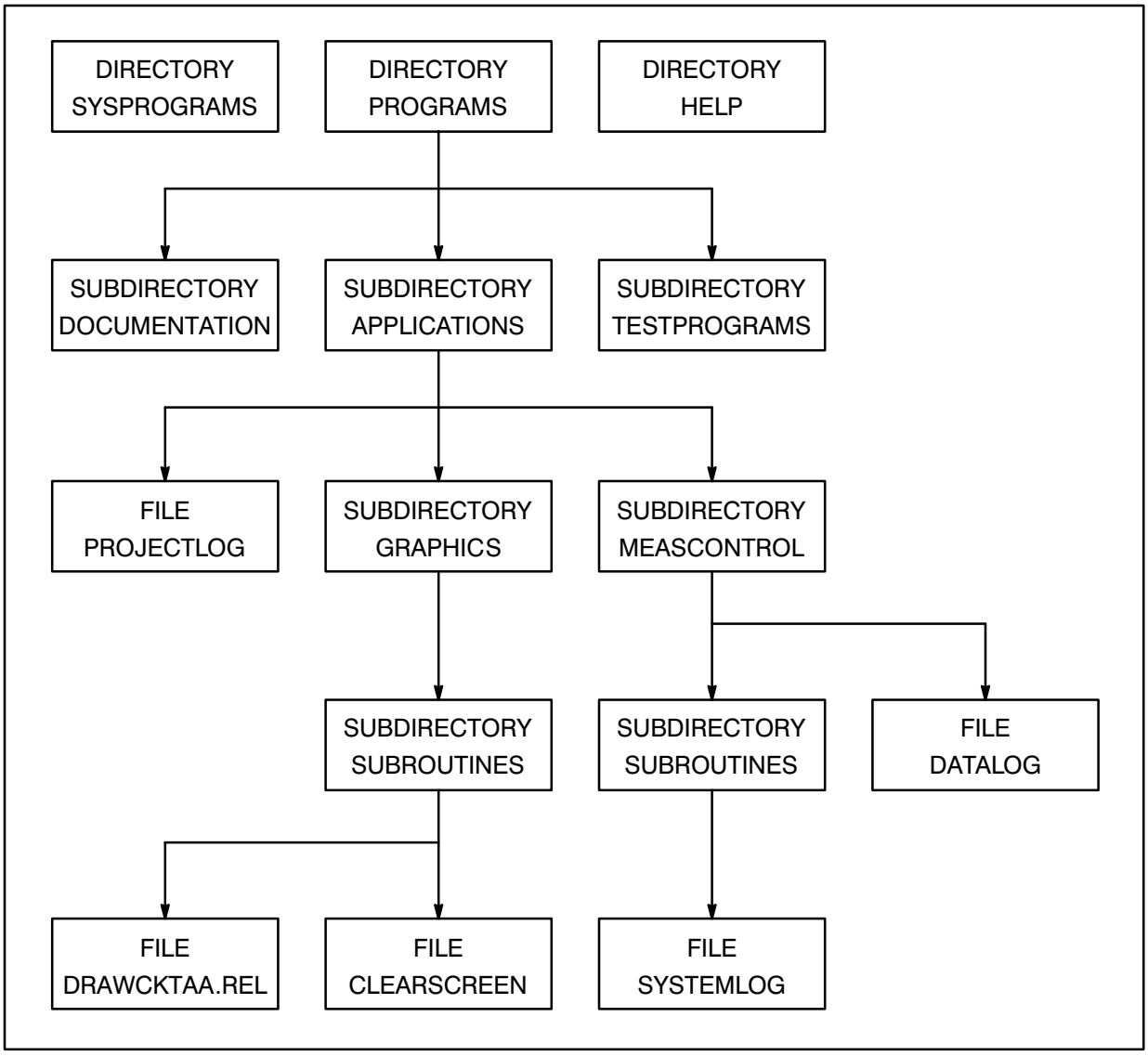


Figure 3-1. Sample Hierarchical File Organization

Subdirectories

Subdirectories are directories that are contained or embedded within other directories. Subdirectories in turn can contain other subdirectories, and there can be many levels of subdirectories. Unlike directories, subdirectories can have duplicate names as long as all names within a single directory are unique.

Subdirectories have the same properties as directories, and in this manual, references to directories also apply to subdirectories, unless otherwise noted.

When you want to specify a file that is in a subdirectory, use the hierarchical format, preceding the file name with the subdirectory name and separating the two names by slashes. For example, if a file named MANUAL.TXT is in directory /DIR, it can be specified as follows:

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

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

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

The first form above is used when there are no subdirectories. The second form is used to specify a search path in a hierarchical file structure in which there may be many levels of subdirectories. There is no limit to the number of levels of subdirectories that 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, enter the following to specify a file named DRAWCKTAA.REL located 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, the following form is used:

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

The hierarchical file structure provides a search path to minimize the search time and allow duplicate file names for files that reside in different directories or subdirectories.

In a hierarchical file specification, a directory name is always preceded by a leading slash; without the slash, the name is assumed to be a subdirectory. For example,

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

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

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

specifies that FILE.TXT is in subdirectory ARCHIVE of subdirectory SYSTEM. (Because no slash precedes the name SYSTEM, it is assumed to be a subdirectory.) In this case, because a directory is not specified, the working directory is assumed, which means SYSTEM is a subdirectory of the working directory. In other words, the entire search path is different.

Directory Specifiers "." and ".."

Two other specifiers can be used to indicate the directory path to a file and are useful for moving around the directory tree without explicitly specifying subdirectory names. They are:

. – identical to "the current working directory"
.. – identical to "the parent of the current working directory"

These characters are used in place of specific directory names in a file descriptor.

For example, assume the directory structure is as shown in Figure 3-1 and the current working directory is /PROGRAMS/APPLICATIONS/GRAPHICS. To access the file PROJECTLOG, you can use the file descriptor

```
/PROGRAMS/APPLICATIONS/PROJECTLOG
```

or you can enter

```
../PROJECTLOG
```

Here the leading ".." is identical to "/PROGRAMS/APPLICATIONS" (the father directory of GRAPHICS). Similarly, the file DATALOG can be accessed using

```
../MEASCONTROL/DATALOG
```

Multiple sets of ".." can be used to continue up the directory tree. For example, if the working directory is

```
/PROGRAMS/APPLICATIONS/GRAPHICS/SUBROUTINES
```

the file SYSTEMLOG can be reached with the following descriptor:

```
../../MEASCONTROL/SUBROUTINES/SYSTEMLOG
```

Here the first ".." refers to the parent of SUBROUTINES (GRAPHICS) and the second refers to the parent of GRAPHICS (APPLICATIONS). The remainder of the descriptor continues down from that point. An equivalent descriptor is

```
/PROGRAMS/APPLICATIONS/MEASCONTROL/SUBROUTINES/SYSTEMLOG
```

You can string the ".." characters together until the top level of the directory tree is reached. At that point, additional ".." sequences have no effect. For example, from /PROGRAMS/APPLICATIONS, the descriptor

```
../../../../../DOCUMENTATION
```

is identical to /PROGRAMS/DOCUMENTATION.

The "." character refers to the working directory, and the descriptor "./xxx" is basically equivalent to the descriptor "xxx". The reason for using "." instead of just specifying the file alone is that

some commands and applications treat the two descriptors differently. For example, the command

```
ru prog.run
```

causes PROG.RUN to be picked up from the working directory or some other directory in the programs search path (refer to the discussion of the RU command for details on the search path). But the command

```
ru ./prog.run
```

forces PROG.RUN to be picked up only from the working directory; that is, it disables the search algorithm.

Directory Specifier "#n"

You can use another leading character sequence,

```
#n
```

where n is a number from 0 to 8, in place of the directory name in a file descriptor. This instructs FMP to search for the specified file using UDSP number n. For example, the file descriptor

```
#1/prog.run
```

tells FMP to search through the directories defined in UDSP number 1 to find the file PROG.RUN. (UDSPs are further described later in this chapter and in the discussion of the PATH command in Chapter 6.)

Note that some commands and applications use a default UDSP when searching for files; for example, the RU command uses UDSP #1. This sequence overrides the default UDSP. For example, the command

```
ru #4/prog.run
```

causes a search for the file PROG.RUN within UDSP #4 instead of the default UDSP #1, which is normally used for the RU command.

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 a file type extension. There are standard RTE-A file types defined with the following characteristics:

Type −1	Symbolic link files. A symbolic link is a file that indirectly refers to another file. The symbolic link file itself contains a file descriptor that points to the new file. The LNS command, described in Chapter 6, is used to create a symbolic link.
Type 0	An I/O device. Type 0 is used in accessing devices with file calls. There is no disk file or directory entry for type 0 files, and they do not have the other properties listed in this section.
Type 1	Random access files. These 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	Fixed-length record, random access files. The record length is defined when the files are created. They are usually user-created, large data files.
Type 3	Type 3 and higher files are variable-length and higher record, sequential files suitable for use as text files. There is no difference in the handling of file types 3, 4, and 7. 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 4 is recommended for text files. By convention, type 5 is used for Compiler or Assembler relocatable output files, type 6 is for program files that are memory-images of executable programs, and type 7 is for Compiler or Assembler absolute binary output files. Type 6 files are treated the same as type 1 files.
Type 8	Type 8 and higher files are user-defined, with the following exceptions.
Type 12	Byte stream files. These do not have any structure information in them. The directory entry for each type 12 file contains a pointer to the last byte in the file. The fields for number of records and record length are not defined for type 12 files.
Type 6004	CALLS catalog files.

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 can create a type 1 file with the following entry:

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

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

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

File Size

The file size parameter in the file descriptor specifies how many blocks of disk space the file needs. One block is 128 words (256 bytes or characters). One printed page takes about 10 blocks of disk space. You can specify how big a file should be when you create it. If you do not specify the size 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, for example, when you create 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. For example, to create a file of 100 blocks, enter the following:

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

To specify both size and type, enter the following:

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

Except for type 6 and some type 1 files, the file system automatically increases the file size to accommodate more data as needed through a process called “extending the file.” “Extents” are always at least as big as the original file size, because there are performance advantages to having fewer, larger extents. Type 6 and some type 1 files cannot be extended, because they are memory images of programs of the RTE system.

Files that are larger than 16383 blocks are rounded off to multiples of 128 blocks. Files can be as large as any disk available in your system. Files larger than 16383 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 is specified by a size of $-50000/128 = -391$, rounding to the nearest larger number in absolute value. Large files are usually created by a program and rarely by a user.

Be aware that a file size parameter larger than 32767 blocks will be accepted, but the desired file size will not be created. For example, the following command creates a file with 13110 blocks:

```
CI> cr file::::36214
```

Record Length

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

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

This field is also used in type 3 and higher 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 you in type 3 and higher files is ignored.

File Ownership and Associated Group (VC+ Only)

A directory's owner is the user who created it. All files in a directory are considered owned by the directory owner. The associated group of a directory is the associated group of the directory owner. The same is true for subdirectories. However, the owner and the associated group of a subdirectory can be different from the owner and associated group of the directory that contains the subdirectory.

The directory owner can change the protection status of files in that directory. The protection status defines the read/write access allowed for the owner, members of the associated group, and general users. See the section on "Protection" below for more information. The directory owner (and no other user) can also reassign the directory ownership and the associated group.

An entire CI volume can have an owner and associated group. The initial owner of a volume is the user who initialized the volume, and the associated group is that user's associated group. A volume's ownership and associated group may be reassigned using the owner command.

See the section on "Manipulating Directories" in this chapter for more information about ownership, associated groups, directories, and subdirectories.

Protection (VC+ Only)

File protection is a security measure in the RTE-A file system that governs read and write access. It is defined when a file is created or copied into a directory, and it can be specified differently for the owner, members of the associated group, and general users. The default file protection provides both read and write access for the owner and read access only for members of the associated group and other users.

When a file is created, it assumes the protection status defined for the directory on which it resides. A copied file assumes the protection status defined for that file if one exists; otherwise, it assumes the protection of the directory into which it is copied.

You can specify protection on a file-by-file basis, in any combination of read and write access for the owner, members of the associated group, and general users.

Directories also contain protection information that governs the protection status of any file created in that directory. For example, if a directory allows read and write access for the owner and read access only for members of the associated group and general users, all files created in that directory have that same protection status unless it is changed by the PROT command.

Read and write protection for a directory differs slightly from that for a file. Directories that are write-protected (read access only) prevent general users from changing information in the directory through CI commands; they cannot create, purge, or rename files in the directory. Directories that are read-protected prevent general users from finding out the contents of those directories.

A CI volume also has a protection status that regulates the reading, creating, purging, or renaming of the global directories on that volume.

Time Stamps

Time stamps are maintained for all files in the RTE-A file system except those created with FMGR. 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 examples that follow illustrate the use of time stamps.

```
CI> dl <file descriptor> a       (Examine time last accessed)

CI> dl <file descriptor> c       (Examine time created)

CI> dl <file descriptor> u       (Examine time last updated)

CI> dl <file descriptor> uac     (Display all three time stamps)
```

The creation time of a file is set when the file is created. Access time is set whenever a file is opened. Update time is set whenever a file is written, whether the data in the file has been modified or not. Because the update time is not set until a file is closed, the update time of an open file is not accurate.

Examining the directory information for a file has no effect on the file's access or update time.

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, remains the same as that of the existing file, thus preserving the revision history of the file.

File Masks

Access to multiple files is simplified when you use a file mask, which lets you specify several files with a single entry, using one or more of the fields in the file descriptor. For example, the daily entries of a system log can be accessed by masking the date code in the file names. Alternatively, all those files can be accessed by specifying "syslog−−−−−−".

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

In this case, the dash (−) is used to mask one character position (except a blank character).

The @ character is used to mask all characters. Thus, 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 security code (FMGR files only)
file type
file type extension
file size
file record length
time stamps
user.group
backup status

The "." and ".." alternate directory specifiers may be used in file masks just as in regular file descriptors. However, the #n specifier may not be used.

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.

You can use an LU number as the file name, but the type extension will be ignored. There must be a placeholder for the type extension to separate it from the mask qualifier. For example, the following command lists all files on LU 16 that have the backup bit set:

```
CI> dl 16..b
```

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 to facilitate finding the desired files, as shown in Table 3-3.

Table 3-3. Mask Qualifiers

Characters	Description
(user.group)	Mask by specified user. Return the files belonging to a given user, files in a specified group, or files belonging to a given user regardless of group.
a	Access time stamp mask (see the section on “Time Stamp Masks” in this chapter).
b	Match only those files that need to be backed up. (Refer to the discussion of the TF utility in the *RTE-A Backup and Disk Formatting Utilities Manual*.)
c	Create time stamp mask (see the section on “Time Stamp Masks” in this chapter).
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 the mounted disk volumes in the system, including the FMGR file system disk cartridges. (This can take a long time, depending on the size, contents, and the number of volumes.)
g	A search directive. When used with ‘d’, ‘k’ or ‘s’ directives, masking will not search symbolic links to directories.
k	A search directive. Search from the specified directory down through any subdirectories in that directory, applying the mask to all files within the search path. ‘k’ is similar to ‘d’ in that it preserves the directory structure (for example, in a copy) and similar to ‘s’ in that it applies the mask to each file in the path. ‘k’ overrides or cancels a ‘d’ qualifier.
l	Return the directory information for a symbolic link file rather than the directory information for the file pointed to by the symbolic link.
m	Return extent entries on FMGR directories.
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.
u	Update the time stamp mask (see the section on “Time Stamp Masks” in this chapter).

Table 3-3. Mask Qualifiers (continued)

Characters	Description
w	Walk through FMGR directories. FMGR directories are written on the disk in a staggered fashion. They must be accessed in the same staggered fashion to find files in the order that they appear in the directory. This is known as walking. An application in which the order of file access is not important can gain performance by accessing the directory in a faster, non-staggered manner known as running. The masking routines use the fast way unless the 'w' qualifier is set or the buffer area supplied is 8K words, in which case no speed is gained by running.
x	Match only files with extents (not applied to FMGR files).
y	Return correct extent information on directories (requires an additional disk access for each directory).

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, or as a range. The format is as follows:

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

where the xxxxxx.xxxxxx is a date and time in the format YYMMDD.HHMMSS; thus, 920529.120000 is noon May 29, 1992. Only one choice among c|a|u (create, access, or update) is allowed. The default is the last update time. The dash character "−" is not a mask character when used in the qualifier field, but is used to specify a range of dates. If "−" is not used, the specification is for files that match that date/time. For example:

```
c880416.121108-910611.141411
```

This entry specifies files created between April 16, 1988 12:11:08 and June 11, 1991 14:14:11. The time can be specified with as few digits as desired. Thus "`a89-91`" specifies files last accessed during or after 1989 and before or during 1991.

The time stamps in the file system begin on Jan 1, 1970. Dates specified in years between 00 and 37 (inclusive) are interpreted as being the year 2000 through 2037. Time stamp values do not go beyond 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, only purged files qualify.) If the file type extension is not specified, only files with blank type extensions match. If the directory and subdirectory are not specified, only the working directory is used. The directory and subdirectory specification has precedence over the e option (if both are specified, only the specified directory is searched).

There are several special cases in specifying directories using file masks. If the mask ends with a slash (/), as in /FOO/JOE/, it is 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.

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> dl /foo/
```

Note that /FOO will not list the files in directory FOO, but the following entries will:

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

If the file name in a mask ends with the wild card character (@) and the file type extension is not specified, 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, only subdirectory JOE in global directory /FOO is matched. 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.su82`	(Search all CI volumes 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)

Examples using the (user.group) mask qualifier:

`dl /@.@.s(user)` or `dl /@.@.s(user.@)`	(Return the files belonging to user, regardless of group, in all CI directories and subdirectories)
`dl @.@.(user)`	(Return the files belonging to user in the working directory (will be all or none as the same user owns all files in a directory)
`dl /@.@.s(user.)`	(Return the files belonging to the user with associated group NOGROUP in all CI directories and subdirectories)
`dl /@.@.s(@.)`	(Return the files with associated group NOGROUP in all CI directories and subdirectories)
`dl /@.@.s(user.group)`	(Return the files belonging to the user in specified group and search all CI directories)

Examples using search directive qualifiers:

`co /A/B@.@.k /SCRATCH/`	(Copies all subdirectories and files matching B@.@ within the directory /A to the directory SCRATCH, preserving the directory structure)
`co /A/@.@.k @`	(Copies all subdirectories and files within the directory /A to the working directory, preserving the directory structure)
`co A/@ @`	(Copies all subdirectories and files within the directory /A to the working directory, preserving the directory structure because the CO command implies use of the 'd' search structure)

Examples using symbolic link qualifier:

`DL @.@:::4`	(Display all type 4 files and symbolic links that point to type 4 files)
`DL @.@.l:::4`	(Display all type 4 files)
`DL @.@.l:::-1`	(Display only symbolic link files)

Other examples:

`dl @.txt`	(Display files in the working directory with file type extension TXT)
`dl a@.@.c83`	(Display files in the working directory that start with a, created during 1983)
`dl /joe/@.@.sc80-83`	(Display files in directory /JOE created during the period between 1980 and 1983. Also, the s qualifier directs a search of any subdirectories of directory JOE for similar files)

Masking and FMGR Files

For FMGR files, the dot ".", is both a legal character and a delimiter in the mask. If necessary, the masking routines modify masks as follows for FMGR files with dots in their names before applying the masks to the FMGR files.

- The name and type extension are examined as if they referred to a CI file.
- If there is no dot in the mask, no change is made.
- If the type extension is null (that is, there is none), the dot is replaced by a "−" (minus) character. For example, "xyz." becomes "xyz−".
- If the type extension is "@", the dot is changed to "@". For example, "xyz.@" becomes "xyz@@". A CI file name represented simply as "@" (which is equivalent to "@.@") is modified to "@@@".
- Otherwise, the type extension is put back with the name. For example, "xyz.w" remains "xyz.w".

All FMGR files with a single dot in their names will be found, but additional files may also be returned. When defining masks for FMGR files with names containing multiple dots, replace all but one dot with a minus (−) character.

Destination File Masks

The CO, CP, MO, MV, LNS, and RN commands can 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`" renames all the files on the current directory that 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 is used for the destination file name or file type extension. If either is defaulted using the @ character, the name or file type extension of the source file is 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. The .DIR file type extension cannot be changed in the destination file descriptor. If the source file type extension is .DIR, 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 results in the default name and file type extension.

For the type and record length fields, the values from the source file are always 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 consists of both the destination mask and the source file directory path, starting with the destination mask, to which 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.

Copy all files in subdirectory /PROGRAM/DOCUMENTS into subdirectory /MANUAL/DOCUMENTS.

```
CI> co /program/documents/@ /manual/documents/@
```

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

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

In this example, the d qualifier in the source mask specifies all files in the directory /DOCUMENTS. The subdirectory will be created if it does not exist.

Copy these files to a subdirectory called /MANUAL/CHAPTERS, changing the subdirectory name at the destination.

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

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

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

The destination subdirectory in this example will be created if it does not exist.

More examples are as follows:

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

CI> co main.lst @.temp           (Copy MAIN.LST to MAIN.TEMP)

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

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

File Operations

The following sections describe the file operations you can perform using CI commands. The CI commands that are used here are documented in more detail in Chapter 6. The tasks performed by the DL, CO, MO, PU, and CR commands described here can also be performed in a similar manner by the LS, CP, MV, RM, and TOUCH commands, respectively. The latter commands, which have been provided as of RTE-A Revision 6000, are similar to UNIX* commands. These commands are also described in detail in Chapter 6. Note that the CP, MV, RM, and TOUCH commands are supported on VC+ systems only.

Directory Listings

One of the most useful file operations is listing the files in a directory, which shows the files available to you. The directory list command is DL. Entering DL without any parameters returns a list of file names in your working directory, sorted in alphabetical order. For example, to list all files in the working directory:

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

Here 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. Such names are not recommended for important data.

DL can also be used to get a list of the files contained in another directory simply by specifying the name of the directory. There are several ways to list the contents of a directory, as shown below.

To list all files in directory named JONES:

```
CI> dl ::jones                    (Recommended for FMGR files)

CI> dl @.@::jones

CI> dl /jones/                    (Recommended for hierarchical files)

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

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

```
CI> dl /jones/subdir/

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

This gives the names of the files contained in those directories. The trailing slash must follow the directory or subdirectory to get the desired effect.

See also the LS command described in Chapter 6.

---

*UNIX® is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

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> li /system/welcome.cmd
```

In this example, the file is displayed on the terminal screen in pages of lines separated by the following prompt:

```
More...
```

After the first use of the LI command your prompt may look like this:

```
More [x%] ...
```

where *x*% represents the percentage of the file already listed.

You can respond to this prompt by entering a single character, optionally preceded by a number, from 1 to 32767 called *n* below, to select from the following standard options:

Character	Action
<space>	List another page, or another *n* lines, if given.
<return>	List the remaining lines without page breaking.
A or Q	Abort the listing.
+	List one more line or skip *n* lines and list one more line.
P	Set page size to *n* lines and list another page.
Z	Suspend the LI program (restart with the system GO command).

The abort character "a" can be either upper or lowercase. After it is entered, the listing stops and the CI prompt is displayed.

Several other commands use this method of display, pausing after each screenful to let you read what has been displayed with the same choices for abort or continuation.

The LI command also provides many more paging functions and options than the standard choices shown above. If you are using a printing terminal, you can use the −0 option to copy the file to the printing terminal without stopping. Refer to the LI command description in Chapter 6 for details of all the LI options.

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

```
File:  filedescriptor, list? [Y]
```

Respond to this prompt by entering a single character:

Character	Action
Y or <space>	Yes, list this file.
N	Skip this file, move to the next file.
A or Q	Abort the LI command.

LI has many options for display formatting and file filtering; see Chapter 6.

Copying Files

The CO command can be used to make a copy of any type of file and to copy files to or from I/O devices.

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

```
CI> 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 specified 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 and 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 6 of this manual. Following are more examples of file copying entries.

Copy /SYS/REPORT1 to the working directory:

```
CI> co /sys/report1 report1
```

Note that the destination file uses the default working directory and is not defaulted to the source file directory.

Copy a file to an existing file on the working directory:

```
CI> co file1 masterfile d
```

'd' is the replace duplicate option; the current MASTERFILE is to be purged if it exists.

Copy a file to magnetic tape (LU 8):

```
CI> co file 8
```

Copy a file to the terminal screen (LU 1):

```
CI> co file 1
```

See also the CP command described in Chapter 6.

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> 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 6 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 is an enormous file, this takes a long time, and there must be enough disk space for both copies.

You can use the MO command to move the file to a different directory if both directories are on the same LU. If the directories are on different LUs, an error is returned. You must use the CO command to copy the file to the new directory and the PU command to purge the original file. Or you can use CO with the P option to purge the source after the copy. Before moving a file between directories, you can use the DL command with the L option to display the LUs on which the directories reside.

The following example moves FILE1.TXT from directory /SMITH to directory /JONES:

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

See also the MV command described in Chapter 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 using a file mask. You must have write access to the directory containing the files to be purged.

To remove FILE1.TXT, enter the following:

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

This removes FILE1.TXT from the working directory. The disk 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 disk area occupied by the purged file has not been overwritten, the file can be recovered with the UNPU command. This is very useful if you inadvertently purge the wrong file.

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

For example:

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

If you choose the Stop Asking option, the prompt is not displayed again; only a message indicating that the file is being purged is displayed.

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

```
CI> 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, NOT the file protection, when purging files.

See also the RM command described in Chapter 6.

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> unpu file1.txt
```

There is no particular limit to the length of time that a purged file remains recoverable. It depends on random factors such as the number of files being created and the position of the file on the disk 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 cannot be recovered, the file has been overwritten.

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. You can unpurge all of the files of the same name successively by unpurging and renaming them one at a time.

Creating Empty Files

Empty files can be created with the CR command. The file space specified for these files is reserved 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> 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> cr file.dat::system:1`	(Create a type 1 file)
`CI> cr /system/subdir/file.txt:::1`	(Create a type 1 file in a subdirectory)
`CI> cr file.mnl:::1`	(Create a type 1 file in working directory)
`CI> cr /system/bigger.dat::::100`	(Create a file of 100 blocks)

See also the TOUCH command described in Chapter 6.

Change File Protection (VC+ Only)

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 or by a superuser.

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

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

FILE.TXT    rw/r/r              (File is write protected from members of
                                 the associated group and 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, then the status for members of the associated group, followed by a slash, then the general user status.

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

```
CI> prot report rw/                 (Read and write allowed for owner only)

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

CI> prot testdata.txt rw/rw/r       (Read/write for owner and group; read
                                     for others)

CI> prot memo rw/r/                 (Read/write for owner; read for group;
                                     no access for others)
```

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

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

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.

Creating Symbolic Links (VC+ Only)

Symbolic links can be created with the LNS utility. Symbolic links can refer to FMP files, FMP directories, or devices. A symbolic link may also be used to point to a remote file or directory. The file pointed to in the symbolic link does not need exist at the time that the link is created. LNS overwrites existing files when the user has write access to the existing file and its directory.

To create a symbolic link called /HELP/FGREP that points to GREP:

```
CI> lns grep /help/fgrep
```

When the user does not have write access to an existing file, LNS will prompt the user before overwriting an existing file. To create a symbolic link named /DEV/MAGTAPE that points to LU 8:

```
CI> lns 8 /dev/magtape
override protection (r-/r-/r-) for /DEV/MAGTAPE? (y/n)
```

The −I option can be used to cause LNS to issue a prompt before overwriting an existing file. If the response begins with a 'Y' or a 'y' and the user has write permission to the directory, the existing file is overwritten. To create a symbolic link to the /SCRATCH directory in the subdirectory SCR.DIR :

```
CI> lns -i /scratch/ scr/
override protection (rw/rw/rw) for SCR/SCRATCH.DIR? (y/n)
```

The −V option can be used to cause LNS to display each symbolic link as it is created. To create a symbolic link to a remote source file, /SRC/PROG.FTN>SOURCES, in the current directory :

```
CI> lns -v /src/prog.ftn>sources ./
lns: ./PROG.FTN -> /SRC/PROG.FTN>SOURCES ... [ok]
```

Refer to the LNS command description in Chapter 6 for more information.

Manipulating Directories

Directories can be thought of as system files with which only the operating system is concerned. Each directory contains information about the files that are in the directory, although the data in the file itself is not in the directory. File data is kept elsewhere on the disk volume. (Volumes are described separately in this chapter.) Directories have an initial size, and they are automatically extended to hold more files as necessary. When files are purged, the directories are not truncated; the space previously occupied by the purged files is 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 file 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 file type extension when dealing with directories because it is implied by the way the name is used. For example, .DIR 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, as the file type extension .DIR cannot be used in front of a slash.)

Operations involving directories include creating directories, changing the working directory, listing directories, renaming directories, purging directories, and examining and changing directory ownership and protection. These are all discussed in the following sections.

Creating a Directory

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

```
CI> crdir /system
```

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

```
CI> crdir /system 12
```

This creates directory /SYSTEM on disk volume LU 12. To find out what disk volumes are available, use the CL command. In this example, because 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:

```
RW/R/R          (Read and write allowed for owner and read only for other users)
```

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

Note that the following command does not create a global directory; rather, it creates a subdirectory called SYSTEM in the current working directory.

```
CI> crdir system
```

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> crdir /system/subdir
```

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

The difference between specifying subdirectories and directories is that a leading slash is used for a global directory, while none is used for a subdirectory.

Display/Setup Working Directory

The working directory is searched first by the file system when it searches 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> 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> wd games
```

Here GAMES, a subdirectory in the current working directory, becomes the working directory.

```
CI> wd /games/rules
```

In this case, subdirectory RULES is a working directory.

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

```
CI> wd 0
```

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

The WD command can also be used to post the contents of the command stack to the associated file or to change this file. The default command file is CI.STK on the working directory. To post the contents of the stack to the default command stack file, enter:

```
CI> wd,,+s
```

CI.STK is opened and command stack posted there.

This can also be done as you change the working directory. For example:

```
CI> wd /debbie +s
```

The above command causes the file associated with the command stack to be posted with the contents of the stack. Then the command stack is overwritten with the contents of

/DEBBIE/CI.STK or cleared if the file does not exist. You can specify any file to be associated with the command stack. For example:

```
CI> wd,,cmdstackfile.stk
```

In this case, the command stack is posted to the current command stack file if one exists; if it does not exist, it is created. Then the contents of CMDSTACKFILE.STK are placed into the command stack. If this file does not exist, the command stack is cleared. At logoff, this file is posted with the contents of the command stack and closed.

See also the CD and PWD commands described in Chapter 6.

Moving Directories

The directory path of a file or subdirectory can be changed by the MO command, which is especially powerful in manipulating directories. MO 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> 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.

See also the MV command described in Chapter 6.

Displaying Directory Owner (VC+ Only)

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

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

Changing Directory Owner and Associated Group (VC+ Only)

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

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

Use this command with caution. Once the ownership is changed, you are no longer the owner and may not have 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.LAB can change the ownership and associated group. The subdirectories within directory /SYSTEM are not affected.

If you do not specify an associated group in the OWNER command, the associated group becomes the default logon group of the user who is the owner. For example, assume the group LAB is the default logon for JONES. The following command

```
CI> owner /system jones
```

changes the owner and associated group of the directory /SYSTEM to JONES and LAB, respectively.

If you do not want a directory to have an associated group, you can specify NOGROUP. This also turns off group protection in the directory, as shown in the example that follows.

```
CI> owner /temp
Owner of /TEMP is JONES.LAB                (An associated group is defined)

CI> prot /temp directory /                 (Group protection is on)
name          prot
TEMP.DIR      rw/r/r

CI> owner /temp jones.nogroup              (Making no associated group)

CI> owner /temp
Owner of /TEMP is JONES                    (No associated group is defined)

CI> prot /temp directory /
name          prot
TEMP./DIR     rw/r

CI> owner /temp jones.accounting           (Defining an associated group)

CI> owner /temp
Owner of /TEMP is JONES.ACCOUNTING

CI> prot /temp directory /                 (Group protection is on again)
name          prot
TEMP.DIR      rw/r/r
```

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> pu /games
```

Note that the form ::GAMES cannot be used because this is interpreted by PU as all files in directory GAMES. PU will purge them all 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> pu /system/sub.dir
```

See also the RM command described in Chapter 6.

Display/Change 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.

The following examples show displaying protection status:

```
CI> prot /system                    (For directory /SYSTEM)

CI> prot /system/                   (For all files in directory /SYSTEM)

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

CI> prot /system/data/              (For all files in subdirectory DATA)
```

To change the protection for a directory (/SYSTEM):

```
CI> prot /system rw/rw/rw           (Read/write access for everyone)
```

To change the protection for a subdirectory (DATA):

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

Searching for Files

When you enter a file-referencing CI command, CI checks to see if you specified a directory; if so, 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 UDSPs #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 following search sequence is used to locate the file:

1. 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.

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

3. 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 (VC+ Only)

User-Definable Directory Search Paths (UDSPs) 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> path 2 . /jones/utilities/cmds /cmdfiles
```

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

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

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

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

UDSP #1, which is used by the RU command, can be set to use 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/PROGS
4. /PROGRAMS

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

```
CI> path 1 /MINE/PROGRAMS . /MINE/MORE/PROGS /PROGRAMS
```

Refer to the description of the PATH command in Chapter 6 for more details.

Specifying UDSPs in File Descriptors (VC+ Only)

Just as UDSPs #1 and #2 are used by the TR and RU commands by default, it is possible to explicitly request a UDSP search sequence for a particular file. For example:

```
CI> li #2/comp.cmd
```

requests the LI command to search through UDSP #2 to find COMP.CMD. This lists the same file that the TR command finds.

The sequence #N/FILE may be used with the FmpOpen routine and with any command that opens the file, except where a file mask is required (such as with DL).

Manipulating Volumes

Each physical drive consists of one or more volumes. A volume is a self-contained section of a disk, independent of any other volume. A volume is always identified by its disk LU number, from 1 to 63. Volumes never cross physical drives, and files and directories never cross volumes.

Each volume contains a unique set of information about the files on it, including the names of all the global directories on the disk, as well as a table that tells which disk 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, changing ownership and protection, and listing contents of a volume. An operation that is infrequently performed is initializing a volume, making it ready for system use.

Mount/Dismount Volumes

Mounting a volume makes that volume and all the files on it available to the file 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 disks, which must be mounted after they are installed and dismounted before they are removed.

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

```
CI> mc 12
```

If the disk volume has a valid FMP or FMGR directory, the volume is mounted. If the disk volume does not have a valid FMP or FMGR directory, you are prompted to confirm that the volume should be initialized. This is done to avoid the accidental corruption of volumes that are not FMP or FMGR types (for example, backup utility volumes).

Initializing a volume sets up information needed by the operating system, including the list of directories and the bit map for keeping track of disk space usage.

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. If duplicates occur, the names of the duplicate directories are displayed. If you need the new ones, you can rename the duplicate directories already mounted, then dismount and remount the volume.

For example, to dismount volume LU 12:

```
CI> dc 12
```

Before you dismount a volume, make sure there are no open files, working directories, restored programs, or directories that are part of a current user's UDSPs on that disk volume. Otherwise, when you try to dismount the volume you will get an error message each time an error is encountered and the dismount command will be aborted.

The DC command shows 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 disk LU:

```
CI> WH,AL                    (Check all RP'd programs)

CI> DL lu o                  (Check for opened files. This lists all files on the
                              disk volume, which can take a long time if there
                              is a large number of directories and files.)

CI> WD 0                     (Remove the working directory. This command
                              must be used for each user who has a working
                              directory on that LU in a multiuser
                              environment.)

CI> WHOSD dir|lu             (Check for any session accessing a specified
                              directory or LU as a working directory or as
                              part of a UDSP)
```

Volume Ownership and Protection

Volumes have owners, associated groups, and protection as do directories. The original owner of a volume is the user who initialized it. The protection of a volume governs the accessibility of global directories on that volume.

Ownership is displayed and changed with the OWNER command. For example:

```
CI> owner 12v                (Display the owner of volume 12)

CI> owner 15v fred.lab       (Change the owner of volume 15 to FRED
                              and the associated group to LAB)
```

Protection is displayed and changed with the PROT command. Only the owner of a volume can change the protection. For example:

```
CI> prot 10v                 (Display protection of volume 10)

CI> prot 13v RW/R/R          (Change the protection of volume 13 to rw/r/r.
                              Note that with the new protection, only the
                              owner may create global directories on volume 13)
```

Listing Volumes

The CL command is used to list the volumes that are currently mounted. The CL command has no parameters. It provides a list of two types of volumes: those mounted as described above and those mounted as FMGR cartridges (as discussed in the section on "FMGR Files" in this chapter). Unmounted volumes are not listed.

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

Initializing Volumes

Initializing a volume prepares it for first-time system use. This function is done automatically by the MC command, but the IN command can also be used on a mounted volume. The IN command can be used to remove all the data on a volume without having to purge all the files first. Initializing a disk volume permanently destroys any existing files, so be certain that the files on that LU are no longer needed. In a VC+ environment, this command can be used only by the superuser.

For example, to remove all data on volume 12 without dismounting it, enter the following:

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

File Manipulation Utilities

The following are the file manipulation utilities available for use in the CI file system. Refer to Chapter 6 for the detailed description of each utility.

Concatenate Files (MERGE) collects two or more input files and combines them in a single composite output file.

Extended Record Converter (OLDRE) is a standalone utility that provides backward compatibility with loaders and generators that do not accept extended record formats.

Compare Files (SCOM) compares two input files and identifies their differences.

File System Utilities

The following are the file system utilities provided for use in the CI file system. Refer to Chapter 6 for the detailed description of each utility.

File System Conversion (FSCON) converts the directory structure of a FMGR cartridge to a hierarchical directory entry. Once converted, FMGR files have the characteristics of the hierarchical file system.

File System Pack (FPACK) rearranges the files on a file system volume to pack them together more tightly. This allows more or larger files to be created.

File Compacting and Disk Pack (MPACK) compacts files and packs a disk, removing unused blocks and extents from the files and arranging all of the disk's free space as one large block.

Report Disk Free Space (FREES) scans the free space table on the hierarchical file system disks and reports on the total amount of free space and the largest amount of free space.

Report File Space by Owner (FOWN) scans files according to a user-specified mask and displays the total disk space used by each owner. This utility can be used to identify users with excess disk space or files using excess space.

File System Verification (FVERI) scans the directories and table structure of a hierarchical file system disk LU and reports inconsistent data. Extent pointers, data pointers, non-negative file types, and so on, are checked. This utility can be used after a system crash to verify the file system is still intact.

Transferring Data to and from Devices

You can send data to and from an I/O device instead of a file 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 disks, CTDs, and Distributed System (DS) network links.

The CI commands that transfer data to or from files are CO, CP, and LI. Other CI file-referencing commands that do not deal with the data in files (for example, RN and PU) cannot be used for this purpose.

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, data is terminated with a CNTL-D character (control key CNTL and D typed at the same time). For example:

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

The CO command puts everything you type into the file until a CNTL-D is entered.

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

```
CI> co 1 1
```

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, you cannot 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, the second program waits until the LU becomes available. Terminals are not locked so that messages can still get through to them.

Other special processing ensures that the data is transferred to or from the LU in the proper format. The system recognizes printers, magnetic tapes, and terminals, and it does special processing required for them. For most devices, data transfer is not a problem. If you have special devices, a special program must be written for computer-device interface.

In addition to the CI commands, most programs that use files 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 disk file and in this case, an LU number will not be accepted. This may occur because the program needs to read the data twice, or because it needs 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 files are handled by the FMGR program. FMGR should be used only for existing files created with FMGR. When using FMGR files in the RTE-A file system, note the differences between the two types of files.

The main characteristics of FMGR files and the difference between FMGR and normal files are:

- File names are limited to six characters.
- Directory names are limited to two characters, or can be a number.
- Subdirectories are not allowed.
- There can only be one directory per volume; the volume and directory are known collectively as a cartridge.
- There are no file type extensions, time stamps, or owners.
- Protection is included as part of the file descriptor in the form of a security code parameter.
- There is no facility available to unpurge a file.
- The following characters are not allowed in FMGR file names: `+ − : ,`

CI file-referencing 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 disk and/or CI directory, and CO will copy files to and from a FMGR disk 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 as a FMGR directory, but you can set it to zero:

```
CI> wd 0
```

This indicates that you have no working directory. When you have no working directory, the file system searches 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:

- Names with slashes cannot be used.
- Names with dots or ending with dots are not acceptable, except for a single dot in character position 2, 3, 4, or 5.
- The “at” sign (@) is interpreted as a wild card character in CI commands, although a FMGR file name containing @ will eventually be selected.

We recommend that you rename such files. 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.

Some CI commands do not work with FMGR files because of the differences between FMGR and CI directory information. 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.

DS File Access (DS Only)

RTE-A systems that use the DS/1000-IV Distributed System Network can access files located on other RTE-A systems within the DS network. This includes FMGR files located on other RTE-A 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. Local system (or local node) means your system, and 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.

In addition to the DS/1000-IV Distributed System Network, DS transparency monitors TRFAS and DSRTR must be linked and RP'd in your system.

Specifying Remote Files

RTE-A DS transparency software is used to access files at remote systems. For files in a VC+ environment, refer to the "Multiuser Remote File Access" section.

Files and directories in a remote system can be listed and copied to and from your system. Wildcard characters can be used in the file name 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; these are kept in a file called NODENAMES, which is described in the *RTE-A System Generation and Installation Manual*, part number 92077-90034. 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 [comment]
```

As an example:

```
* Test System1 (comment line)
1 SYS1
* Test System2
2 SYS2
* Central System
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>Nodename
```

or

```
File::Directory>Nodename
```

This specifies a file located at the node named NODENAME. The > sign must follow all other file information, including type, size, and record length. 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 the first character cannot be accessed. For example, the name >FILE can be used in a file specification, but A>FILE is interpreted as file A in the remote system named FILE.

Multiuser Remote File Access

If the remote system operates in the multiuser environment, provided by the optional HP 92078A VC+ Package, the appropriate account logon entry can be included in the remote file specification. The account name and password, if one is required, are specified within square brackets; for example, [USER] or [USER.GROUP]. 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 multiuser environment:

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

or

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

If the USER account has a password, you must enter it; use 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 logon, you can access all files available in that account, with the same restrictions applicable to that account. You'll remain logged on as long as the file is open and 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 VC+ system to another, two logon entries and two nodes are required for the source and destination system. The node specification for a local system can be omitted. File masks can be used.

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

or

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

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 that have special meaning, such as > or [. The DS transparency software operates from CI and other programs that use the RTE-A file system. If your system operates strictly with FMGR, 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> dl /dir/@>27
```

If a system failure such as power failure occurs while remote files are being accessed, note the following:

- If the remote system is down, requests to it will timeout, causing an error return from the FMP call making the request.
- 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 at the local end. Accesses to such files may get errors. Use the CLOSE utility described below to close these files.
- If the local system goes down, its files will be left open at the remote system. DS transparency software Revision 5000 and later will attempt to close them the first time the local system sends a request to the remote system after it is rebooted. Versions earlier than Revision 5000 will not close files that were left open. The recommended way to close these files is to use the CLOSE utility described below.

To close open files while accessing remote files:

```
CI> close /directory/file              (At local node)
```

or

```
CI> 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 logon name for the local file if one was supplied when it was opened, even if the file is in your local node. CLOSE must be given the full file descriptor of the file to be closed, including the path and DS information.

Remote File Access Limitations

In some cases, CI file manipulation commands cannot be used on a remote system. The most common cases are:

- The default working directory cannot be used at a remote system.
- A program contained in a remote file cannot be run. (However, you can copy the file to the local system and then run the program.)
- Volumes at remote systems cannot be mounted or dismounted.
- Ownership of directories cannot be examined at a remote system.
- I/O devices (such as terminals or printers) at a remote system cannot be accessed.

4

Controlling Programs

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. A brief summary of the program control commands is shown in NO TAG. Refer to Chapter 6 for a full explanation of these commands.

. Program Control Commands

Command	Task
AS *prog* <*part #*> [C|D]	Assign partition
AT *time* [*intv*l] *prog*|*file* [*pram*5*]	Set run time
BR [*prog*]	Break program execution
CZ *prog* [<*seg #*>|AL]	Display/modify CDS code space
DT *prog* [*size emaSize*] [*msegSize*]	Display/modify CDS data space
GO [*prog* [*pram*5*]]	Resume suspended program
OF [*prog* [ID]]	Remove program
PR *prog* [*priority*]	Display/modify program priority
PS [*prog*]	Display program status
RP *file* [*prog*]	Restore program
RS [*prog*[|*session*]] [*pram*5*]	Restart program
[**RU**] *prog*|*file* [*pram*5*]	Run program with wait
SS [*prog*]	Suspend program
SZ *prog* [*size* [*mseg size*]]	Display/specify program size
VS *prog* [*vsSize*]	Display/modify virtual EMA size
WS *prog* [*wsSize*]	Display/modify VMA working set size
XQ *prog*|*file* [*pram*5*]	Run program without wait
? [*prog*]	Online help

Program Identification

RTE-A provides many different programs 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 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.

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

Program Priorities

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

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), enter:

```
CI> ru edit
```

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

```
CI> edit
```

Any time a non-CI command is entered, CI first checks the $RU_FIRST predefined variable. If the variable is set to TRUE, CI assumes that the RU command was intended and attempts to execute the file. If you will be executing more programs than command files, you should set the $RU_FIRST variable to TRUE.

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

```
CI> edit prog.ftn
```

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

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

runs the editor, opens file PROG.FTN, and executes the editor S command (enter screen mode).

Parameters are accepted by other RTE-A programs such as LINK, FTN7X, and MACRO. These are described in their respective manuals. User programs may be written to accept up to five numeric parameters from the runstring and a character string. This facility is described in the *RTE-A Programmer's Reference Manual*, part number 92077-90007.

Program Execution

Upon receipt of a RU command, RTE-A searches for an existing ID segment for the program specified or creates 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 dispatches programs from this list in order of their priority.

Program CI is suspended to allow interaction between the program and your terminal. When the program terminates, CI again issues its prompt and accepts commands. This cycle is known as "run with wait."

Sometimes it is desirable to let a program run while continuing CI interaction. This may be so 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> 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 the CI run command, RU. For example, to run a program called OWNER, enter:

```
CI> ru owner
```

In this example, if the program was restored (that is, was assigned an ID segment), CI executes it. After the program terminates, it remains restored. If the program was not restored, CI restores and executes the program, and releases 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 and User-Definable Directory Search Path (UDSP) number one is not defined, CI first searches for an existing restored program, then for a file in the working directory, and finally for a file in a system global directory called PROGRAMS. If only the program name is specified and UDSP #1 is defined, CI first searches for an existing, restored program, then uses the search path defined by UDSP #1.

The following example illustrates how CI searches for programs, assuming UDSP #1 is not defined.

When the command EDIT is entered, it is examined by CI and interpreted as a program because it is not a CI command. CI searches for an ID segment restored for EDIT. If one is found, CI runs 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, AT, and RP commands as well as to scheduling operations done by other system programs such as EDIT and LOGON. 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 allows CI to find EDIT quickly in directory DIRNAME.

One way to make use of the default program search sequence is in program development. Because 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.

You can use UDSP #1 to change the default program search sequence. See the PATH command for information on defining UDSP #1.

Program CI can handle cases where there are two or more programs scheduled with the same name. This can happen in two situations: several copies of a program may be running at the same time (for example, EDIT may be run by several users); or shortening of two different file names may lead to the same 5-character program names (for example, 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, and so forth. 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 5.

Running Programs without Wait

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

```
CI> xq prog/file                (Scheduling PROG/file without wait)
CI>                             (PROG/file executing; CI back in interpretive
                                 mode)
```

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

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

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

Any errors reported during program execution are displayed at the terminal along with 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 and 6 for use of the WH command.

Time Scheduling Programs

To schedule a program to start at a later time (up to 24 hours), the AT (execute time) command is used. AT runs a program at a particular time based on the processor time-of-day clock. 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, enter:

```
CI> at 12:00:00 CLOCK a b c
CI>
```

CI returns control to you after this command. At 12:00, program CLOCK runs once. The AT command handles ID segments and program naming in the same way as the RU and XQ commands.

The time can be specified with the AM or PM parameter or in 24-hour format; 1:30 pm is 13:30. Minutes and seconds 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 runs at 4:00 pm tomorrow, rather than immediately.

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

```
CI> at 12:00:00 1 hour CLOCK a b c
```

Note that the a, b, and c parameters are passed to CLOCK on the first run only.

The time interval for repeated execution can be specified as hours, minutes or seconds. These can be abbreviated to the first character of the word (h, m, or s). Numbers such as 90 seconds are legal; the interval value can be up to 4095. The time scheduled program running at regular intervals must be stopped with the OF command. Refer to Chapters 2 and 6 for more information.

Systems having VC+ can also use the CRON utility and CRONTAB to schedule programs at specified dates and times. Refer to Chapter 6 for details.

Restoring Programs

Restoring a program is a process that establishes a program ID segment or a prototype ID segment for a program file.

Program ID Segments

Typically, program restoring 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 disk, 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 is restored.

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

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

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

The program can now be run using the RU, XQ, or AT command and the ID segment will remain allocated upon program termination. This is useful for shared CDS programs in the VC+ environment or programs that may be needed by only one user.

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

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

The second user can now use the RU, XQ, or AT 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 RTE-A will automatically create a new name in this case (described in the "Running Programs with Wait" section). In a multiuser environment, RP'd programs that are not running or time scheduled will be OF'd when the user logs off.

Prototype ID Segments

Using the "D" option with the RP command establishes a prototype ID segment (proto-ID) for a program file in a list in XSAM. Proto-IDs are selected words of program ID segments that are used to create program ID segments quickly (without going to the disk) when a program is scheduled without specifying a directory path. This process is called ID duplicating or "duping."

Program ID Duplicating

RTE-A provides a way to quickly create program ID segments by duplicating existing program ID segments or prototype ID segments. This process is performed if the program is scheduled without a directory path and one of the following conditions exists:

1. A program with the same name is permanently restored, busy, and clonable.

2. There is a proto-ID defined for the program in the system.

This saves considerable time when scheduling programs, because the disk does not need to be searched for the program file to create the program ID segment.

Removing Programs

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

```
CI> of <prog>
```

If the program was not restored with RP, its ID segment is released. If the program was restored with the RP command, only the execution is terminated, and the program ID segment remains intact. To remove the program's ID segment, the second parameter, ID, is needed. For example, to remove the ID segment of program TOWER that was restored with the RP command:

```
CI> 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, which may leave the file in an abnormal state.

The OF command with the D parameter removes currently defined proto-IDs from the system.

Breaking Program Execution

You can use the BR command to stop a program in an orderly manner, rather than abruptly as with the OF command. BR 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 CM> prompt. If the program was scheduled without wait (XQ or AT command), the BR command can be issued from CI. BR can be entered with or without a program name. For example:

```
CI> test2
   (press any key or, if FIFO buffering is enabled, press the break key)
CM> br test2
CI>
```

The program name must be the same name as the name reported by the WH command (including the session identification if in the VC+ environment), 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 6 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 *RTE-A 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 until the program finishes or use the OF command.

Suspending a Program

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

The SS command does not interrupt any I/O operation in progress. It waits until the I/O operation is finished. Note that this may take a long time, and there is no message while CI is waiting.

Resuming Program Execution

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

```
CI> xq test2
CI> ss test2
CI> go test2
```

Restarting a Program

The RS command restarts a program that is not executing properly; for example, a program that is hung on a downed device. The program is aborted and rescheduled for execution. The RS command can be used only from CM.

The following example restarts CI:

```
CM> rs,ci
```

Displaying Program Status

The PS command can be used to return the status of a specific program. PS returns some of the same information as the WH command, but in a shorter format. For example,

```
CI> ps prog1
PROG1 PR( 99) PC( 0) OF
```

Changing Program Priorities

All programs running under RTE-A have a priority number that 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 can also be changed dynamically with the PR command, as shown below.

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

A primary task of the RTE-A 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 a very long elapsed running time in a busy system, or if it does not run at all, its priority may be too low. On the other hand, if it runs to the exclusion of other user programs, then its priority might be too high.

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

```
CI> pr test2 50                    (Changes priority of program TEST 2 to 50)
```

If you give the PR command with a program name but no value, it displays the priority of the program.

```
CI> pr test2
Program priority = 50
CI>
```

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

Changing Memory Requirements

Some programs may require dynamic memory allocation; for example, reentrant routines or Pascal recursive procedures and dynamic data structures. Memory requirements may vary depending on input parameters or data given to the program. RTE-A will not be aware of these factors and might 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. You can use LINK to make sure the program will get extra memory every time it runs (this is described in the *RTE-A LINK User's Manual*, part number 92077-90035.) Or you can use the SZ command after the program has been restored but before running it. For example, to change the memory allocation of DATALOGGER to 20 pages:

```
CI> rp datalogger
RP'ed DATAL                        (Note the 5-character program name)
CI> sz datal 20
```

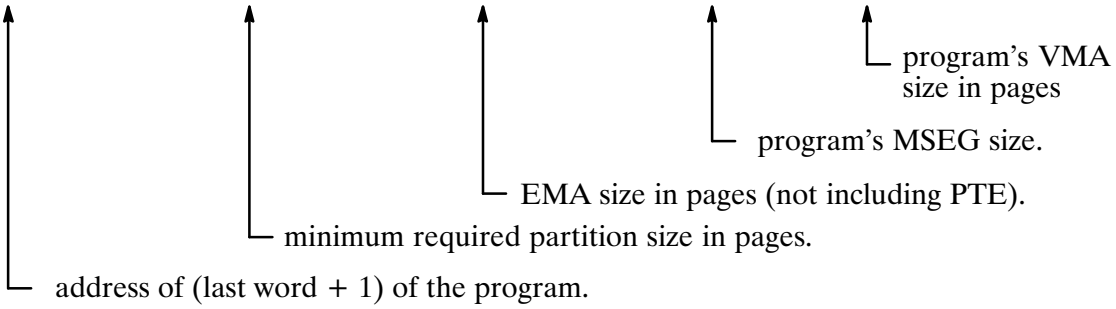
Now DATALOGGER will have 20 pages. The new memory partition allocation remains in effect as long as DATALOGGER is restored. If it is removed, it will revert to that defined by LINK. The SZ command cannot be used for a program that is executing. This command applies only to non-CDS programs. The CDS commands are described later in this chapter.

If a program uses EMA, the SZ command modifies the EMA data space only (in the range of 2 to 32,733 pages). Refer to "Models of EMA/VMA" in this chapter. For example:

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

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

```
CI> sz proge
Last=65211   Min Part=211   EMA/WS=180   Mseg=2   VMA=0
```



Assigning Partitions

The RTE-A system memory is divided at bootup 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> 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> as proga 0
```

If your system has the VC+ option and uses CDS programs, use the C or D parameter to assign the code or data partition, respectively:

```
CI> as proga 1 c

CI> as proga 8 d
```

Changing Virtual Memory Area

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

The WS size, from 2 to 32,733 pages, and the VMA space (VS), up to 65,536 pages, are defined using LINK. Refer to "Models of EMA/VMA" in this chapter. These can be changed with the WS or VS commands, respectively.

You may want to change the size of WS and VS with LINK, because this changes the size permanently. Alternatively, you can use CI commands after restoring the program. The WS or VS commands can also be used to find out the area defined. For example:

```
CI> rp datalogger

CI> ws datal                        (Display VMA information for DATA1)
2 Last Adr= 70221 Min. Part= 29 EMA/WS= 20 Mseg= 2 VMA= 200


CI> vs datal                        (Display VMA information for DATA1)
3 Last Adr= 70221 Min. Part= 29 EMA/WS= 20 Mseg= 2 VMA= 200
```

To change the WS and VS areas of a program:

```
CI> ws datal 45                     (Change working set size to 45 pages)

CI> 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 reverts to that defined at program link time. Refer to Chapter 6 for more information on the WS and VS commands.

Shared Programs (VC+ Only)

In systems with VC+, if program memory usage is a concern, shared programs let you save memory. Any CDS program can be specified as shareable at link time (details are given in the *LINK User's Manual*). Sharing a program means that each program invoked (RP'd) from that type 6 program file will share one code partition when dispatched. This saves memory and the disk overhead of loading a code partition for each program dispatched. Each shared program will own a separate data partition.

A typical shared program environment can be set up by entering the following:

```
CI> rp sh s2
CI> rp sh s3
CI> xq s2
CI> xq s3
CI> xq sh
```

In this example, only one code partition exists and it is the one shared by programs SH, S2, and S3 during their execution. There are separate data partitions for these programs.

Shared programs are transparent to the users. However, the code partition attributes cannot be altered without relinking. An error message is displayed when attempts are made to change the code partition attributes.

Changing CDS Program Memory Requirement (VC+ Only)

CDS programs have two areas of memory associated with them, one for code (the program itself) and one for data. To change the size allocation for these areas, use the CZ and DT commands.

The CZ command changes the size allocation of the code section. It changes the amount of memory the code uses by changing the number of code segments (pieces of the program) that are kept in memory at one time; the other pieces are kept on the disk. RTE-A keeps the most actively used pieces of the program in memory, leaving the others on the disk. To set up a CDS program called BIG with 5 code segments in memory at once, use the following command sequence:

```
CI> rp big
CI> cz big 5
```

If you use the keyword AL in place of a number, all code segments are kept in memory. This allocates more memory to program BIG, allowing less for other programs in the system. Note that all code segments must be in memory before a shared program can be executed.

The DT command changes the size allocation of the data section. To allow program BIG to use 30 pages of data, enter the following:

```
CI> rp big
CI> dt big 30
```

If BIG is an EMA program, the EMA space size is modified as in the SZ command.

The only method for a permanent change is to use LINK. The SZ, CZ, and DT commands are only effective until the program ID segment is released.

These commands can also be used to display the memory allocation for a particular CDS program. For example:

```
CI> cz big
Partition size=15  Program segments=10  Segment Blocks=3
```

This shows that the code partition will be 15 pages; there are 10 segments in BIG and three segments can be memory resident. Because all segments must be of equal size, it can be seen from the example that each segment will occupy five pages (partition size/segment blocks).

```
CI> dt big
Last Adr= 64000 Min.Part.= 112  EMA/WS= 85 Mseg= 1  VMA=2000
```

Models of EMA/VMA

There are three “models” of EMA/VMA for programs to use:

1. The Normal EMA/VMA model is used by programs that do not need to access more than one shareable EMA area, nor to access a shareable EMA in conjunction with a local EMA or VMA, nor to use an extended EMA or VMA working set size. This model is the most commonly used.

2. The Large EMA/VMA model is used by programs that need to access more than one shareable EMA area, or to access a shareable EMA and a local EMA or VMA.

3. The Extended EMA/VMA model is used by programs running on an A990 Computer that need to access an EMA or VMA working set that is larger than 1,022 pages. This model also offers all the features of the Large model: multiple shareable EMAs, and mixed shareable EMA and local EMA/VMA. Programs that use this model will execute correctly only on an A990 Computer.

The EMA/VMA model to be used by a program is specified at link time. Refer to the *RTE-A Programmer’s Reference Manual*, part number 92077-90007, for detailed information.

. Three Models of EMA/VMA

Feature	EMA/VMA Model		
	Normal	Large	Extended
Maximum size of any one EMA in pages	1022	1022	32733
Maximum size of a VMA working set in pages	1022	1022	32733
Maximum number of SHEMAs a program may access	1	64	64
Number of SHEMAs accessible when local EMA/VMA used	0	63	63

5

Multiuser Session Operation (VC+ Only)

This chapter describes the multiuser session capability available on systems with the HP 92078A VC+ Package. Multiuser session handling is recommended for program development systems as well as other systems where several users are using the RTE-A Operating System at once. Topics in this chapter include logon and logoff, group and user accounts, creating and modifying user accounts, capability levels, superuser capabilities, session handling, identifying programs, and running out of SAM.

Note The concepts of capability levels and of sufficient user capability to perform a function are discussed throughout this chapter but apply only if Security/1000 is installed and turned on in your system. The concept of superusers applies only if Security/1000 is not installed in your system or Security/1000 is installed but turned off.

Logon and Logoff

Logging on is the process of entering a user.group identification and a password, if one is defined. The user.group identification consists of a user name and group account name separated by a period to identify who is going to use the system. You must log on before you can perform any operations on the system.

RTE-A displays a prompt on a currently enabled (but not currently logged on) session terminal when it is ready to accept a logon request. If the logon prompt was not displayed, hit any key to get the attention of the system. A sample prompt is as follows:

```
Please log on:
```

You can log on by entering a user.group identification (established for you by the System Manager or installer) at the logon prompt. Every user in the multiuser system has a unique user name and is a member of one or more groups. An example of a valid user.group identification is:

```
Please log on:Sandi.Lab          (User “Sandi” in group “Lab”)
```

The “.group” portion in the user.group identification above is optional, because every user has a default logon group as part of his/her user definition. Thus, if the default logon group for user Sandi is group QA and just the user name Sandi is entered at the logon prompt, the user is logged

on as user Sandi in group QA (Sandi.QA). If user Sandi wants to be logged on in group Lab, the ".Lab" portion of the user.group identification must be specified. To determine your default logon group you can list your user account definition with the LIST command in GRUMP, the Group and User Management Program. You can also log on without specifying a group name and do a set command in CI; your user.group identification will be in the CI set variable $LOGON. A user's default logon group can be changed with the ALTER USER command in GRUMP by the System Manager, a superuser, or a user with a high enough capability.

After a user.group identification is received, the system prompts for a password.

```
Password?
```

Enter your password. Your entry will not be displayed on the terminal screen to prevent others from seeing your password. If you do not have a password, just enter a carriage return at the 'Password? ' prompt.

Alternatively, you may enter your user name or user.group identification followed by a slash (/) if you have no password; this avoids being prompted for a password. For example,

```
Please log on: Sandi.Lab/          (User "Sandi" in group "Lab" has no password)
```

If you do have a password, you may enter your user name or user.group, followed by a slash and your password (do not use commas or blanks as separators), all on one line, thus avoiding being prompted for your password. Because your password is displayed using this method, it should only be used if you are not concerned with account security. For example,

```
Please log on: Sandi.Lab/aslan   (User "Sandi" in group "Lab" with
                                   password "aslan")
```

If the user name in the user.group identification provided is not recognized as valid by the system program LOGON, one of the following error messages is displayed on your terminal (assume FRED is the user name supplied):

```
LOGOX: No such user FRED
```

or

```
LOGOX: Logon incorrect
```

If the group name in the user.group identification provided is not recognized as valid by LOGON, the following error message is displayed (assume BANK is the group name supplied):

```
LOGOX: Unable to access group BANK
```

If the password provided is incorrect, one of the following error messages is displayed:

```
LOGOX: Incorrect password
```

or

```
LOGOX: Logon incorrect
```

The messages are preceded by "LOGOX:", as opposed to "LOGON:", because the system program LOGON handles both the logon and logoff process and some of the messages are used for both processes. You can try to log on again by repeating the process. After two or three unsuccessful attempts you should contact your System Manager for assistance.

There are also checks for CPU usage and connect time limits for users within groups and for groups as a whole. Thus, you can enter a valid user.group identification and password but if you

have exceeded any one of the four limits, you will be denied access to the system. If this happens, one of the following messages will be displayed on your terminal (assume SANDI.LAB is the user.group identification supplied):

```
LOGOX: Your CPU time within LAB has been exceeded
LOGOX: Your Connect time within LAB has been exceeded
LOGOX: LAB CPU time limit has been exceeded
LOGOX: LAB Connect time limit has been exceeded
```

The first two messages above refer to the limits of the user within the group (SANDI in group LAB for the given user.group identification). The second two messages refer to the limits of the group as a whole (group LAB for the given user.group identification). If you receive any of these messages, you should contact your System Manager for assistance.

If there are no error messages displayed on your terminal, you are now successfully logged on. The system then runs a predefined startup program as part of the logon process to get you started. Typically, this program is the standard Command Interpreter, CI. The program to be executed at logon can be changed to a program of your choice with the ALTER USER command in GRUMP by the System Manager, a superuser, or a user with high enough capability.

Only users with accounts can log on. User accounts are explained in the "User Accounts" section. If you forget your password, there is no way to reconstruct it. The System Manager, a superuser, or a user with high enough capability can assign you a new password without supplying the old password using the ALTER USER command in GRUMP. Any user can change a password with the PASSWORD command in GRUMP, providing he/she knows the existing password.

Logging off is straightforward. The operating system automatically logs you off when your last active or scheduled program terminates. If this program is CI, an EX command after the CI> prompt will log you off. When you exit from CI and have a logoff program/command file, it is executed or transferred to before your CI terminates and the OS logs you off. Note that if CI is your primary program and aborts for any reason, such as when an OF command is issued from CM when your CI is waiting for a locked LU, you are automatically logged off and the logoff program/command file is not executed. The logoff program/command file can be defined and modified with the ALTER USER command in GRUMP by the System Manager, a superuser, or a user with high enough capability.

If you cannot exit from your primary program and have programs running, you must give directives for the disposal of these programs. This situation is discussed further in the session handling section of this chapter.

If multiuser accounting is enabled, the system program LOGON, which handles both the log on and log off process, updates the accounting information in the user and group configuration files that correspond to the user.group you are logged on with. Then your CPU usage and system connect time for this session are displayed on the terminal. For example:

```
Session 110 finished.  Tue Nov 24, 1988 9:36:45 am

                          Connect time                    CPU usage
Session:            5 hr  5 min 22 sec       0 hr 0 min  4 sec 660 msec
Cumulative:      684 hr 51 min 34 sec       8 hr 8 min 42 sec 140 msec
```

When logging off, you should allow all of the accounting information to be displayed. If you interrupt the system program LOGON in any way, for example, by turning off your terminal, switching your terminal connection, or hitting a key, LOGON can be suspended. If LOGON is suspended on your terminal, it can cause problems for other people trying to log on and log off.

Group Accounts

The operating system maintains information on all groups in group configuration files in the /USERS directory. Each group configuration file contains the group ID, CPU and connect time totals, CPU and connect time limits, an LU access table, a list of its members, and other pertinent information.

The system uses this file when it creates a session for a user trying to log on associated with the group it defines. The file is used to verify that the user is a member of the group, to check that the accounting limits of the group were not exceeded, and to include the group's resources in the operating environment of the user session. When a user associated with a group logs off, the CPU usage and connect time totals are updated in the group configuration file if multiuser accounting is turned on.

User Accounts

The operating system maintains information on all users in user configuration files in the /USERS directory. Each user configuration file contains the user's real name, the encoded password, other pertinent information about the user, and information unique to the user for each group to which the user belongs.

When you log on, the system uses this configuration file to:

- verify the user logon name,
- check the user CPU usage and connect time limits,
- run the startup program,
- designate a working directory,
- initialize the UDSP tables,
- create the session LU access table,
- determine the size of the Environment Variable Block, and
- initially determine the capability of the user (in the form of a capability level and/or the superuser/non-superuser status).

When you log off, CI retrieves the logoff program/command file from the user configuration file, if one is defined, and executes it. The system uses the user configuration file to update CPU usage and connect time totals if multiuser accounting is turned on.

The user logon name is used by programs that provide system status. The WH US command can be used to display all the active users and their associated sessions. The spooling system can be directed to maintain a record of users logging on and off with the error logging feature.

Your working directory is established automatically when you log on. This eliminates the step of establishing your working directory with a WD command. The name of the working directory can be any existing directory available to you. If this directory does not exist, a message is displayed on your terminal screen. A typical practice is to use the same name for the working directory as the logon name. For example, user SMITH would be assigned directory /SMITH.

Creating and Modifying Accounts

In order to log on, a user must have an established user account. A user account is defined by a user configuration file, located in the /USERS directory, which contains information about the user. Every user must belong to one or more groups. A group account is defined by a group configuration file, also located in the /USERS directory, which contains information about the group.

The GRoup and User Management Program (GRUMP) is used to create and modify group and user accounts with the NEW and ALTER commands, respectively. Group and user accounts can be removed from the system with the PURGE command in GRUMP. Typically, these commands can only be used by the System Manager, a superuser, or a user with a very high capability.

Listings of group and user accounts can be obtained with the LIST command in GRUMP. As stated earlier, the PASSWORD command in GRUMP can be used to change a user's password if the existing password is known. See the *RTE-A System Manager's Manual*, part number 92077-90056, for a description of GRUMP commands.

Superusers

The concept of superusers and non-superusers applies only in environments where Security/1000 is not installed or is installed but not turned on.

Generally, only the System Manager is given superuser capabilities. The System Manager has a thorough knowledge of RTE-A and the current system configuration and will use the superuser account only for system maintenance duties.

Superusers are not subject to file protection; they can read or write any file or directory. Superusers are also given owner privileges to all directories, so they can change protection or ownership of any file or directory.

The commands IN, TM, and TO used with the parameters to modify information are reserved for superusers, because they can adversely affect system performance if used improperly. When normal users try to use them, they get a message indicating that these commands are for superusers only.

Superusers are allowed to control the protected system programs. For example, they can remove or change the priority of system programs by using the OF and PR commands. These programs are protected because they affect system operations.

Capability Levels

A capability level is an integer in the range of 0 to 31, where 0 is the lowest and 31 is the highest. A capability of 31 is equivalent to a superuser in an environment without Security/1000. Capability levels are used to control access to the various functions and resources of the system. The meaning of a capability level is dependent on its context. Specifying what can be done with a given capability is a function of the Security/1000 configuration as defined by the System Manager.

Note The concepts of capability levels apply only if Security/1000 is installed and turned on in your system.

Command Capability Levels

Command capability levels define which set of commands can be executed by the user. The sets of commands are SYSTEM commands, CI commands, SECTL commands, GRUMP commands, LINK commands, and the commands of any program using Security/1000 to control command use. A user has access to all commands at or below his/her capability level. For example, a user with a capability level of 30 has access to all commands available at level 30 and below. Users can see what their capability level is with the WH command in CI or by listing their user account definition in GRUMP.

A user's command capability level is defined and altered with the NEW USER and ALTER USER commands, in GRUMP by the System Manager or a user with high enough capability, and stored in the user configuration file. A user has the same capability level in each group in which he/she is a member. As a user's needs change, command capability levels can be altered with GRUMP by the System Manager or a user with high enough capability.

Program Protection through Capability Levels

Programs can have three capability levels. The program capability level (PROGCPLV) is the capability at which the program runs. It determines which functions a program can perform. A program cannot perform any function where the PROGCPLV is less than the capability required to perform the function. The required user capability level (RQUSCPLV) is the level that a user must equal or exceed in order to run the program. The third capability level is a copy of the original program capability level (ORGCPLV) given to the program at link time or set with the SECTL utility. This is used for internal management and is not directly accessible to the user.

When a program is run with Security/1000 turned on, the higher of the ORGCPLV and USERCPLV is assigned to the PROGCPLV. This allows programs assigned a low capability level to perform tasks requiring a high capability level if requested.

There are two levels of checking performed before you can run a program. The first level check is the file system security. If you cannot access the program file, it cannot be RP'd. If it cannot be RP'd, it cannot be run. The second level of security checking is that the RQUSCPLV of the program, once its ID segment has been constructed, must be less than or equal to the capability level of the user making the schedule request. Thus, if a program does not have an ID segment, both levels of security checking must be passed before it can run. If the program already has an ID segment, that is, it is already RP'd, only the second level must be passed as the file system is not involved in the schedule request.

The double layer of checking can lead to a situation where you may pass the file system check and thus RP the program, but cannot schedule it because the second layer failed. If you fail at either level of the security checking, you will not be able to run the program and the program's ID segment will be released.

Session Handling

When any user logs on, the RTE-A system handles all operations connected with that user. The system management of that user's operations, including all programs initiated by that user, is called a session. A logon creates a session, and a logoff terminates the session. Session handling is a more advanced topic. If you are a first time user of the multiuser environment, you may skip to the paragraph under Identifying Programs.

When a system is generated, the maximum number of concurrent sessions allowed is defined. At the beginning of each session, upon successful logon, the startup program defined in the account file (usually CI) is run. From this point, all programs run from CI belong to the session created. These programs are attached to the same session as CI. This means these programs inherit the capabilities of the session for the duration they are attached to it, including the user name, working directory and user flag attributes. It is important to note that these properties really belong to the session, and not to the individual program. If, for example, the working directory is changed for a session, it is changed for all programs attached to the session.

Most sessions are associated with the terminal where the logon request was made. This terminal is assigned a logical unit number LU 1 for I/O purposes. Sessions that have a terminal are known as interactive sessions. Interactive sessions receive services such as automatic scheduling of CM when a user types a character when CI is unavailable.

Sessions can also be background (or non-interactive) sessions. Background sessions are not associated with a terminal in the same way that normal (interactive) sessions are. Background sessions are initiated either through programmatic logon requests, by programs such as DS monitors, or through an EX command to CI that turns the session into a background session. Background sessions are appropriate for programs that can run without a terminal. They are not appropriate if the programs attached to the session require interactive inputs, because they could conflict with an interactive session using that terminal.

There is always one session created without an owner; this is known as the system session, and it is used to hold programs not attached to any other session. Protected system programs typically are included in this session. The system session has the following attributes: name SYSTEM, a superuser (capability level 31), and a working directory of that set in the Welcome file (unless a program in the system session changes it).

Each session is identified by a session number. For interactive programs, this is the LU number of the terminal associated with the session. For background sessions, the system selects a session number that does not correspond with any LUs in the system. The session number is useful when examining the system status. The session LU number is important in identifying programs attached to other sessions in the system.

Identifying Programs

There are two kinds of programs: normal programs such as EDIT, which are used by all users, and system programs such as D.RTR, which are not cloned. This attribute is selected when the program is linked.

There can be one or more copies of each normal program per session. The second through *n*th copy will be clones of the first. For example, each user has a copy of the normal program CI, named CI in his/her session. If a user runs a second copy of CI, it will be a clone of the first copy, called CI..A.

Also note that the system session is considered an extension of each user session. This means if there is a normal program running in the system session and you, as a session user, try to run the program in your session, it will be a clone of the one in the system session. Thus, if CI is running in the system session (for example, CI is the startup program and the welcome file is executing) and CI is run in a user session, the user session gets a clone of CI (for example, CI..A). Similarly, if you try to RP CI, the system reports "Program name exists CI".

To identify a program associated with a different session, qualify the name with a session number. For example, to check the priority of EDIT associated with LU 14, enter:

```
CI> pr edit/14
```

Without the LU number, you will receive a message such as "No such program" because there is no EDIT attached to your session, nor is there any EDIT that is a system program.

System programs can always be identified by name, as there is only one. They are usually protected to prevent modification of their attributes and they are never cloned with different names.

Running Out of SAM

If your RTE-A/VC+ system runs out of System Available Memory (SAM), the PROMT program takes action to allow the system to be recovered. If you get error messages indicating this problem exists, inform your System Manager.

6

Command Descriptions

This chapter contains descriptions of all CI commands. The commands are described in alphabetical order. A tutorial of most of these commands and a command summary were provided in previous chapters of this manual. Commands or capabilities that are specific to the HP 92078A VC+ option are indicated as "VC+ Only". Base set commands, which are available from the system prompt, are indicated by an asterisk (*).

Capability Levels for CI Commands

If Security/1000 is installed, capability levels can be defined for all CI commands. You must have a capability level that is equal to or greater than the level of a command in order to use it.

A number of CI commands map directly onto FMP routines or the OS kernel. For example, PU (purge file) maps onto FmpPurge, and BR maps directly onto the OS kernel command BR. It is therefore possible to pass the CI command security check but fail at a deeper level, such as FMP or the kernel. More details are provided in the *RTE-A System Manager's Manual*.

CI commands fall into the following categories:

- The command is really a program. Three levels of security are involved:
  - CI security check. Can the command be issued from CI?
  - Program security. Do you have enough capability to run the program that implements the command?
  - FMP or operating system kernel checking. Do you have enough capability to call the FMP routine or issue the kernel command that the program uses?

  CI commands in this category:

  ASK, DL, IO, IS, PATH, RS, SP, WH, WHOSD.

- The CI command maps onto an FMP routine. Two levels of security are involved.
  - CI security check. Can the command be issued from CI?
  - FMP routines. Do you have the capability to call the FMP routine that CI is about to use?

  CI commands in this category:

  CD, CR, CRDIR, DC, IN, LI, MC, MO, OWNER, PROT, PU, PWD, RN, RP, UR, UNPU, WD, XQ.

- The CI command maps onto an operating system kernel command. Two levels of security are involved.
  - CI security check. Can the command be issued from CI?
  - Operating system kernel command. Do you have the capability to use the operating system kernel command that CI is about to use?

  CI commands in this category:

  AS, BR, CZ, DT, GO, OF, PR, PS, SS, SZ, UL, UP, VS, WS.

- The CI command is handled internally. One level of security is involved.
  - CI security check. Can the command be issued from CI?

  CI commands in this category:

  ALIAS, AT, CL, CN, ECHO, EX, FUNCTION, FUNCTIONS, IF-THEN-ELSE-FI, RETURN, SET, TM, TO, TR, UNSET, WHILE, ?.

Some documented utilities that are run implicitly by CI, such as LINK, SECTL, and TF, are subject to the following:

- Program security checking; a check to determine whether there is enough capability to run the program.
- FMP or operating system kernel checking; a check to determine whether there is enough capability to call the FMP routine or issue the kernel command that the program uses.
- Any capability level checking that the program may do internally on its functionality or commands.

Precedence Within CI

The precedence within CI is: aliases, internal CI commands, functions, or implied TR or RU (that is, transfer or run files). In order to bypass this, for example, to have an alias use a CI command, the command must be preceded by a backslash (\) and be in uppercase.

? (Help)

Purpose: Displays a summary of CI commands or a brief description of a command or item on the summary display.

Syntax: `? [`*command*`]`

Description:

This command provides a quick reference of CI commands and utility programs. The form "?" without any parameters lists the HELP directory, showing a summary of available files. Entering "`?` *command*" lists a file called /HELP/*command*; for example, "`? 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.

* (Comment)

Purpose: Allows entry of comments in transfer files.

Syntax: `* [`*comment*`]`

Description:

When an asterisk (*) is the first character in a line, CI ignores the entire line. The asterisk can be used to add comments to a CI command (transfer) file.

AB2MI (Absolute Binary to Memory Image)

Purpose: Converts an arbitrary type 7 file to a type 1 file in memory image format.

Syntax: `AB2MI [`*input_file output_file*`]`

input_file is the file name of a type 7 file or device.

output_file is the file name of a type 1 file.

If no parameters are specified, AB2MI prompts you interactively, as follows:

```
This program copies an absolute binary (type 7)
file to a memory image (type 1) file.

Please enter the input and output filenames.
Format:  input::directory,  output::directory::size
```

Description:

AB2MI allows any arbitrary code, whether or not it was produced by the system generator, to be loaded into processor memory. This is useful because memory image format is required by the disk bootstrap program.

AB2MI also patches an existing type 1 file. In this case, the input file overlays selected portions of the output file.

If the output file does not exist, it is created by AB2MI (default size, 256 blocks); otherwise, it is overlaid.

At the conclusion of the conversion, the program displays the message:

```
Highest block written: nnn
```

Here *nnn* is the number of blocks required to contain the input file data.

Break Detection

To stop execution of the program, you may enter the BReak command as follows:

```
CI> br,ab2mi
```

When AB2MI detects a break, it issues the following message:

```
Break Flag Set
```

The program then closes the input and output files and terminates. Note that the output file is incomplete if the break flag is set during a conversion.

AB2MI Error Messages

Break Flag Set

The break flag is tested in each major loop within the program. When the break flag is detected, the program closes the input and output files and terminates.

Checksum Error

The checksum is computed on the input record and must match the checksum word stored in the record itself. This error can result from specifying an input file of the wrong type (that is, ASCII).

Input File Error *xxxxxx*
Output File Error *xxxxxx*

Any error arising from calling a file management subroutine results in this message. The file error code is reported and the files are closed.

ALIAS (Define/Display Aliases; VC+ Only)

Purpose: Defines a CI alias or displays all previously defined aliases.

Syntax:
```
ALIAS [-x|+x] [alias_name [[=] string_value]]
ALIAS alias_name
```

`-x` if the user session has an Environment Variable Block, this option "exports" a defined alias or defines a new one as exported (also known as being in the environment). Exported aliases are available to all copies of CI in the session, including CM.

`+x` imports an exported alias or defines a new alias as imported (also known as being a local alias). An imported alias is defined only for the current CI.

If neither the −x or +x option is given, the alias is defined as a local alias.

alias_name is a string of up to 32 letters, digits, and underscores, not starting with a digit.

string_value is a command string terminated by the end of the command line or a semicolon (;), whichever comes first. Command lines can be 255 characters.

Description:

The command "`ALIAS` *alias_name*" displays the value of *alias_name*.

The command "`ALIAS`" (with no parameters) displays all aliases. The local aliases are displayed first, followed by a blank line, then the exported aliases are displayed. The blank line occurs whether or not any aliases of either type are defined.

When an alias is entered as the first word of a command, CI replaces the alias with the defined value before processing the command. The remainder of the command line is concatenated onto the alias value.

Quoting preserves lowercase letters and spaces.

The precedence within CI is: aliases, internal CI commands, functions, or implied TR or RU (that is, transfer or run files). In order to by-pass this, for example, to have an alias use a CI command, the command must be preceded by a backslash (\) and be in uppercase.

The UNALIAS command, described in this chapter, can be used to delete an alias.

Examples:

```
CI> alias func = function        (Define a local alias called FUNC, shorthand for
                                  the FUNCTION command)

CI> alias funcs=functions        (Define a local alias FUNCS, shorthand for the
                                  FUNCTIONS command)

CI> alias smith jones            (Define a local alias SMITH to be JONES)

CI> alias howdy = 'echo ''How are you today?''
                                 (Define a alias HOWDY that has lowercase
                                  letters and blanks)

CI> alias unfunc unset -f        (Define an alias UNFUNC that will unset
                                  functions)

CI> alias -x whu wh,us           (Define and export the alias WHU; does not
                                  affect local copy, if any)

CI> alias -x func                (Export the previously defined alias FUNC;
                                  deletes the local copy)

CI> alias +x lin=li,-n           (Define a local alias LIN that is LI with the line
                                  number option)

CI> alias li '\LI -n'            (Defines LI to be LI with the line number
                                  option)

CI> alias +x mine                (Take the alias MINE from the EVB and make it
                                  a local alias, assuming MINE was previously
                                  exported)

CI> alias                        (Show defined aliases and their values)

CI> alias +x                     (Show local aliases and their values)

CI> alias -x                     (Show exported aliases and their values)

CI> alias func                   (Show the value of the alias FUNC)
```

AS (Assign Partition)*

Purpose: Assigns a program to a reserved partition.

Syntax: `AS` *prog* `<`*part #*`> [C|D]`

prog is the program name, up to five characters, with an optional session identifier.

part # is a number that identifies the partition to which the named program will be assigned. If the partition number equals 0, the AS command removes the current assignment.

`C|D` optional parameter (VC+ only).

C (or code) = assign the code (executable) section of the program to the reserved partition.

D (or data) = assign the data section of the program to the reserved partition.

The default is the data section.

Description:

The program being assigned must have been restored previously with the RP command and must be dormant. Whenever the program runs, its code and/or data are placed in the designated partition(s). If assigned to partition zero, they are placed in dynamic memory. Refer to the *RTE-A System Design Manual* for a description of reserved and dynamic memory. If more than one program is assigned the same partition, the programs contend for the space in a normal priority swapping scheme.

Examples:

```
CI> as test2 2          (Assigns program TEST2 to reserved partition 2)

CI> as test 0 d         (Program test to run in dynamic memory)

CI> as prog 1 c         (Assigns the code section of CDS program PROG to
                         reserved partition 1)
```

ASK (Display a Prompt and Read a Response)

Purpose: Displays a question or prompt, reads the response from the terminal and passes it back to the scheduling program in $RETURN_S.

Syntax: ASK *'character string'*

character string is any question or prompt you want to use. The string must be enclosed in backquotes (``).

Description:

The ASK command displays a question or prompt, reads a response from the terminal, and passes back information about the response to the scheduling program in return variables. It passes the following information back: an indication if the command was successful (Return1), the character length of the response string ($RETURN2), the index of the first response character in the option string ($RETURN3), and the response string ($RETURN_S).

If the character string parameter of ASK contains a '?', the question string stops at the first '?' found and the rest of the string is taken as an option string. The zero relative index of the first response character in the option string is returned to the scheduling program. If you want it to be 1 relative, put a space between the '?' and the first character in the option string. If the first character in the response is not in the option string, a −1 is returned. If there are no non-blank characters in the response, a −2 is returned.

ASK returns the following in $RETURN1 through $RETURN3 and RETURN_S:

$RETURN1 :	0	command executed successfully
	−1	error in executing the command
$RETURN2 :		contains the character length of the response string
$RETURN3 :	≥0	zero relative index of the first response character in the option string
	−1	first response character is not in the option string
	−2	no non-blank characters were entered
	−3	timeout detected
$RETURN_S :		contains the response string

ASK

Examples:

`CI> ASK 'How are you?'` (Displays the question "How are you?" on the next line, reads the answer, and passes it to the scheduling program. If no option string is specified, the index $RETURN3 is −1.)

`CI> ASK 'Command>'` (Displays the prompt "Command>" on the next line, reads the response, and passes it to the scheduling program.)

`CI> ASK 'Purge this (Yes, No, Abort, Stop asking)?YNAS'`

(Displays the question:

"`Purge this (Yes, No, Abort, Stop asking)?`",

reads the response, determines the zero relative index of the first response character in the option string, and passes the information back to the scheduling program.

This will return the following in $RETURN3:

0	if Y is the first character in the response.
1	if N is the first character in the response.
2	if A is the first character in the response.
3	if S is the first character in the response.
−1	if the first character in the response is some other character.
−2	if there are no non-blank characters in the response.
−3	if the request timed out.

AT (Set Program Run Time)

Purpose: Sets the execution time of a program. The program also can be set to run at regular intervals.

Syntax: `AT` *time* [*intvl*] *prog*|*file* [*pram*5*]

time is the program run time, specified in 24-hour or 12-hour format:

hr:min:sec	(For example, "`13:30`" for one-thirty pm)
or	
hr:min:sec `AM/PM`	(For example, "`1:30 pm`")

If time is zero, the program is scheduled immediately.

intvl is the optional rescheduling interval with a value between 0 and 4095 specified in the following unit:

mil	for milliseconds
s or sec	for seconds
m or min	for minutes
h or hour	for hours

Intervals greater than 24 hours are reduced to modulo of 24 hours (for example, 27 hours to 3 hours). The start time must be specified even if only interval scheduling is needed.

prog|*file* is a 5-character program name or a file descriptor that identifies a type 6 file.

*pram*5* are the parameters passed to the program. Up to five are allowed, with up to 256 characters for the whole command runstring.

Description:

A program scheduled with AT executes in the same way as with the RU command except that it waits until the designated time to begin executing. The runstring and parameters are passed only once if an execution interval is specified. You can use the RU command to run programs scheduled with AT before the scheduled time without affecting the schedule. However, any parameters passed to the scheduled program will be lost.

Refer to the discussion of Parameter Passing and Parsing in the *RTE-A Programmer's Reference Manual* to learn more about retrieving a runstring.

AT

Examples:

Set PROGB execution time to 4 am:

```
CI> at 4:00:00 progb
```

or

```
CI> at 4 progb
```

Note that if the program name is AM or PM in the above example, CI will be confused. In this case, the runstring must be specified as:

```
CI> at 4 am pm
```

Therefore, to avoid confusion, do not use program names such as AM or PM.

To execute program TIMER every minute:

```
CI> at 1 1 min timer
```

or

```
CI> at 1 60 sec timer
```

Program TIMER will start at 1 am and execute every minute thereafter.

To execute program TIMER now and every 10 seconds after:

```
CI> at 0 10 sec timer
```

BR (Break Program Execution)*

Purpose: Sets a flag to allow limited communication with a program.

Syntax: `BR [`*prog*`]`

prog is the program name, up to five characters, with an optional session identifier. The default is the last scheduled program.

Description:

This command is used to stop programs in an orderly manner. BR sets a break flag in the program's ID segment, providing a way to signal a running program that it is to be stopped. It is up to the program to check the break flag; otherwise, the command has no effect. The break flag can be checked with system call IFBRK, described in the *RTE-A Programmer's Reference Manual.*

If no program is specified and the startup program (usually CI) has scheduled another program, the BR command executes on that program unless it, in turn, has scheduled a program. The search continues down the program scheduling chain and the BR command is executed on the last program. However, if the last program is a protected system program, the BR command executes on the program that scheduled the protected system program.

If the BR command is issued while CI is executing nested command files, you will receive the following prompt:

```
Multiple levels of command files.  Do you wish to
(R)eturn to prior command file or (A)bort all command files [A]?
```

Returning to the prior level means that CI breaks out of the currently executing command file and goes to the next one up. Aborting means that CI breaks out of all the command files, no matter how deeply they are nested, to the interactive level.

CALLM (Merge Text Files for CALLS Utility)

Purpose: Merges together a number of text files that contain input to the CALLS program and creates a single compressed file suitable for reading by the CALLS facility.

Syntax: `CALLM [-`*options*`]` *cmd_file dest_file*

-options is a string of one or more of the following characters preceded by a dash:

- `l` suppress listing the names of files read
- `o` overlay an existing *dest_file*
- `v` verify that an existing *dest_file* should be overlaid
- `c` inhibit text compression of *dest_file*. The default is to compress the text in *dest_file*.

cmd_file is the name of a file containing a list of text files to be read, one per line. The CALLS input is extracted from each of these files and merged into *dest_file*.

dest_file is the name of the destination file, to be used as an input file for CALLS. If compression is performed then this file will be of file type 6004.

Description:

Compression is performed via the CompressAsciiRLE routine. As explained in the *RTE-A/RTE-6/VM Relocatable Libraries Reference Manual*, part number 92077-90037, this compression cannot be performed on characters that use the eighth bit of the ASCII code, such as binary data or extended ASCII character sets (Kanji, for example).

The command file is a file in format similar to MERGE command files. Each line contains either the name of an input text file to be merged into *dest_file*, or the line contains a comment prefixed with "* ". Each input text file may be the compressed output of a previous CALLM execution if it is of file type 6004. The suggested file type extension for CALLM command files is ".CMRG".

The text files may be program source code that contain CALLS input in comments, where the comment must start with character "*" or "{" in column 1, and be followed immediately by the CALLS directive or explanatory text. CALLM will include in the *dest_file* only lines between and including ".topic" and ".end" CALLS directives, leaving out the intervening source code. If a plain text line not within a source code comment begins with either "*" or "{" in column 1, then that character must be doubled, such that the first one is discarded as a comment character.

.Include Directive

The CALLM utility processes an additional directive that includes another text file into the output file at the position where the directive is given. The syntax is:

```
.include <file>
```

These directives cannot be nested.

The example below shows the source code input format in FORTRAN or Macro:

```
*.topic mysubroutine
*.group subroutines
*one-line description of mysubroutine
*
*Calling sequence:
*
*      call mysubroutine(parm1,parm2)
*
*         :   etc.
*
*.see otherroutine
*.end
```

A Pascal example:

```
{.topic myprocedure
{one-line description of myprocedure
{
{Calling sequence:

          :   etc.

{.end
{}
```

Note the closing brace on the last line that terminates the Pascal comment.

See the section earlier in this chapter on the CALLS utility for more information about that program, and about the text input format expected.

CALLS (Online Help Facility)

Purpose: Looks up keywords entered by the user in a catalog containing definitions of keywords and associated text, and displays that text.

Syntax: `CALLS [-`*flags*`] [`*keyword*`]`

-flags is a string of characters preceded by a dash (−). Where an argument is required, the next word in the runstring is consumed, delimited by blanks or a comma. The flags are:

`C` *catalog* The name of the CALLS catalog to use. By default, directory /CATALOGS and type extension .CALL are added to the given name. The default catalog is /CATALOGS/CALLS.CALL.

`L` *listfile* Divert the text listing to the named file. By default the text is listed to the terminal.

`P` *pgsize* Set the number of lines per page for "More..." prompting on the terminal. The default size is 22 lines.

`B` Build the index file and terminate. See the discussion later in this chapter on index files.

For example, "`calls -c utils -p 5`" and "`calls -cp utils 5`" both use catalog /CATALOGS/UTILS.CALL and five lines per page.

keyword is the keyword for which the associated text is to be listed. If not given, the default keyword ("[default]") for the selected catalog is listed.

Description:

The catalog used by CALLS can specify hierarchical groupings of keywords and can suggest related keywords that may be of further interest after the text for a certain keyword is viewed.

At certain times, CALLS may prompt you to select another topic to display as indicated below:

```
Put cursor on desired name or type new name, press return.
```

This occurs when no topic keyword is given in the runstring, or when a mask is given. This also occurs when the topic selected has other topics associated with it, which you may also want to read.

When you press carriage return, CALLS reads the line under the cursor from the screen, isolates the word under or to the left of the cursor, and uses that word as the new topic name. If there is no word to the left or under the cursor, CALLS looks to the right of the cursor. If there is no word on the line at all, CALLS terminates. CALLS isolates the word by looking for blanks, commas, or right parentheses. To terminate CALLS, type carriage return on a blank line.

If an unknown keyword is given, CALLS lists the 16 keywords in alphabetical sequence around the given keyword, and then enters interactive mode as above.

CALLS Catalog File

The CALLS catalog is a text file that acts as a database containing keywords and explanatory text. Catalog files may be compressed by the CALLM utility. The default catalog name is actually based on the name by which CALLS is scheduled (that is, the second word in the received runstring). If CALLS is RP'ed under a different name or the .RUN file is renamed, the new name becomes the default catalog name for that copy.

For example, if you issue "`rp calls utils`" and then execute UTILS, it will use the default file named /CATALOGS/UTILS.CALL.

CALLS Directives

The CALLS catalog files may be plain text files in the format given below, but more commonly the final catalog is built by the CALLM utility, which merges together plain text files and performs text compression on the result. Additionally, CALLM can extract CALLS catalogs from comments in source code. (See the CALLM section later in this chapter or enter "? callm" from the CI prompt for more information about the CALLM utility.)

A catalog file consists of explanatory text lines and CALLS directives. The CALLS directives must begin in column 1, and are:

`.topic` *primarykey*`[,`*aliaskey*`,`*aliaskey* `...]`

Begin a new topic. *primarykey* is the official name of this topic, a string of up to 64 characters not containing blanks or commas. *aliaskey* is an alias name following the same syntax; the user can receive help on this topic by specifying either *primarykey* or any of the *aliaskey*s.

The next line of text (that is, which is not a CALLS directive) is used as the one-line description for this topic. This description appears with the *primarykey* when the topic explanation is read, and for any "associated topics" menus.

Any subsequent text lines are printed verbatim by CALLS when this topic is read, until a subsequent ".end" or ".topic" directive is found.

`.group` *key*`[,`*key* `...]`

Used within a topic, joins this topic to a group of related topics given by the named keys. Each *key* may name a primary key used for a topic elsewhere, or may be used solely in the ".group" directives for the related topics. A discussion on related topics appears below.

`.page` Forces a page break ("More..." prompt) at the current location in the explanatory text, if the listing is to the terminal.

`.see` *key*`[,`*key*`,`*key* `...]`

"See also" relates the named keys to the current topic, such that a menu of the named keys is presented after this topic text is read, and the user is invited to select one of these keys for more help. A discussion on related topics appears below.

`.end` Terminates the current topic.

`.include` Directive that is recognized only by the CALLM utility. See the section later in this chapter on the CALLM utility for more information.

Relating Topics to Other Topics

Two of the CALLS directives are used to "relate" topics to other topics:

`.group` for topic groupings.

`.see` for referrals to "see also" topics.

A topic grouping occurs when several topics use the same key in a ".group" directive. When the user requests help on that key, a menu of all the topics that belong to this group is presented.

For instance,

```
.topic help, ?, ??
.group commands
Help!

description of help command
.end

.topic exit, quit
.group commands
Exit this program

description of exit command
.end
```

If the user requests help on "commands", CALLS answers with:

```
The following topics are associated with "commands":

help -- Help!
exit -- Exit this program

Put cursor on desired name or type new name, press return.
```

If "commands" were used as a primary key for its own topic, that text would be listed before the above menu is given. For example, some general information about the "command" entry could be given before the menu of actual commands is presented.

The ".see" directive is used within a topic to refer the user to other topics that may be of interest. A similar menu of those topics is printed after CALLS lists the current text block. For example,

```
.topic NewProduct
one-line description of NewProduct

NewProduct relies heavily on ProductA and ProductB

.see ProductA, ProductB
.end
.topic ProductA
    .
    .
    .
  etc.
```

Index File

The first time CALLS runs on a catalog, and after subsequent updates of the catalog, CALLS builds a file called the index file. CALLS builds the index file in the same directory and with the same name as the catalog, but with type extension ".indx". More specifically, if the index file is missing or has an update timestamp that is older than the corresponding catalog, CALLS rebuilds the index file. CALLS will also attempt to rebuild the index if it appears that the index is invalid for the catalog, even if the update timestamps are in order.

Note The index file contains FMP internal file position pointers into the catalog file for the various topics, plus the keyword list and associated topic groupings. This means that the first person to run CALLS on a catalog after an update must have write access into the catalog directory for the index file to be successfully created. It is suggested that the system manager installing a new catalog immediately run CALLS on the catalog with the "−b" option to build the index.

CD (Change Working Directory)

Purpose: Changes the working directory.

Syntax:
```
CD [-| directory]
CD old new
CD [-p] [directory] (VC+ Only)
```

`-p` when using any of the ksh-style editing modes, the command "`cd ..`" moves the current directory one path component closer to the root directory. The `-p` option preserves the physical path when treating symbolic links. "`cd -p ..`" changes the working directory to the parent directory of the current working directory. This option is only available for the ksh-style editing modes.

Description:

The CD command can take one of two forms. In the first form, it changes the current directory to *directory*. If a dash (−) is specified as the argument, the directory is changed to the previous directory ($OLDPWD). The default for *directory* is the value of the $HOME variable.

The second form of CD substitutes the string *new* for the string *old* in the current working directory name, $WD, and tries to change to this new directory.

When either the CD or WD command is used, the variables $WD and $OLDPWD are updated. The WD command always sets $WD to the physical name of the current working directory. When $VISUAL is set to any of the ksh-style editing modes (EMACS, GMACS, or VI), the CD command preserves the logical naming when treating symbolic links. For any of the other $VISUAL modes, the CD command saves the physical name of the current working directory in $WD. $VISUAL is only available in the CDS version of CI.

Examples:

```
CI> pwd
/niners/lott
CI> cd /raiders
CI> pwd
/raiders
CI> cd -
CI> pwd
/niners/lott
```

Suppose LOTT.DIR exists in both /NINERS.DIR and /RAIDERS.DIR. The "`cd` *old new*" syntax can be used to switch directories. In the following example, the current working directory is /NINERS/LOTT; by substituting the string "nin" for all occurrences of the string "raid", the current working directory is changed to /RAIDERS/LOTT:

```
CI> pwd
/niners/lott
CI> cd nin raid
CI> pwd
/raiders/lott
```

The CD command functions in a different manner depending on how $VISUAL is set. For ksh-modes, CD retains the logical path of the working directory. The same sequence of commands can result in a different working directory if symbolic links are used.

For example, suppose that the following directory structures exists.

```
/work.dir
/work/source.dir
/work/source/subs.dir
/work/source/progs.dir     >   symbolic link to  /source/progs.dir

/source.dir
/source/progs.dir
/source/subs.dir
```

The following sequences of commands have different results:

non ksh-style modes	ksh-style modes		
(always physical)	default (logical)	or	optional (physical)
`CI> cd /work/source`	`CI> cd /work/source`		`CI> cd /work/source`
`CI> cd progs`	`CI> cd progs`		`CI> cd progs`
`CI> pwd`	`CI> pwd`		`CI> pwd –p`
`/source/progs`	`/work/source/progs`		`/source/progs`
`CI> cd ../subs`	`CI> cd ../subs`		`CI> cd –p ../subs`
`CI> pwd`	`CI> pwd`		`CI> pwd`
`/source/subs`	`/work/source/subs`		`/source/subs`

CL (List Mounted Disks)

Purpose: Displays all mounted disk volumes.

Syntax: `CL`

Description:

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

Examples:

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

CLOCK (Access A990 Clock Chip)

Purpose: Accesses the A990 Computer's clock chip.

Syntax: `clock [-Q] [A990] [SET|TM]`

`-Q`	Quiet. Do not output the current times to the terminal. The return values will still be set.
`A990`	Parameter to maintain compatibility with prior revisions of this program. It is ignored.
`SET`	Writes the current system time to the A990's time-of-day clock.
`TM`	Sets the system time to the time read from the A990's time-of-day clock.

Description:

CLOCK is used to read from and write to the time-of-day clock that is part of the HP 1000 A990's hardware. When CLOCK executes, it reads the time-of-day clock and reports the clock's current value along with the current system time.

Return Values:

$RETURN1	=	0	Success.
	=	−1	The processor where CLOCK executed does not have a clock chip.
	=	−2	Invalid option in command line.
	=	−3	The clock chip's battery is depleted.
	=	−4	The clock chip's time has not been set.
	=	−5	You must be a superuser to set the time.

The remaining return values are used to retrieve the hours, minutes, seconds, Julian day of the year, and a timestamp that is compatible with FMP masking (YYMMDD.HHMMSS). If the "SET" option is specified, the values returned refer to the system's time. Otherwise, they refer to the clock chip's time. These fields are always zero when $RETURN1 is −1, −3, or −4.

$RETURN2	=	Hours (0−23)
$RETURN3	=	Minutes (0−59)
$RETURN4	=	Seconds (0−59)
$RETURN5	=	Julian day of the year. (1−366)
$RETURN_S	=	Timestamp compatible with FMP masking (YYMMDD.HHMMSS).

Examples:

```
CI> clock
Clock chip time: Tue Jul  5, 1994  8:54:14 pm
    System time: Tue Jul  5, 1994  8:54:14 pm
CI> echo $return1 $return2 $return3 $return4 $return5 $return_s
0,20,54,14,186,940705.205414

CI> * Set the system time from the A990 clock and then reset the
CI> * access time for each file in /PROGRAMS that has an access
CI> * time that is equal to or later than the time retrieved from
CI> * the A990 clock.  An error is generated for any programs that
CI> * are active, such as TOUCH.
CI> clock -q tm; touch -a /programs/@.@.a$return_s-
Program is active TOUCH.RUN::PROGRAMS:6:185:128
```

CN (Control Device)

Purpose: Controls peripheral devices.

Syntax: `CN` *lu function* [*pram*4*]

lu is the logical unit of the device to receive the control request.

LU 1 controls the user's terminal. To control the system console, indicate LU = 100001B.

function is the control function code (0-63B) as defined in the function field of the control word (*cntwd)* listed for each driver in the *RTE-A Driver Reference Manual*. A two-character mnemonic code can be used for some of the more commonly used control functions. The following is a list of the mnemonics and their equivalent function codes. Note that the action performed for a particular function code is dependent on the driver. Refer to the *RTE-A Driver Reference Manual* for a complete list of the function codes available for a particular driver and the actions that they perform.

Mnemonic	Equiv. Octal Func. Code	Pram 1-4 Definition	Action
AB	none	None	Abort current request (at head of queue)
AD	24B	Device address	Establish new device address
TO	11B	# lines on page	Issue top-of-form or line spacing on printer
RW	4B	None	Rewind cassette tape
EO	1B	None	Write end-of-file
FF	13B	None	Forward space file
BF	14B	None	Backward space file
FR	3B	None	Forward space record
BR	2B	None	Backward space record
DP	25B	1-4 prams	Set device prams
n/a	0	None	Clear device
n/a	20B	Program name	Enable primary program
n/a	21B		Disable primary program
	40B	Program name	Enable secondary program
	41B		Disable secondary program

*pram*4* are the optional parameters that specify additional device details as appropriate for a given driver. Specific meanings for these parameters may be found in the *RTE-A Driver Reference Manual* under each driver.

If an ASCII character string is given as a parameter, it is parsed into 2-byte words and passed to the driver as separate parameters.

Examples:

`CI> cn 4 rw`	(Rewinds the tape in cassette tape unit, LU 4)
`CI> cn 6 to -1`	(Causes a top-of-form, page feed, on printer LU 6)
`CI> cn 7 20B proga`	(Establishes PROGA as the primary program to be scheduled on LU 7 when an asynchronous interrupt is generated)
`CI> cn 16 ad 12`	(Sets HP-IB device address 12 at LU 16)

Refer to the *RTE-A 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* [*option*]

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 Masks" 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. When copying from a device, the default file type is type 3; a different file type must be specified if one is desired. Note that the destination LU should not be a cartridge tape drive.

option The following characters indicate particular actions to be taken:

`A`	ASCII records, no checksum (default).
`B`	Binary.
`C`	Clear backup bit on source after copying (note that backup bits for @.DIR files are not cleared with this option).
`D`	Replace duplicates; existing file with the same name will be replaced.
`N`	No carriage control in source.
`P`	Purge source after copying.
`Q`	Quick; do not record access time on source.
`S`	Preserve directory information (timestamps, protection, and backup bit) of the source file.
`T`	Truncate destination to length of valid data.
`U`	Replace duplicates if update time is older.

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 should be used to default destination file names (unless output is to an LU).

The file mask is a very powerful but complicated tool, and it should be used with caution. For example, you can copy all type 6 files on several different directories to a particular directory, which can be a global directory or a subdirectory. An implicit D qualifier is used whenever you use a wildcard mask. This means that if any directory matches the mask, all files in that directory will also be copied.

The D qualifier can be overridden with the mask qualifiers K or N, which is particularly useful with time qualified copies, because directory time stamps are not maintained. Note that the D qualifier is automatically appended to the unspecified mask and appears in error messages. For example:

```
CI> co /global/@.ftn /new/@.ftna
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 are those of the copying process. However, the update time of the new file is that of the current file, to maintain a history of the latest revision date.

When copying a file to a line printer, the characters in the first column are not printed because they are used by the printer for carriage control. The N option indicates that characters in the first column are printed.

When using the C option, the backup bit on directories is not cleared. The backup bit and the time stamps on directories are never changed, because although directories are structured like files, they are not accessed like files. When a directory is copied, a new directory by the same name is created and all the contents are copied.

The Q option is used when you do not want to have the access time of the file updated. It is useful when you are copying from a file that resides on a write-protected disk. Normally, the file system attempts to update the file access time when it opens the file, and because the LU is write-protected, the CO command would fail.

The S option allows you to save directory information (timestamps, protection, and backup bit) of the source file.

The T option allows a file with wasted space to be copied into a new file as a perfect fit. The end-of-file directory information of the source file is used to determine how many blocks of valid data to copy to the destination file. This option is not used with type 1, 2, and 6 files or FMGR files.

The U option allows overwriting of the destination file, but only if the update time of the destination file is older than that of the source. Because FMGR files do not have update times, they are considered the oldest.

The file type of the destination file is the same as the source file if you do not specify a different one. If the destination file size is not specified, 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 file and the other user is the owner of the destination directory. Otherwise, it will have the protection of the directory into which it is copied.

When copying type 1 files to devices, the device record length is set to 256 bytes. If a different record length is desired on the device, copy the type 1 file into a type 2 file with the desired record length. The type 2 file can then be copied to the device.

When copying type 2 files to or from devices, the record length of the type 2 file is used as the device record length when reading from or writing to a device. When copying from a device to a type 2 file, if records exist on the device that are greater than the record length of the type 2 file, the records are truncated without warning. When the record length of the type 2 file is too large for CO to accommodate, CO reports an illegal DCB buffer size error.

When copying from a device to a type 1 file or from a device to another device, CO reads as large a record as possible from the source device. The input buffer size is dependent on the amount of free memory available to the CIX program. Any records on the source device that have record lengths greater than the input buffer size is truncated without warning.

When copying from a device to a variable-length file, records are truncated to 256 bytes without warning.

For VC+ only, when the variable $QUIET_CMD is set to ON, the message

```
Copying FILE1 to FILE2 ... [ok]
```

is not displayed. Also, there is no message if the U option was specified and there was no copy because the destination file was current.

Examples:

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

This command 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 remain unchanged. Note that all files copied by this directive will be copied to the same directory. To copy subdirectories to subdirectories, use the K qualifier in the mask instead of the S qualifier.

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

This command copies all files with file 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> 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 (that is, type 6 files), a longer file size must be explicitly specified. After copying, the file is truncated to its actual size.

The following example is not allowed because CI does not copy a directory into its own subdirectory:

```
CI> co @.dir.d sub/@.@
```

If CI were to allow copying a directory into a subdirectory of itself, this command would find subdirectory SUB in the working directory and copy it into subdirectory SUB, creating file SUB/SUB.DIR. Then, following the D qualifier, all files in subdirectory SUB would be copied, including SUB/SUB.DIR. This would continue until the name overflows the FMP limit of 63 characters. Therefore, either do not copy from a directory into its subdirectory, or use the N qualifier to disallow copying of subdirectories.

```
CI> co @ /dir/@ t
```

This command copies all the files in the working directory into /DIR/, but only copies as much data as the directory information says is valid.

```
CI> co @ /backup/@ u
```

This command copies into /BACKUP/ the files in the working directory whose update times are newer than the corresponding file in /BACKUP/.

CP (Copy Files and Directory Subtrees; VC+ Only)

Purpose: Copies files and directory subtrees.

Syntax:
```
cp [-F|-I] [-PQRV] file1 dest_file
cp [-F|-I] [-PQRV] file1|mask1 [file2|mask2 ... ] dest_directory/
cp [-F|-I] [-PQRV] file1|mask1 [file2|mask2 ... ] dest_mask
cp [-F|-I] [-PQV] -R directory1/ [directory2/ ...] dest_directory/
```

`-F` Forced copy - purge any existing destination file before each copy without prompting for confirmation. Only write access to the directory is required.

`-I` Interactive copy - issue a prompt requesting confirmation for each copy that would overwrite an existing file. The −I option is ignored if the −F option is also set.

`-P` Preserve - preserve the directory attributes when copying files; ownership of directories is also preserved.

`-Q` Quiet; inhibit error/warning reporting.

`-R` Recursive copy - recursively copy a directory subtree to another directory. If the destination directory already exists, the source directory and all of the files under it are copied to the destination directory. If the destination directory does not exist, it is created and all of the files under the source directory are copied to the destination directory.

When the −R option is specified, symbolic links are copied such that the target points to the same location as the source.

`-V` Verbose mode.

`--` indicates the end of the options (required if *mask* begins with '−').

file1 ...
mask1 ...
directory1/ ... one or more files or directories to be copied.

dest_directory/
dest_mask the directory or destination file mask to which the file or directory is to be copied. When files are copied to *dest_directory*, the files are copied into the directory. When a source mask is specified (*mask1 ...*) or when two or more files are to be copied, the destination must be a directory or a destination mask. When the –R option is specified, the destination directory will be created if it does not already exist.

When specifying directories in the argument list, a directory may optionally be specified with a trailing slash (*dirname/*) or with the type extension (*dirname*`.DIR`). The trailing slash or the type extension is necessary when a file name without a type extension exists that has the same name as the directory.

CP

Description:

The CP command copies files to new or existing file names, or into an existing directory. CP can also copy directory subtrees into an existing directory.

By default, when copying a file to a new destination file, if the destination file already exists and the user has write access to the file, its contents are destroyed. To overwrite files when using the −F option, the user only needs write access to the directory of the files being overwritten.

When CP does not have write access to the destination file or directory, CP reports an error.

If the destination file is a symbolic link to an existing file, by default, CP overwrites the existing file and retains the symbolic link. When the −F option is specified, the symbolic link file is purged before the copy.

CP copies a symbolic link file when the source mask includes the L mask qualifier or when the −R option is specified. By default, when the file being copied is a symbolic link, CP copies the contents of the file pointed to by the symbolic link.

The −R option is required when copying one or more directory subtrees to another directory.

CP will only create subdirectories when the −R option is used or when either the D or the K mask qualifier is specified in the source mask.

When copying type 1 files to devices, the device record length is set to 256 bytes. If a different record length is desired on the device, the user should copy the type 1 file into a type 2 file with the desired record length. The type 2 file can then be copied to the device.

When copying type 2 files to or from devices, the record length of the type 2 file is used as the device record length when reading from or writing to a device. When copying from a device to a type 2 file, if records exist on the device that are greater than the type 2 file's record length, the records are truncated without warning. When the type 2 file's record length is too large for CP to accommodate, CP reports an illegal DCB buffer size error.

When copying from a device to a type 1 file or from a device to another device, CP reads as large a record as possible from the source device. The input buffer size is dependent on the amount of free memory available to the CP program. Any records on the source device that have record lengths greater than the input buffer size are truncated without warning.

When copying from a device to a variable length file, records are truncated to 256 bytes without warning.

CS/80 cartridge tape devices may not be used with CP.

Return values:

$RETURN1	returns the number of files that could not be copied.
$RETURN2	returns the number of files that were successfully copied.

Examples:

```
CI> cp file1 file2 subdir/
```
(Copy FILE1 to SUBDIR/FILE1 and FILE2 to SUBDIR/FILE2; SUBDIR.DIR must already exist and, for symbolic links, files pointed to by the links are copied)

```
CI> cp –r file1 file2 subdir/
```
(Copy FILE1 to SUBDIR/FILE1 and FILE2 to SUBDIR/FILE2; SUBDIR.DIR will be created if it does not already exist and, for symbolic links, the links themselves are copied.)

```
CI> cp –r /dir/sub1/ /dir/sub2/ /destdir
```
(Copy the directory trees /DIR/SUB1.DIR and /DIR/SUB2.DIR to /DESTDIR/SUB1.DIR and /DESTDIR/SUB2.DIR; for symbolic links, the links themselves are copied)

```
CI> cp /dir/@.@.k /destdir/
```
(Copy every file in the /DIR directory to the /DESTDIR directory and preserve the subdirectory structure; for symbolic links, files pointed to by the links are copied)

```
CI> cp –p /dir/@.@.kgl /destdir/
```
(Copy every file in the /DIR directory to the /DESTDIR directory, preserving the subdirectory structure and the directory attributes of each file; for symbolic links, the links themselves are copied)

```
CI> cp 8 data.tar:::2:400:5120
```
(Copy data from the device at logical unit 8 to file DATA.TAR and use a record length of 5120 words as the input and output buffer size)

CR (Create File)

Purpose: Creates a disk file.

Syntax: `CR` *file*

`CR` *file*[*user*]>*node*

file is a file descriptor, up to 63 characters, in one of the following formats:

Standard
[*/dir/*[*subdir/*]]*filename*[:::*type*[:*size*[:*rlen*]]][*ds_port*]

Combined
[*subdir/*]*filename*[::*dir*[:*type*[:*size*[:*rlen*]]]][*ds_port*]

FMGR
filename[:*sc*[:*crn*[:*type*[:*size*[:*rlen*]]]]][*ds_port*]

where:

dir specifies the unique (global) directory for the file. The directory name can be up to 16 characters long, not counting delimiters (slashes).

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 including a file type extension. The file name can be up to 21 characters: 16 characters for the name followed by a period and 4 characters for the file type extension. The file type extension is used to describe the type of information in the file. Standard file type extensions are described in Chapter 3 of this manual.

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

1 Type 1 files are random access files that 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.

ds_port Specifies the node and user for files to be accessed via DS transparency. The format of the ds_port is:

>*node* [*user/password*]
or
[*user/password*]>*node*

where:

>*node* is optional; if not entered, the current node is used.

[*user/password*] is optional; if not entered, the default DS account at >*node* is used.

Note that the square brackets ('[' and ']') shown in the *ds_port* format description are required characters, unlike the square brackets used elsewhere to denote optional parameters.

Description:

The CR command creates an empty file. The minimum information that 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 to which you have write access. Only the owner of the directory can alter the protection status of the file thus created.

CR

Examples:

```
CI> cr /applications/documentation/compiler
```

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

```
CI> 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> 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> 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 is the character string that identifies the directory. It can be up to 63 characters and 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*

The default size is equal to the track size of the disk used, typically 48 or 64 blocks for hard disks and 30 or 16 for flexible disk. The directory size is extended as needed.

lu specifies where to place a global directory. It must be a mounted disk volume. If it is set to zero, the disk 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 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 of a file descriptor.

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

The size of the directory can be specified in the same way as a file is created. There are four directory entries per block, and two directory entries are used for internal information. Thus, if a size of four blocks is specified, the directory can hold 14 file entries (extents require additional entries) before the directory must 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. The maximum size allowed is 64 blocks. Some programs assume that directories contain no more than 32767 files.

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

The default protection for a global directory is RW/R/R. The default protection for a subdirectory is the protection of the directory in which it is created.

CRDIR

Examples:

`CI> crdir jones`	(Create Subdirectory JONES in the working directory)
`CI> crdir jones::::12`	(Create Subdirectory JONES in the working directory with 12 blocks)
`CI> crdir smith/jones`	(Create Subdirectory JONES on Subdirectory SMITH in the working directory)
`CI> crdir /smith/jones`	(Create Subdirectory JONES in global directory SMITH)
`CI> crdir jones::smith`	(Create Subdirectory JONES in global directory SMITH)
`CI> crdir ::HP`	(Create global directory HP on the same LU as the working directory)
`CI> crdir /HU`	(Create global directory HU on the same LU as the working directory)

CRON (Clock Daemon; VC+ only)

Purpose: Executes programs at specified dates and times.

Syntax: `XQ CRON`

Description:

CRON executes programs at specified dates and times. Regularly scheduled programs can be specified according to instructions found in CRONTAB files. Users can also submit their own CRONTAB file via the `CRONTAB` command. Programs that are to be executed only once can be submitted by using the `AT` command. CRON should be executed only once. This is best done by running CRON from the system welcome file ("`xq cron`").

CRON only examines CRONTAB files during process initialization and when a file is changed with the `CRONTAB` command. This reduces the overhead of checking for new or changed files at regularly scheduled intervals.

A history of all actions taken by CRON is recorded in /usr/lib/cron/log.

Caution CRON does not make any adjustments for daylight savings time.

When the system time is changed, CRON always resets to the new time. No scheduling adjustments are made.

CRONTAB (User CRONTAB File; VC+ only)

Purpose: Copies the specified file into a directory that holds the CRONTAB files of all system users (see CRON command).

Syntax:
```
CRONTAB file
CRONTAB -r
CRONTAB -l
```

file is the file descriptor of a CRONTAB file.

`-r` removes the invoking user's CRONTAB file from the CRONTAB directory.

`-l` lists the CRONTAB file of the invoking user.

Description:

CRONTAB copies the specified file into a directory that holds all users' CRONTAB files. When CRONTAB is used to add a file, it causes the CRON utility to re-read all of the CRONTAB files.

The output of CRONTAB can be redirected to an output file by specifying either '>*filename*' or '>>*filename*' in the runstring. The output file specified must be delimited by commas and is position independent. If the file already exists, it will be overwritten. To append to a file, '>>*filename*' can be used. If the file does not exist, it will be created.

Users are permitted to use CRONTAB if their names appear in the file /usr/lib/cron/cron_allow. If that file does not exist, the file /usr/lib/cron/cron_deny is checked to determine if the user should be denied access to CRONTAB. If neither file exists, only superusers are allowed to submit a job. If only cron_deny exists and is empty, global usage is permitted. The allow/deny files consist of one user name per line.

A CRONTAB file consists of lines of at least six fields each. The fields are separated by spaces or tabs. The first five are integer patterns that specify the following:

minute (0−59),
hour (0−23),
day of the month (1−31),
month of the year (1−12),
day of the week (0−6 with 0=Sunday).

Each of these patterns can be either an asterisk (meaning all legal values), or a list of elements separated by commas. An element is either a number, or two numbers separated by a hyphen (meaning an inclusive range). Note that the specification of days can be made by two fields (day of the month and day of the week). If both are specified as a list of elements, both are adhered to. To specify days by only one field, the other field should be set to *.

For example:

`0 0 1,15 * 1`	runs a command on the first and fifteenth of each month, as well as on every Monday.
`0 0 * * 1`	runs a command only on Mondays.

The sixth field can designate an optional numeric field. When the sixth field is a numeric parameter, it specifies the logical unit that should be used as the log LU for the session that is created to execute the command. If omitted, the log LU defaults to the system LU 1.

The remaining field of a line in a CRONTAB file is a program runstring that is executed by CRON at the specified times. The program is invoked from your startup working directory. By default, CRON will upshift the characters in this runstring before executing the program. There are two methods of quoting available to allow characters to pass unaltered to the destination program. A single character is quoted by preceding it with a backslash (\). A string is quoted by enclosing it in backquotes (‘). CRON does not perform any variable substitution (for example, $VARIABLE). If this is required, you can schedule CI and pass it a CI command file.

A comment line is indicated with a pound sign (#) in column 1.

Example:

The following CRONTAB entry executes a CI command file that performs a backup at 11:30 pm on every week night.

```
30 23 * * 1-5 /programs/ci /cmdfiles/backup.cmd
```

The following CRONTAB entry executes DSCOPY on the first and fifteen of every month at 12:00 am. LU 97 will be used as the log LU for the CRON session. Backquotes are required to preserve the case of the target file name, system, and login name.

```
0 0 1,15 * * 97 dscopy /system/backup.log ‘/tmp/log>system[user/pw]‘
```

The following CRONTAB entry executes a program called “test” at 10-minute intervals from 0900 through 0950 every day.

```
0,10,20,30,40,50  9 1-31 1-12 0-6 /programs/test
```

CSYS (Copy System)

Purpose: CSYS copies type 1 (memory image) files from a CS/80 disk to a cartridge tape (CTD) in a memory-based system.

Syntax: `CSYS,`*filename*`,`*tapeLU*`,`*file#*`,[SA:[`*next file#*`]],[`*BCMoffset*`]`

filename is the name of the type 1 file on disk.

tapeLU is the LU number of the CTD to be copied to.

file# is the number of the file VCP will use (this is the "target") to boot the system from the tape.

`SA` is an optional parameter that specifies that the memory-based system will be hidden inside an ASAVE file. This allows ASAVE files and memory-based systems to reside on the same CTD tape. This option uses a 256-block VCP file for the ASAVE header and an additional 256-block VCP file for the trailer. When using this option, *file#* should not be zero.

next file# is an optional subparameter of the SA parameter. It specifies where to put the ASAVE end-of-data record. The default value for *next file#* is immediately after the memory-based system.

To reserve space for more than one memory-based system:

next file# = *last file#* + {truncation to an integer of ((*blk size* + 255)/256)}

where: *last file#* is the file number of the last file to be put on tape.
blk size is the number of disk blocks in the last file on cartridge tape. This value is derived from the information contained in the FMGR or CI DL command. Note that a CTD block is 512 words and a disk block is 128 words.

BCMoffset is the BCM file number. The file # and the BCM offset determine the starting block for the file on tape. For example, if the file # is 1 and the BCM offset is 1, then the file starting block is 288 (256 + 32).

Description:

Files are written to the CTD in a format compatible with the RTE-A VCP (Virtual Control Panel) boot/loader file system organization. This allows you to boot the system from any file you write to the tape into the system.

Note You should boot the tape to verify that the file has been correctly copied to it.

Refer to the *RTE-A System Generation and Installation Manual* for details of the boot procedure and the boot command.

The CTD drive is a high capacity device that requires data to be transferred in a continuous streaming mode. A 64K byte disk buffer called a cache is created in an area reserved on the associated disk drive. The data is thus buffered and transferred to the CTD in a continuous stream. Refer to the *RTE-A Driver Reference Manual* for details of the disk-caching scheme.

CSYS runs only in an online environment with the HP 7908, 7911, 7912, 7914, 7946 disk drives and CTD drive.

Operation

CSYS first checks that the file exists and that the tape LU is that of a CTD. If those requirements are satisfied, CSYS clears the CTD tape cache, starts reading the file in 512-word blocks from the disk file, and writes to the cartridge tape.

The reading process continues until the end of the disk file is reached. When the file is copied, CSYS closes the cache and verifies the blocks just written on the tape drive. After verification, the disk file is closed and CSYS issues the message:

```
CSYS COMPLETED.  xxxx BLOCKS WRITTEN TO TAPE.
```

If you selected the SA option, the following message also appears:

```
ADDING ASAVE FORMAT
```

CSYS does not acknowledge the system BReak command; once invoked, the program continues to completion or until aborted by an error.

Note that CSYS will not run if the CTD LU is buffered by RTE-A (use the FMGR, BL command to check this condition).

Examples

For the following examples !PBV and !XYZ are type 1 files that are 480 blocks long. The CTD tape LU is 24 with select code 27 and HPIB address 0.

Example 1: Enter the runstring as shown.

```
CI> csys,!pbv,24,0
```

CSYS returns the prompt:

```
120 (KB) BLOCKS WRITTEN TO TAPE
```

This puts the program !PBV onto the beginning of the CTD tape. It can be booted by typing:

```
VCP> %bdc0127
```

Example 2: Enter the following commands.

```
CI> csys,!pbv,24,1,sa:5
CI> csys,!xyz,24,3
CI> asave,ta,24|sa,16,ap,ve|ex
```

The first line puts ASAVE header information into CTD tape file 0, !PBV into tape files 1 and 2 and, ASAVE trailer information into tape file 5.

The second line saves !XYZ into files 3 and 4.

The third line appends an LU immediately after the ASAVE trailer information. Note that there are two ways of looking at files on tape. CSYS and the boot loader consider a file to start every 64 tape blocks. ASAVE and ARSTR consider a file to run until an ASAVE trailer record. This means that CSYS files 1,2,3,4 and 5 of this example will be put into the first ASAVE file. The second ASAVE file contains LU 16.

To boot !PBV enter:

```
VCP> %bdc010127
```

To boot !XYZ enter:

```
VCP> %bdc030127
```

To restore LU 16 enter:

```
CI> arstr,ta,24|re,16:2,2,ve|ex
```

Note that LU 16 is restored from ASAVE file number 2. ASAVE file 1 contains !PBV and !XYZ and should not be restored.

Loading CSYS

To load CSYS, use the load file #CSYS and enter:

```
CI> link #csys
```

Error Messages

The following error messages can be generated by CSYS. Unless otherwise specified, errors are fatal and immediately terminate the program.

FILE NOT TYPE 1

The file specified is not a type 1 file; it cannot be copied to the tape in the proper format.

ILLEGAL LU

The tape drive LU specified is the wrong type; it must be a CTD.

WARNING. TAPE WRITE DID NOT VERIFY

Warning only. This is not a fatal error; it indicates that tape verification failed, but the file was completely written to the tape.

TAPE IS NOT LOADED

The tape is not loaded in the tape drive.

TAPE IS NOT INITIALIZED

The tape in the drive was not initialized (certified) by the utility FORMC.

TAPE IS WRITE-PROTECTED

The tape is write-protected and thus data cannot be written on the tape.

INVALID ASAVE FORMAT PARAMETER

Some error was made specifying the SA parameter on *next file#* subparameter.

Standard FMP errors are reported using the common file system error messages (that is, “FILE NOT FOUND XYZ”).

CZ (Display/Modify Code Partition Size; VC+ Only)*

Purpose: Displays or modifies the code partition size of a CDS program.

Syntax: `CZ` *prog* `[<`*seg #*`>|AL]`

prog	is the program name, up to five characters, with an optional session identifier.
seg #	is the number of segments to be included in the calculation of the partition size. This number must be less than or equal to the actual number of segments.
`AL`	is the parameter that specifies that all the segments in the program are to be included in the calculation of the partition size.

Description:

The partition size displayed is in the following format:

```
Partition size=pppp   Program Segs=rrr   Segment Blocks=mmm
```

where:
pppp = partition size required in pages
rrr = number of segments in the program
mmm = maximum number of segments that will be loaded into the code partition at a time.

The program code partition size is modified to a new value determined automatically by the system from the parameter specified. This value is derived by multiplying the number of segments specified (*n* or AL) and the code segment block size. The segment block size is equal to the number of partition pages required (*pppp* shown above) divided by the maximum number of segments (*mmm*).

If the AL parameter is specified, all segments are loaded into memory on dispatch. Note that a shared program must have all the code segments in memory.

Examples:

```
CI> cz prog 5          (Allow 5 code segments to be in a partition at once)

CI> cz prog al         (Load all code segments when the program is dispatched)

CI> cz prog            (Display program segment information)
```

DC (Dismount Disk Volume)

Purpose: Dismounts a disk volume.

Syntax: `DC` *lu*

lu is the positive LU number of the disk volume to be dismounted.

Description:

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

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

DL (Directory List)

Purpose: Lists files in a directory.

Syntax: `DL [`*mask* `[`*options* `[`*file*`|`*lu* `[`*msc*`]]]]`

mask is a field specifying the names of files matching the mask to be displayed. The default is all the files in the working directory.

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

options specify what particular information from the directory will be displayed. They can be listed without any delimiters, in any order.

`A` Time last accessed displayed in the following format:

Wed Jun 30, 1988 9:55:48 am

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

`C` Creation time displayed in the same format as option A.

`E` File type extension (for sorting only).

`F` File type.

`L` Location of the file; displays the block address and LU of the main file entry. The first block on disk 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 is not displayed.

`P` Protection on the file is displayed in the following form (read and write abbreviated to first letter):

owner/other	The directory containing the entities listed does not have a group associated with the owner (rw/r).
owner/group/other	The directory containing the entities listed has a group associated with the owner (rw/r/r).

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

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

`T` Temporary file marked with asterisk (*).

U	Time last updated is displayed in the same format as option A.
W	Words in the file, up to EOF.
X	Files with extents to be marked with an asterisk (*).
Y	Security code (FMGR files only).
Z	Contents of symbolic links.
*	A useful subset of the above (FWNSXP).
!	All of the above.
+	Ascending sort by item specified.
–	Descending sort by item specified.

file|lu is an optional file or LU where the DL output is to be stored.

msc is the master security code for the system. Needed only if the security codes of FMGR files are requested (Y or ! option).

Description:

The DL command displays a list of the files that 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 requires one line per file. If several options are specified, multiple lines per file may be required.

The display is normally sorted by name. There are two sort options: + for ascending sort order and − for descending sort order. Preceding an option specifier with + causes the list of files to be sorted with the lowest value first. Preceding an option with − causes the reverse. If either + or − is specified and not followed by an option specifier, the names are ascending or descending sorted. The default is an ascending sort by name. The number of files that can be sorted depends on the amount of free memory the program has.

If there are too many files, as many as possible are sorted and displayed, then another list of files is sorted and displayed until all the files are displayed. Sizing the DL program scheduled by CI to a larger size increases 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 the last program that accessed that file failed to close it. 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 are 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 + or −, the files are sorted by type extension and file name. The E option is ignored if specified without + or −.

The protection displayed depends on whether the owner of the directory belongs to NOGROUP or another defined group. For example, if the directory's owner belongs to NOGROUP, the group protection is not displayed and appears as RW/R. Otherwise, the group protection is displayed, for example, RW/R/R.

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, no security code is displayed. Note that if the master security code is zero, any value (or no value) can be entered for the *msc* parameter. If necessary, see your System Manager or a superuser for the system security code.

DL

Examples:

```
CI> dl                    (Display all files in the working directory)

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

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

CI> dl /program/          (Display all files in directory PROGRAM)

CI> dl /joe/foo           (Display file FOO in directory JOE)

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

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

CI> dl,,!

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

COPY.REL       *     rw/r/r   5    86   86  6312  127  128 8390/38
create time   Wed Jan 12, 1989  9:16:13 am
access time   Wed Jan 12, 1989  9:47:09 am
update time   Wed Jan 12, 1989  9:39:47 am

COPY.SRC    *  *     rw/r/r   4    92  184 13418  820   38 7908/38
create time   Wed Jan 12, 1989  9:00:33 am
access time   Wed Jan 12, 1989  9:44:29 am
update time   Wed Jan 12, 1989  9:30:35 am
```

The previous 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; an asterisk (*) indicates that the file has extents (X option)

`ba` – backup; an asterisk (*) indicates that the file needs to be backed up (B option)

`tmp` – temporary; an asterisk (*) indicates that the 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 the longest record in the file (R option)

`addr/lu` – block address of the beginning of the file (L option)

`open` – name of the program (if any) accessing the file (O option)

`sc` – security code; displayed only for FMGR files (Y option)

```
CI> dl @ -1*m

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

COPY.REL       rw/r/r   5    86    86  6312  127 8390/38
COPY.SRC    * rw/r/r   4    92   184 13418  820 7908/38

CI> 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.

DT (Display/Modify Data Partition Size; VC+ Only)*

Purpose: Displays or modifies the data partition size of a CDS program.

Syntax: `DT` *prog* [*size* *emaSize* [*msegSize*]]

prog	is the program name, up to five characters, with an optional session identifier.
size	is the size of the data partition in number of pages.
emaSize	is the number of pages needed for EMA if EMA is used.
msegSize	is the size of data space mapped into the EMA if it is used.

Note that the data partition size plus *msegSize* must be less than 32 pages.

Description:

The program named must be dormant. The display format of this command is:

`Last Adr=`*lllll* `Min. Part.=`*pppp* `EMA/WS=`*eeee* `Mseg=`*mm* `VMA=`*vvvvv*

where:

lllll = size of the data segment in words (not including EMA/WS)

pppp = partition size required in pages

eeee = size of EMA or working set area in pages

mm = MSEG size

vvvvv = virtual memory size in pages

The display is the same as the SZ command except that the "`Last Adr`" field (*lllll*) is always on a page boundary.

DT changes the size of the data partition to the number of pages given in segment size (or EMA size). For EMA programs, the data partition and EMA sizes are altered by that number of pages. If the *msegSize* parameter is entered, the MSEG is set to that size. Note that working set and virtual EMA sizes are not changed by DT, but can be changed by the WS and VS commands.

Example:

```
CI> dt spice
Last Adr= 64000 Min.Part.= 112 EMA/WS= 85 Mseg= 1 VMA= 2000
```

ECHO (Display Parameters at Terminal)

Purpose: Displays parameters, separated by commas, at the terminal.

Syntax: `ECHO [`*parameters*`]`

parameters is 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 shifts the input to uppercase, puts commas between the parameters in the string, performs variable substitution, and removes CI quotes (backquotes and backslashes). You can use CI backquotes 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> echo ru edit test.ftn          (Display specified string)
RU,EDIT,TEST.FTN                   (String is displayed in uppercase and commas
                                    separate parameters)

CI> echo $session                  (Displays value of $SESSION)
45                                 (Session number is 45)

CI> wd /mine/temp                  (Set working directory)

CI> echo ‘Your working directory is ‘$wd
Your working directory is /MINE/TEMP
                                   (Display message indicating your current
                                    working directory)
```

EX (Exit)

Purpose: Terminates the Command Interpreter program.

Syntax: `EX [`*option*`]`

option specifies the action to take if programs are active. The values are as follows:

- `B` Make background session (non-interactive, in multiuser environment only).
- `C` Continue execution; do not log off.
- `L` Log off; active programs are aborted.

Description:

EX terminates CI and prints the message "`Finished`".

For HP 92077A systems, no further user action is required after entering this command. For systems with VC+, follow the description given below for the appropriate action to be taken.

If CI is not the primary program, no further action is required after entering the EX command. CI terminates and the father program regains control.

The simplest and most common case is where CI is the primary program and no programs are active. CI schedules or transfers to the logoff program/command file if one is defined for the user; programs RP'd but not active are removed without comment. System utility programs are put in the system session; then CI terminates and the session is logged off.

If CI is the primary program and there are active programs scheduled by XQ or AT commands, CI displays the active programs and prompts for the action to be taken. There are three options available: remain in CI without exiting (C), abort all active programs and exit immediately (L), or run the current session as a background session where active programs continue to run until they finish, but the terminal is freed for other sessions (B).

If the log off (L) option is specified, CI schedules or transfers to the logoff program/command file if one is defined for the user. All active programs that are not system utilities are aborted. Programs RP'd but not active are removed without comment. System utility programs are put in the system session. Then CI terminates and the session is logged off.

If the background session (B) option is specified, CI schedules or transfers to the logoff program/command file if one is defined for the user, changes the current session to a background session, and terminates. Note that if the only program or all remaining programs in the background session are dormant, saving resources, the program(s) will be aborted and the session will terminate.

If EX is entered from a command file, CI follows the same steps it does for the log off (L) option in interactive mode with active programs.

Note If spooling is active in the session when it is about to be terminated in any of the previously mentioned cases, all LU redirection for the session is terminated, and associated spool files are closed and released or passed to the outspool queue prior to terminating the session.

Example:

To exit CI and abort any active programs:

```
CI> EX L
```

To exit CI with active programs:

```
CI> EX
Your programs:
APPL1
PROGX
Continue, Logoff, Background, or ? [C]?
```

FOWN (Report File Space by Owner)

Purpose: Scans the files named by the masks supplied in the runstring, displaying the total disk space used by each owner. Disk space can be reported in kilobytes, megabytes, or blocks. The default display mode reports disk usage in 128-word blocks. FOWN also displays the total disk space used by all files matching the masks.

Syntax: `FOWN [`*options*`] [`*fileMask*`] [`*fileMask2* `...]`

options is one or more of the following (note, however, that the −k and −m options are mutually exclusive):

- `-k` report the disk usage in kilobytes. Selecting this option also changes the value returned in $RETURN_S.
- `-m` report the disk usage in megabytes. Selecting this option also changes the value returned in $RETURN_S.
- `-q` (quiet) inhibit error reporting.
- `--` marks the end of the options. This is only required when the first mask begins with a hyphen (−).

fileMask *fileMask2* ... specifies the files to scan. Refer to Chapter 2 in this manual for a description of file masks. The default mask is `/@.@.sgl`, which displays information about all FMP files on the system. Symbolic links to directories are not followed. This can be analyzed to identify users or classes of files that may be using excess disk space.

Description:

If FOWN is unable to identify the owner of a file, it reports the system number corresponding to the owner in the form "`Unknown` (*number*)". This typically means the file owner is no longer a user on this system. Owners of files on remote systems are always shown as numbers.

The message "`FMGR files not scanned`" indicates that some FMGR files matched the mask but were not counted because FMGR files contain no ownership information.

FOWN reports only the total space used if one owner owns every file in the mask.

The output of FOWN can be redirected to an output file by specifying either "*>filename*" or "*>>filename*" in the runstring. The output file specified must be delimited by commas and is position independent. If the file already exists, it will be overwritten. To append to a file, "*>>filename*" can be used. If the file does not already exist, it will be created.

Return Values

$RETURN1 = 0 FOWN executed successfully.
≠ 0 An error was encountered during the file search.

$RETURN2 the high-order bits (31−16) of the total number of blocks used by all files found matching the mask.

$RETURN3 the low-order bits (15−0) of the total number of blocks used by all files found matching the mask.

$RETURN4 the high-order bits (31−16) of the number of files found.

$RETURN5 the low-order bits (15−0) of the number of files found.

$RETURN_S the total disk space used by all files found matching the mask.

Examples

Example 1: The default mask is used. User number 7 is no longer known to the system.

```
CI> fown
Scanning...    Mask = /@.@.s

  Owner           Disk Blocks  Number of files
DOUG.NOGROUP           6715        1065
DON.NOGROUP            4032         843
MANAGER.SYSTEM         2761         404
DOUGL.NOGROUP           328           3
NAOMI.NOGROUP            69           2
Unknown (#7)            198          58
                       -----        ----
Total                  14103        2375
```

Example 2: All type 6 files are counted and their owners identified. The message "FMGR files not scanned" indicates that some FMGR files matched the mask but were not counted. There is no ownership information available for FMGR files.

```
CI> fown,@.@.e:::6
Scanning...   Mask = @.@.E:::6

  Owner           Disk Blocks  Number of files
DOUG.NOGROUP           2943         362
DON.NOGROUP             433          48
DOUGL.NOGROUP           143           3
                       ----         ---
Total                  3524         413

FMGR files not scanned
```

Example 3: Ownership ID requested for directory /HOME only, thus the format of the output is changed.

```
CI> fown,/home/
Scanning...   Mask = /HOME/

Owner: HOME.NOGROUP Total Blocks: 641 Number of files: 86
```

Example 4: Report the disk usage, in megabytes, and owners of all files on the system. Inhibit error reporting and redirect the output to the file DISK_USAGE.LST.

```
CI> fown -qm >disk_usage.lst
```

Example 5: Return the number of kilobytes used on disk LU 17 in $RETURN_S. Inhibit error messages and redirect the output to LU 0.

```
CI> fown -q -k 17 >0
```

FPACK (File System Pack)

Purpose: Rearranges the files on a disk volume, packing the files together more tightly to increase the size of the largest free space on the volume.

Syntax: FPACK, *lu*

lu is the LU of the volume to be packed.

Description:

When the FPACK operation is complete, there is usually free space at the high end of the volume. After the disk volume is packed, you can run the FREES utility to determine the amount of free space and the largest area of free space.

If the disk volume can be dismounted during the packing process, then MPACK should be used instead. Because MPACK dismounts the volume, it can move all files and directories. Thus it can do a more thorough pack and is much faster than FPACK. See the description of MPACK later in this chapter for more information.

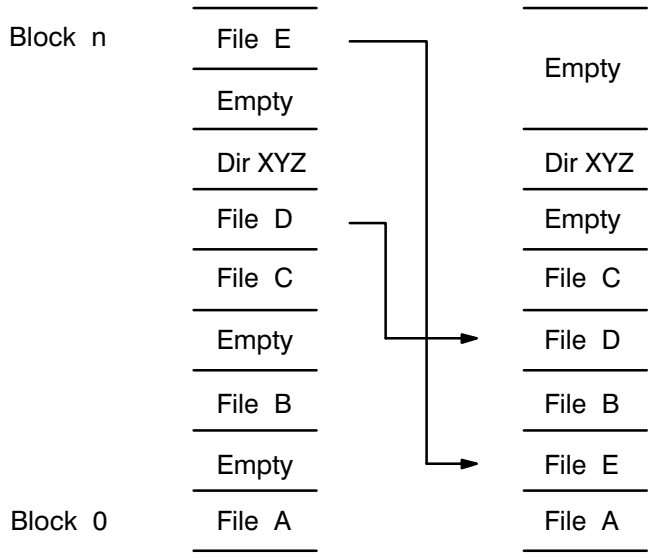
The Packing Process

FPACK first scans the directories and generates a list of files in the order of their location on the disk volume. Then FPACK copies files from higher numbered blocks to free spaces in lower numbered blocks on the disk, 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, and so on) are not changed.

Note that to copy a file, you must have read/write access to the file and to its directory; generally, only a system manager has access to all files and directories.

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 are also not copied.

The process is illustrated below. Assuming that all files are the same size, FPACK converts the disk 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. FPACK skips the directory, then copies File D. This process frees one area of space at the top, enlarging the available free space area on the volume. An area of free space still remains below directory XYZ.

After FPACK moves as many files as possible, it scans the LU again and issues an ordered listing of the files that could not be moved, beginning at the highest disk volume address (a maximum of ten files are listed). The files listed are generally those that cannot be copied, for example, type 6 files, open files, directories, and 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 may run FREES to see the amount of free space that now exists on the disk volume and the size of the largest free space area. (Refer to the description of FREES provided later in this chapter for the utility output format.)

If you still do not have enough free space, you can gain more by moving the appropriate files. For example, you can move down directory XYZ (shown above) to increase the largest free space area. The procedure for moving the directory is provided in the next section. If file C can then be moved, there is even more space in the largest area. (If file C cannot be moved, moving file D does not help.)

Alternatively, you can purge some existing files from the disk volume, and run FPACK again to move files into the now empty space.

Moving Directories

The following sequence of CI commands moves the global directory XYZ to a different disk location. The leading slash in the command argument indicates the named directory is a global directory.

1. Change the working directory to XYZ. This simplifies the command sequence.

```
CI> wd /xyz
```

2. Create the temporary directory TEMP. The directory is created on the same disk volume as the working directory, at the lowest numbered block available.

```
CI> crdir /temp
```

3. Move all 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 defining the entry is issued to your terminal. A successful move is indicated with the notation [ok]. If an entry cannot be moved, the notation [failed] is given, followed by the reason (often an open file on XYZ). If you cannot correct the failure and move the file, move all the files back to directory XYZ (mo /temp/@.@ /xyz/@.@) and stop the attempt.

```
CI> mo @.@ /temp/@.@
```

4. Change the working directory to TEMP.

```
CI> wd /temp
```

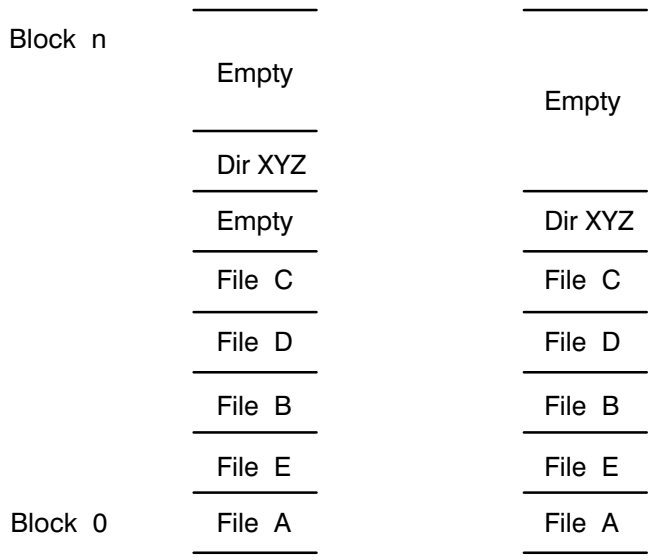
5. Purge the old directory, freeing the disk space. (Note that you must include the type extension .DIR when purging a directory.) If the directory is not empty, it cannot be purged. Move the files back to XYZ, purge TEMP, and terminate the action.

```
CI> pu /xyz.dir
```

6. Restore the original name. The files now have the same name as before, but the directory has moved.

```
CI> rn /temp /xyz
```

After this operation, the example disk volume structure is converted, as follows:



Moving Subdirectories

You can move subdirectories the same way; however, you must specify the hierarchical path, as:

```
CI> crdir /xyz/temp.dir
CI> mo /xyz/sub/@.@ /xyz/temp/@.@
CI> pu /xyz/sub.dir
CI> rn /xyz/temp.dir /xyz/sub.dir
```

The concept is the same: Create another subdirectory, move all the files into it, purge the original, and rename the new subdirectory to the original name.

Moving Files

When you use the CO command to copy a file to a disk volume, the file is placed in the lowest free space large enough to hold it. This allows you to move files on a disk volume to recover free space. If you copy a file to a different name on the same directory, the new copy is moved to a lower free space.

For example, using the original example in this section, you can move file E to the lowest free space with the following command sequence (assuming file E is on directory XYZ).

1. Change the working directory to XYZ to simplify the command sequence.

```
CI> wd /xyz
```

2. Copy file E to file X (which does not exist). File X is placed in the lowest free space available.

```
CI> co e x
```

3. Purge the original file.

```
CI> pu e
```

4. Restore original file name.

```
CI> rn x e
```

When this is done, the example disk volume structure has a larger free space.

FPUT (Bootable System Installation)

Purpose: Installs bootable systems and diagnostics in the space on a file system volume that was reserved by the CI IN command.

Syntax: FPUT *filedescriptor diskLU* [*VCPfilenumber* [*BCMoffset*]]

filedescriptor	is the name of the file to be copied to the bootable area on the CI file system volume given by *diskLU*.
diskLU	is the CI file system volume to which the specified file will be copied.
VCPfilenumber	is an offset within the bootable area in 256-block units.
BCMoffset	is an offset within the bootable area in 32-block units.

Description:

You can use FPUT to install a BOOTEX file that was initialized by INSTL, or to install diagnostics or user-configured memory-based systems in a bootable position on a file system volume.

The VCP bootstring for a SCSI or CS/80 disk is in the form:

%BDC*ffbusc*

where:

ff	is a VCP file number that describes an offset into the bootable area in 256-block units. Default is 0. The *ff* parameter provides the option to select one of several bootable systems stored on the same disk LU. This allows you to, for example, place more than one copy of BOOTEX on the LU, such that multiple RTE-A systems that require different revisions of BOOTEX may be booted.
b	is the bus address of the disk.
u	is the unit number of the disk (always 0 for SCSI or CS/80).
sc	is the select code of the interface card.

As an example, assume that your system has LU 10 on a CS/80 disk that is unit 0, bus address 2, and select code 24. To install BOOTEX on that system, you would first run INSTL to make a copy of BOOTEX that is configured properly for the target system (see the INSTL documentation). Assuming that the output file name is “NEWBOOT” and that it is 768 blocks long, you could then run “FPUT NEWBOOT 10” to copy it to block zero of the target disk. The bootstring to boot the resulting system would be “%BDC002024”. It could be abbreviated to “%BDC2024” because the *ff* parameter defaults to 0.

You may also wish to have a second copy of BOOTEX for redundancy. The FPUT command would be “FPUT NEWBOOT 10 3”. The *VCPfilenumber* parameter is 3 because “NEWBOOT” is 768 blocks long and 768/256 is 3; so the first copy occupies VCP file numbers 0, 1, and 2. Thus, the next available location is 3. This would be booted with ”%BDC032024”.

The *BCMoffset* parameter is used in a similar manner, except that it is in units of 32 blocks. (See the *HP 1000 A-Series Computer Diagnostic Operating and Troubleshooting Manual*, part number 24612-90013, for more information about the BCM--Basic Control Module.)

The absolute block number relative to the start of the disk is given by:

Block = (*VCPfilenumber* * 256) + (*BCMoffset* * 32)

FPUT Operation

FPUT opens the file to make sure it exists. You can install any file, so long as it fits in the reserved space. Note that there is no directory describing the files in the reserved space. FPUT verifies that the passed LU is a file system volume and that you reserved enough space on the volume for this file. Reserve space by using the IN command described in this chapter.

If the file does not fit, FPUT exits with an informative message. Files in the normal portion of the volume are not affected.

Caution Be aware that FPUT overwrites any previous information in the reserved space without warning. For example, suppose BOOTEX, which requires 768 blocks, is installed at offset 0 of the disk. If a memory-based system is installed at offset 1 (starting at block 256), FPUT overwrites part of BOOTEX without warning.

This error is not detected until the next time you try to boot the system, when the cause may easily be forgotten. It is your responsibility to remember what is in this reserved space. The easiest thing to do is put BOOTEX at block 0 (INSTL will do this for you) and not put anything else there.

FPUT does not install files on an FMGR cartridge. You can place files on an FMGR cartridge in a bootable position by controlling their placement in the directory. Put the first bootable file on the cartridge first (this is where BOOTEX is usually found). Additional bootable files can be placed immediately after the first one. Make sure each bootable file is on a 768-block boundary by padding shorter files with empty dummy files.

FREES (Indicate Free Space on a Volume)

Purpose: Scans the free space table on hierarchical file system disks, and reports the amount of free space and the size of the largest free space. The amount of free space on a volume is an indication of how much more data will fit on the volume, while the largest free area defines the largest file that can be created on that volume without first packing the disk.

Syntax: `FREES [`*options*`] [`*diskLU*`] [`*diskLU*`:`*diskLU*`] ...`

options is one or more of the following:

`-v` inhibit the inverse video bar graph.

`+l:`*lu* sends the listing to the given LU.

`-g` bar graphs are not relative to the largest disk.

disksz where *disksz* is one of the following:

`+t` report disk sizes in number of tracks.
`+m` report disk sizes in megabytes.

sort where *sort* is one of the following:

`-s` inhibit sort, report will be in cartridge list order.
`+h` sort by largest hole size.
`+d` sort by disk size.

quiet where *quiet* indicates quiet mode and is one of the following:

`+q` quiet mode, return the free space information to the scheduling program in $RETURN_S.
`+qd` return the disk size in $RETURN_S.
`+qf` return the free size in $RETURN_S.
`+qh` return the largest hole size in $RETURN_S.
`+qr` return the reserved size in $RETURN_S.
`+q%f` return the percent free in $RETURN_S.
`+q%m` return the percent max in $RETURN_S.

diskLU specifies a disk LU to display; it can be repeated.

diskLU:*diskLU*
specifies a range of disk LUs to display; if no disk LU is specified, all CI volumes are listed.

Description:

Running FREES with a ? or ?? (`RU,FREES,?[?]`) causes FREES to display usage information and examples as shown below.

At the user's option, FREES expresses size in number of blocks (the default), number of tracks (+t option), or in megabytes (+m option).

At the user's option, FREES sorts the disks by descending free size (the default), by largest hole size (+h option), or by total disk size (+d option), or the sort can be inhibited to list the disks in cartridge order (−s option).

FREES reports the total free space as an absolute number and also as a percentage of total space on the volume. The only way that this number can be increased is to purge unnecessary files from the disk.

The size of the largest free area (the hole size) is reported as an absolute number and also as a percentage of total free space on the volume. The lower this percentage, the more fragmented the volume is and the more effective the FPACK or MPACK utilities will be in packing the volume.

Following the space table is a short summary that totals all the disk space and free space on the scanned disks.

Unless inhibited by the −v flag, FREES overlays the columnar output with a bar chart to indicate graphically the amount of free space. FREES has two methods of bar chart display, Global (default) and Local (−g option). In the Global mode, the widths of the bars are shown proportional to the largest disk that was scanned. Think of it as a form of autoscaling. In the Local mode, the widths of the bars are computed relative to the individual disks. Note that if only a single disk is displayed there is no difference between Global and Local modes.

In the bar chart, the total width of the bar is proportional to the total free space, while the highlighted portion is proportional to the amount of fragmentation. Here is an example:

```
ru,frees 10 11

LU      Total     Resvd     Free   Max Free %Free %Max
---+----+----+----+----+----+----+----+----+----+----
11    100000         0    80000    40000     80   50
10     10000      1538     8000     4000     80   50


%Free     = free space as percentage of total space.
%Max      = largest free space as percentage of total free space.
Total storage on all disks scanned is 110000 blocks; 88000 free blocks; 80% free.
```

Note that although there is 80% free space on both disks, the bar for LU 10 is much shorter than the bar for LU 11 because the default Global mode was used. For both LUs, half of the bar chart is highlighted because the largest free chunk is 50% of the total free space.

```
ru,frees 10 11

LU      Total     Resvd     Free   Max Free %Free %Max
---+----+----+----+----+----+----+----+----+----+----
11    100000         0    80000    40000     80   50
10     10000      1538     8000     4000     80   50


%Free     = free space as percentage of total space.
%Max      = largest free space as percentage of total free space.
Total storage on all disks scanned is 110000 blocks; 88000 free blocks; 80% free.
```

In this example, both bars are the same width because this is a Local mode display and both LUs have the same percentages of free and fragmented space.

When the +l option is used to send the output to a printer the −v option is automatically invoked and a form feed is generated at the end of the report. The −v option is also automatic if the driver type for the user's terminal is not type 5 (HP protocol).

FREES

Return values:

$RETURN1:	=	0	FREES executed successfully.
	≠	0	an error was encountered during the scan of a disk LU.
$RETURN2:			the number of disk volumes successfully examined.

The following return variables are set when quiet mode is enabled.

$RETURN3 :	disk LU (when reporting on a single disk LU).
$RETURN4 :	high order bits (31-16) of the value returned in $RETURN_S.
$RETURN5 :	low order bits (15-0) of the value returned in $RETURN_S.
$RETURN_S:	this string contains all or part of the free space information depending on the "+q" option selected.

Examples:

`CI> frees`	(Show free space information on all mounted volumes)
`CI> frees 54`	(Show free space information on volume 54)
`CI> frees 54 22`	(Show free space information on volumes 54 and 22)
`CI> frees 10:15`	(Show free space information on volumes 10 through 15)

FSCON (File System Conversion)

Purpose: Converts the directory structure of a 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, and so on). Note that unconverted programs may have difficulty accessing the converted files.

Syntax: `FSCON,` *lu*

lu is the LU of the FMGR cartridge to convert.

Requirements for Successful Conversion

Before beginning the conversion, FSCON checks to see if all the conditions described below are met; if not, FSCON terminates with an 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 upon such factors as the number of files in the directory and the number of extents. You can usually free enough space by purging unneeded files and using the FMGR PK command to pack the disk before calling FSCON. If there is not enough disk space for the conversion, FSCON exits with the following message:

```
Not enough free space on disk
```

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 disk cartridge exceeds this size, FSCON terminates with the following message:

```
Disk too big to convert
```

The disk must be dismounted before conversion, to preclude the possibility of any open files, swap files, or active type 6 files. If the disk cartridge is mounted, FSCON terminates with the following message:

```
Cartridge must be dismounted
```

FSCON only converts FMGR cartridges. Any other cartridge causes the utility to terminate with the following message:

```
Doesn't look like an FMGR disk
```

The Conversion Process

FSCON scans the directory on the given LU and builds a new directory in the unused space at the end of the disk, before the FMGR directory. At the same time, it builds a free space table. The FMGR directory is scanned once to determine whether it is possible to do the conversion, and scanned again to 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 is still in place, and all the files remain unchanged.

The final step in the conversion process is to overwrite the FMGR directory with the hierarchical CI directory. This is done in a single disk write operation. The new directory name is the CRN of

the FMGR cartridge. In this way, the name of a file remains unchanged (except as noted in the following section). For example, a reference to &SORC::DB accesses the same file before and after the conversion.

After the successful conversion, FSCON informs you "Cartridge converted". If an error occurs during conversion, FSCON issues the appropriate error message, followed by "Cartridge not converted", and terminates.

File Renaming

If a file name includes a slash (/) or a period (.), those characters are changed to "|" and "~" respectively. This is necessary because in the CI file system, a slash delimits directories and subdirectories, and a period delimits the type extension and mask qualifier; therefore, a file name containing those characters would not be found. As each file name is changed, the following message is displayed:

```
Renaming <name> to <name>
```

After conversion, you may list the files with the DL command. For example:

```
DL,@~@::dir *list all files with "." changed to "~"

DL,@|@::dir *list all files with "/" changed to "|"
```

Converted CI Directory Entries

When a new CI directory is built, 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 *RTE-A System Manager's Manual* for the format of the file directory entry):

Word			Meaning
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 occurs, FSCON issues the related message and terminates:

Cartridge not converted

This message appears 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 disk LU.

Cartridge must be dismounted

The cartridge to be converted must be dismounted, which guarantees there are no open files, active type 6 files, or the swap file on this cartridge.

Disk too big to convert

Disk has more than 128K blocks.

Doesn't look like an FMGR disk

FSCON only converts FMGR disks into CI file system disk volumes.

Insufficient memory, size up program

FSCON uses free space for tables, and runs faster with more memory. Sufficient memory for at least one track of the disk is required. If the LU contains many type 1 or type 2 files with extents, more free space is required. Resize the program.

Not enough free space on disk

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 disk, 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.

FUNCTION (Define a Function; VC+ Only)

Purpose: Defines a CI function.

Syntax:
```
FUNCTION [-x|+x] function_name {
  command_line1
  command_line2
  :
  command_lineN
  }
```

`-x`	if the user session has an Environment Variable Block (EVB), this option "exports" a defined function or defines a new one as exported (also known as being in the environment). An exported function is available to all copies of CI in the session.
`+x`	imports an exported function or defines a new function as imported (also known as being a local function). An imported function is defined only for the current CI. If neither the −x or +x option is given, the function is defined as a local alias.
function_name	is a string of up to 32 letters, digits, and underscores, not starting with a digit.
command_line1 − *command_lineN*	are CI command lines.

Description:

A function is essentially a memory-resident CI transfer file. It can include positional parameters and logical constructs. When the name of a function is given in a command line, CI transfers control to it as if to a command file. A function can be defined either interactively or from within a command file.

Note that only one function can exist of a given name, either local or exported.

If a function is defined interactively, the user prompt changes to indicate what command must be entered to terminate. This is also the case for the conditional constructs. The closing brace must be the first character on a line followed by a space, comma, semi-colon, or carriage return to be valid. If followed by a semicolon, the next command is to be executed. If it is followed by a space or comma, the remainder of the line is to be ignored.

Also, syntax checking is done in the definition of a function (as is done in CI transfer files) and if the function has errors, the function is not saved.

The precedence within CI is: aliases, internal CI commands, functions, or implied TR or RU (that is, transfer or run files). In order to by-pass this, for example, to have an alias use a CI command, the command must be preceded by a backslash (\) and be in uppercase.

All defined functions can be displayed via the FUNCTIONS command.

A function can be deleted with the UNSET command, −f option.

Examples:

```
CI> function flink {
} > if ftn7x $1.ftn – –
THEN > then
FI > link $1.lod
FI > else
FI > li $1.lst
FI > fi
} > }

CI> function dll {
} > dl $1 –ufs
} > }
```

Using aliases and functions, it is possible to redefine a CI command. The following example shows how to redefine WD to include changing the prompt of the current working directory. Note the quoting of WD in the function.

```
CI> alias wd my_wd
CI> function my_wd {
} > \WD $1 $2
} > set prompt $WD‘>‘
} > }
```

The result is:

```
CI> wd /system
/SYSTEM >
```

Note that in order to have a function that does nothing, at least one line containing a blank must be entered. For example, the following will NOT be saved:

```
CI> function nop {
} > }
```

FUNCTIONS (Display Functions; VC+ Only)

Purpose: Displays all or specific defined functions.

Syntax: `FUNCTIONS [+x|-x|`*function_name*`]`

`+x`	display only local functions.
`-x`	display only export functions. This option is only applicable if the session has an Environment Variable Block (EVB).
function_name	is a string of up to 32 letters, digits, and underscores, not starting with a digit.

Description:

When all functions are displayed, the local functions are displayed first, followed by a blank line, then the exported functions are displayed. The blank line is displayed whether or not functions of either type are defined.

Examples:

```
CI> functions                (Display all defined functions)

CI> functions -x             (Display all exported functions)

CI> functions +x             (Display all local functions)

CI> functions foo            (Display the definition of the function FOO;
                              assuming it is defined)
```

FVERI (File System Verification)

Purpose: Verifies a hierarchical file system disk LU by scanning the directory and table structures and reporting inconsistencies. FVERI can also fix some of the inconsistences found by the verification.

Syntax: `FVERI [`*lu*`|`*mask*`] [`*options*`]`

lu is the LU of the volume to be verified.

mask is the mask describing a directory in which all files and/or subdirectories are to be verified.

options is one or more of the following options in any order:

`+L,`*file*`|`*lu*	list file or device for errors
`+B`	verify bit map only; illegal if a mask is also specified
`+FB`	fix the bit map
`+FD`	fix illegal directory entries (set purged)
`+FF`	fix file directory information
`+OK`	perform fixes without asking each time

Description:

Running FVERI with a ? or ?? (`RU,FVERI,?[?]`) displays 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.

Two kinds of checks are done:

- Internal consistency of the directories and correctness of the bit map (the bit map is a table on the volume that keeps a record of used and free space on that volume).

  FVERI checks directory consistency by looking for legal values within the entries: extent pointers to valid extents, data pointers having legal disk 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.

- Consistency between directory entries and their associated files (for example, the number of records in the file and number of words in the file), and internal consistency of the files (valid record length, EOF marks, and so on.).

  FVERI checks the consistency between directory entries and their associated files, and checks internal consistency of the files, by reading through each file it finds (using normal FMP calls). It collects appropriate data as it reads (the number of records in the file, number of words, and so on), then compares what it finds with the directory entry for that file.

Three kinds of fixes are done:

- Inconsistencies exist between directory entry information and the data actually within the file. For file type 3, 4, 5, 7, and up, directory information may become inaccurate because of misuse of the file system. Note that the user must have appropriate capability to change the directory and file for any of these fixes.

  FVERI can correct an erroneous word count (end-of-file position), record count, and record length that is kept in a file's directory entry.

- Illegal directory entries (entries not recognized by the file system). Unrecognized entries within a directory can confuse particular functions of the file system.

Note Such directory problems may be a sign of a bigger problem within the file system. For example, an overwritten directory may have been caused by a corrupted bit map. Use this fix carefully. Only superusers can use this fix, and the disk must be dismountable.

  FVERI can change an illegal directory entry to look like a purged entry, so that the file system will not be confused by it.

- Bit map corruption. Occasionally a bit map can become corrupted when the directory structure is still intact. For "free space marked used" there will be space on the disk that will never be accessible. For "used space marked free" there is data loss possibility. FVERI cannot correct a duplicate use of space problem.

Note Such directory problems may be a sign of a bigger problem within the file system. Use this fix carefully. Only superusers can use this fix, and the disk must be dismountable.

  FVERI can overwrite the bit map on the disk with the bit map that was created by scanning the directory structure.

If an LU is specified in the runstring, and the +B option is not given, FVERI will perform both types of verifications. With the +B option, FVERI will only perform the first type of verification. With a given mask, FVERI will verify all files or subdirectories in the directory described by the mask, performing only the second type of verification. Because the bit map covers the entire volume, the bit map verification 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. For this reason, if you want to verify the overall integrity of a disk volume, but do not need to verify 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 still be relatively fast because you are not asking FVERI to verify the whole volume.

FVERI 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 disk, FVERI is best executed by a System Manager (superuser).

FVERI is most efficient when verifying a disk 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 bit map is verified, the bit map directories may appear to be invalid. For this reason, when FVERI is asked to fix directories (+FD) or the bit map (+FB), it will first dismount the LU, lock it, and then remount it. This method guarantees that no one else has access to the disk, so FVERI's data is reliable.

FVERI does not verify or fix FMGR cartridges.

Error Recovery

There are several possible responses to inconsistencies detected in the file system. Some problems may be related to, or caused by, others, so recovery should be done with caution. For example, invalid data pointers will probably cause bit map inconsistencies. In this case, fixing the bit map could allow lost data to be overwritten, so that all opportunity for recovery would be gone.

- The fix options in FVERI can correct some of the problems. Directory information about a file, invalid directory entries, and corrupted bit maps can all be corrected.
- Some directory problems can be corrected by creating a new directory on the same disk, moving the troubled directory's files into the new one, purging the old directory, and then renaming the new one.
- Some directory and bit map problems can be corrected by doing a logical backing to tape or disk, initializing the corrupted LU, and restoring the files. This can be time consuming, and some of the logical backup utilities (TF/FST) use the word count (EOF position) to determine how much of a file to backup. Make sure no "number of words" errors exist on the LU before doing the backup.

FVERI can fix inconsistencies related to particular errors using specific options.

- Error messages that can be fixed with +FF

  `(3) Record length incorrect at block <`*nnnn*`>`

  `(3) Number of words in file incorrect at block <`*nnnn*`>`

  `(3) Number of records in file incorrect at block <`*nnnn*`>`

- Error messages that can be fixed with +FD:

  `(5) Unidentifiable directory entry at block <`*nnnn*`>`

- Error messages that can fixed with +FB:

  `(4) Free space is marked as used space`

  `(9) Used space is marked as free space`

Error Messages

FVERI is susceptible to all FMP errors, which are reported as 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 4; some, usually protection violations, are reported with a severity level of 0. 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 disk where the bad data was found. This is either the block number of the directory entry for some file, or the block on the disk represented by the invalid data in the bit map.

Where possible, the error output will indicate the file whose directory entry is corrupt. This gives some clue to the logical part of the disk 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 FOO.TXT : : JOE. This directory entry is at block 1435 of the disk.:

```
(6) Total blocks in the file less than main size at block 1435
    On file FOO.TXT : : JOE
```

The following list of error messages is arranged in ascending order of severity, from 0 to 9. The first four level 0 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 disk.

This error occurs if the volume is not a CI disk or the volume header is corrupt.

(0) Disk volume not mounted.

The volume must be mounted to be verified. If you cannot mount the disk, FVERI cannot give you any further information.

(0) Record Length exceeds 512 bytes at block <*nnnn*>.

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 record length or word count will not be done for this file.

(2) Directory Tag field is incorrect at block <*nnnn*>.

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.

(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. This message can be caused by another program having the file open, the file being created in a non-standard way, or misuse of the FMP calls. Note that FVERI can fix this problem by use of the +FF option.

(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. This message can be caused by another program having the file open, the file being created in a non-standard way, or misuse of the FMP calls. Note that FVERI can fix this problem by use of the +FF option.

(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. This message can be caused by another program having the file open, the file being created in a non-standard way, or misuse of the FMP calls. Note that FVERI can fix this problem by use of the +FF option.

(3) Directory header/trailer 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 does not 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

The bit map has some disk space marked as used. However, the file system does not have any files or directories in that space. The space is wasted and unrecoverable through FMP. This message can be caused by having read-protected directories FVERI could not verify. FVERI will work better if run by a system manager. Note that FVERI can fix this problem by use of the +FB option.

(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. Note that FVERI can fix this problem by use of the FD+ option.

(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 the extents is not the same as the total number indicated in the file's directory entry.

(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) Directory extent back pointer is wrong at block <*nnnnn*>

Directory extents have a pointer which points back to the previous extent or the main. The pointer 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*>

This file's type is less than or equal to zero.

(7) Invalid directory extent pointer at block <*nnnnn*>

The pointer to the next or previous directory extent points to a disk address beyond the valid address space on this disk.

(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 disk.

(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 disk.

(8) Invalid extent pointer in main at block <*nnnnn*>

The pointer to the first extent of this file points to a disk address beyond the valid address space of this disk.

(8) Illegal directory size at block <*nnnnn*>

Each directory extent must be from 1 to 64 blocks long. This directory is not in that range.

(9) Duplicate use of disk block <*nnnnn*>

Two or more files are stored on the same section of the disk. At least one of the files must be corrupt. This message can occur as a result of other file activity on this disk while FVERI is running.

(9) Blocks per bit value is illegal at block <*nnnnn*>

The bit map represents up to 128 blocks of disk data per bit in the bit map. The actual number of blocks per bit must be a power of 2. There can be no more than 8192 words in the bit map., Finally, if blocks per bit is larger than 1, the number of words in the bit map should be greater than 4K (otherwise blocks per bit should have been half as large). If this value is wrong, the allocation of space on the disk can be corrupt.

(9) Used space is marked as free space at block

There is a space on the disk pointed to by a directory entry; however, it is not marked used in the bit map . This space is likely 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 disk while FVERI is running. Note that FVERI can fix this problem by use of the +FB option.

GO (Resume Suspended Program)*

Purpose: Resumes execution of a suspended program.

Syntax: `GO [`*prog* `[`*pram*5*`]]`

prog is the name of the suspended program. The session identifier is optional.

*pram*5* are the parameters to be passed to the program only if the program has suspended itself.

The GO command resumes execution of a program that was suspended by an SS command or by an EXEC 7 call. It is ignored if the program is not in the operator suspend state. Parameters are ignored if the program was suspended with the SS call or if the program does not require them. Parameters may be passed to a program that has suspended itself with an EXEC 7 call. Refer to the *RTE-A Programmer's Reference Manual* for details.

If prog is not specified and the startup program (usually CI) 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 GO command is executed on the last program. The only exception is if the last program is a system utility program, in which case the program that scheduled it is continued.

Example:

```
CI> go progb
```

GREP, FGREP (Search a File for a Pattern)

Purpose: Searches a file for a pattern. These commands support redirection.

Syntax: `GREP [`*options*`] [`*expression*`] `*file*`|`*mask* `...`

`FGREP [`*options*`] [`*string*`] `*file*`|`*mask* `...`

options one or more options listed below. Options that do not require an argument may be grouped together after a single hyphen and may be in any order. To specify an option that requires an argument, the option must appear either by itself (for example, `-e` *expr*) or at the end of a group of options (for example, `-nbe` *expr*).

- `-b` Each line is preceded by the block number on which it was found. Block numbers are calculated as follows: (the number of bytes that have been read from the file)/256; the result is rounded down.
- `-c` Only a count of the lines containing a match is printed.
- `-d` Directory display mode. Displays directory names on separate lines.
- `-e` *expr* Same as the expression argument in the runstring. Must be used when the expression begins with a hyphen (−). May also be used multiple times to specify more than one expression for which to search.
- `-g` */expr/* Same as the −e option, but the expression is delimited by slashes. Useful when the expression contains a hyphen (−) or a comma (,).
- `-h` Display help information for GREP or FGREP.
- `-l` Only the names of files that contain a match are listed. File names are listed once, each separated by a new line.
- `-n` Each line that contains a match is preceded by its relative line number in the file starting at line 1.
- `-r` Right justify block numbers (−b) and/or line numbers (−n).
- `-s` Suppress error messages produced for nonexistent or unreadable files.
- `-t` Search only text (types 3 and 4) files.
- `-u` Case sensitive match. Note that you must protect the search pattern from CI, see below.
- `-v` All lines that do not contain a match are printed.
- `-x` Exact match; only lines matched in their entirety are printed (FGREP only).
- `-z` Display the file name of each file being searched.
- `--` Marks the end of the options.

expression string is the regular expression pattern (GREP) or string (FGREP) to be used for the search pattern. Spaces and upper and lowercase may be protected from CI by enclosing the pattern in backquotes (‘), for example `‘This Pattern‘`. If backquotes are not used, the string is upshifted and commas are inserted as delimiters. For information on regular expressions, see the *EDIT/1000 User’s Manual*, part number 92074-90001.

file|*mask* is the file(s) to be searched; up to 10 file names or masks may be specified.

Description:

GREP and FGREP search files for lines matching a given pattern. By default, each line found is copied to the session LU. GREP supports regular expression pattern matching in very much the same way as EDIT/1000. FGREP patterns are fixed strings, making it a fast and compact means for finding known text strings.

The output of GREP and FGREP can be redirected to an output file by specifying either “>*filename*” or “>>*filename*” in the runstring. The output file specified must be delimited by commas and is position independent. If the file already exists, it will be overwritten. To append to a file, “>>*filename”* can be used. If the file does not already exist, it will be created.

If an output file is not specified, GREP breaks the output into screen pages. Paging is disabled when an output file is specified. For example, to output to the terminal without paging, specify “>1” on the command line. Multiple redirection strings may occur in the runstring; however, only the last redirection is executed.

An input file name can also be specified in the runstring with the “<*filename*” syntax. This must also be delimited by commas and is position independent. This is only used for the input file when no files or masks are specified in the runstring.

Any file or mask starting with either a “<” or a “>” character must be specified with either a relative or absolute path (for example, “`./>@`” or “`/dir/>@`”). Expressions starting with either of these characters must be specified with the `-e` option (for example, “`-e>`*expr*”).

The default condition for GREP and FGREP is to ignore upper and lowercase distinctions during comparisons.

Return values:

GREP and FGREP returns the following values:

$RETURN1 = 0 if any matches are found
= 1 if no matches are found
= 2 if an error occurs

$RETURN2 The total number of matches is returned as a 32-bit integer; where
$RETURN3 $RETURN2 contains the high-order bits (bits 31-16); and $RETURN3, the low-order bits (bits 15-0).

$RETURN4 The total number of files found containing matches.

Examples:

The following example will search the file EIO for the patterns "−weird,pattern" and "eye". Lines will be displayed with the line number and file access errors will be suppressed. Note that when the −e or −g option is used, the command line may not contain the *expression*|*string* argument.

```
CI> fgrep -ns -g/-weird,pattern/ -e eye eio
```

The following example will search the GREP help file for lines matching the regular expression given and write the output to the file OUTFILE. Note that the expression contains spaces and is case sensitive requiring the −u option and backquotes to protect the expression from CI.

```
CI> grep -u >outfile '\-[a-z] [ e/f]' /help/grep
```

The following example will search files matching the mask −o for the −e pattern. Note that the "−−" flag is required when the pattern contains a leading hyphen.

```
CI> fgrep -- -e -o
```

IF-THEN-ELSE-FI (Control Structure)

Purpose: Allows decision making in a command file or a function.

Syntax:

```
IF command_list1
THEN command_list2
[ELSE command_list3]
FI
```

command_list is a list of commands. Commands may be entered either one per line or several per line separated by semicolons. A command list can be null.

Description:

The IF-THEN-ELSE-FI control structure lets you control execution of a command file or a function. 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 it is zero (the command was successful), CI executes the THEN branch. If it is non-zero (the command was unsuccessful), 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 reaches 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.

You must end the IF-THEN-ELSE-FI control structure with FI; otherwise, CI does not recognize that it is finished and continues to process succeeding commands as though they were part of the THEN or ELSE command list. Therefore, if an IF-THEN-ELSE-FI control structure was just executed and CI is not executing commands as expected, check to see if you entered an FI command to terminate the control structure.

IF-THEN-ELSE-FI

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> if dl test; then co test /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 Disk Volume)

Purpose: Prepares a blank disk volume for use in the system.

Syntax: `IN` *lu* [*blocks* [`OK`]]

lu	is the LU number of the disk volume to be initialized.
blocks	specifies the number of blocks at the beginning of the disk LU to be reserved for the boot extension and diagnostics. The default is no reserved space. Refer to the *RTE-A System Generation and Installation Manual* for details.
`OK`	is the optional parameter that suppresses the user prompt, indicating that the command should be executed as entered.

Description:

This command is used to clear a disk volume, eliminating all its files. Before reinitializing a disk volume with files on it, a prompt is displayed so that you can confirm your intent. A "yes" response must be entered to start the process. The OK parameter can be used to suppress this prompt. After the disk volume is initialized, it will be mounted to the file system.

The number of reserved blocks specified will be rounded up, if necessary, to an even multiple of the number of blocks per bit defined for the bit map for the volume. The FREES utility may be used to display the actual number of blocks reserved.

In the VC+ environment, only a superuser can initialize a disk volume with Security/1000 turned off. When security is installed and turned on, only a user with capability level 30 can initialize a volume owned by anyone or an uninitialized disk. Refer to the appendixes in the *RTE-A System Manager's Manual* that describe the Security Table for details on the capability levels within the IN command.

To initialize a disk volume for use with FMGR files, you must run FMGR and use the FMGR IN command. Refer to the FMGR description in this manual for details.

INSTL (Initialize BOOTEX File)

Purpose: Initializes the boot extension (BOOTEX) file for an RTE-A disk LU. You may run INSTL on an RTE-A or RTE-6/VM system.

Syntax: `INSTL [`*snap*`] [`*system*`] <`*boot dest*`> `*lu*` <`*boot source*`> [,`*option*`]`

snap is the name of the snapshot file output by the generator. This is the snapshot file of the target system.

system is the name of the system file output by the generator. This is the system file of the target system.

If you do not enter the *snap* and *system* parameters, the default is to configure the boot extension file according to the specifications of the current (host) system (RTE-A host only).

boot dest is the name of the file to contain the new boot extension (destination file). This is a file in the target system. When using the INSTL command, specify the full path in the boot destination file. Otherwise, it defaults to the current working directory.

If the specified file exists as a type 1 file at least as long as the source file, INSTL puts the boot extension code into it. (If the file exists, but is not type 1 or is too small, INSTL terminates.) If the specified file does not exist, INSTL creates it and fills it with the boot extension code.

If you are booting from a CI volume, *boot dest* can be a number. If *boot dest* is a number, it is assumed to be an offset into the reserved space on the destination LU. Space on a file system volume is reserved by the CI IN command and reported by the FREES utility. This space is used for bootable files (such as BOOTEX and diagnostics).

This offset specifies where in the reserved space to put the file. It corresponds to the file number in the VCP bootstring (see your computer reference manual for information on the VCP bootstring). Given an offset of 0, the file starts at block 0 on the disk. If the disk LU is the first physical LU on the disk and is unit 0 (which CS/80 disks are), HP-IB address 0, and the HP-IB is at select code 27, then this file can be booted using "%BDC0027". An offset of 3 starts the file at block 768 and it can be booted using "%BDC30027". Note that there is no directory describing the files in the reserved space.

Be aware that INSTL overwrites previous information in the reserved space without warning. For example, suppose BOOTEX, which requires 768 blocks, is installed at offset 0 of the disk. If another BOOTEX is later installed at offset 1 (starting at block 256), INSTL overwrites the last two-thirds of the original BOOTEX without warning. This error will not be detected until the next time the system is booted. To prevent this problem, we recommend that you put only BOOTEX, and nothing else, in the reserved space.

lu is the LU number in the target system that you boot from and that has the system and snap files on it. BOOTEX mounts it automatically for you. If *boot dest* is a number, this is also used as the destination LU for the boot extension.

boot source is the name of an RTE-A boot extension file in the host system. INSTL uses the boot extension code in this file, plus information from the snap and system files, to create the boot extension code that is placed in the destination file. If you do not specify this parameter, INSTL assumes that the destination file is a valid boot extension file and uses that file as the source file (if the destination is a file, not an offset).

option N = No Console

or

E = Enable CS/80 timeout retry

If the target system is generated without LU 1 and this option field is "N" (No Console), no system messages will be displayed on the system console when booting the target system. Thus the target system becomes a consoleless system. If the option field is "E", this enables the CS/80 timeout retry on the first entry into the driver from BOOTEX.

Enter the parameters requested. The destination file may already exist on the disk; if so, it must be a type 1 file at least as long as the source file. If it does not exist, a file is created using the name you enter.

Description:

When you boot the RTE-A system, enter (either explicitly or by default) the physical address of BOOTEX and the name of a type 1 system file or a type 4 boot command file. The VCP/loader ROM loads BOOTEX from that address into main memory and starts executing the boot extension. The boot extension either loads the specified system file into main memory and starts executing or refers to the boot command file and follows the instructions for booting.

This assumes that the boot extension is properly located. The VCP/loader ROM does not check what it is loading. It goes to the address specified in the boot command, loads whatever is there into memory, and attempts to execute it. If you are booting from a disk, make sure BOOTEX is in the right place with the INSTL program and the CI (or FMGR) IN command. When you boot from a disk, the file that contains the boot extension must be on physical cylinder 0 and sector 0 of the boot LU. (This implies, of course, that the disk LU starts at physical cylinder 0; otherwise, the system cannot be booted from that LU.)

When you use INSTL to initialize a cartridge that will actually be used on a differently configured disk LU, for example, one set up for a target system other than the host system, INSTL gets the information it requires from the specified snapshot file and system file of the target system. The boot extension code comes from any existing RTE-A boot extension file of the same revision.

The way INSTL and the IN command are used varies according to the type of system you are using because there are slight differences between the operation of the IN command in RTE-A and RTE-6/VM systems.

The CI IN command allows free space to be reserved at the beginning of a disk LU. This space does not contain BOOTEX code. The FMGR IN command allows you to purge all files on a disk

LU and set (logical) track 0 as the starting track of the LU. When you do this in an RTE-A system, the FMGR IN command creates a file named BOOTEX (type 1, 768 blocks) at the start of the disk LU. Because this file does not contain boot extension code, you must use INSTL to put the boot extension code into the BOOTEX file. (When installing BOOTEX on a CI volume, INSTL can install it directly into the reserved space if the volume's LU number is the same as the LU number it will have at boot time. Otherwise, you can use INSTL to create a new initialized BOOTEX file which may later be put into the reserved space of any CI volume using the FPUT utility.)

In an RTE-6/VM system, you may use the FMGR IN command to purge all files and set the starting track at 0, but it does not create a boot extension file. Use INSTL to create the file and fill it with the boot extension code.

The following examples compare the use of CI and FMGR:

For CI:

```
INSTL, snap, system, 0, 16, BOOTEX    !LU 16 is a CI volume
                                      !and has the system
                                      !and snap files on it
```

or

```
INSTL, snap, system, BOOTX, 16, BOOTEX
FPUT BOOTX 16
```

For FMGR:

```
PU,BOOTEX:-32767:P1                              !P1 is an empty
INSTL, snap, system, BOOTEX::P1, 16, BOOTEX      !FMGR LU. LU 16
                                                 !has the system and
                                                 !snap files on it
```

INSTL Operation

INSTL is not concerned with the physical location of the boot file. It puts the boot extension code in the file you specify, whether or not that file begins at physical cylinder 0 and sector 0 of a disk LU.

If you use the snap and system parameter defaults, INSTL configures the destination file according to the specifications of the host system. If the source file is defaulted, INSTL assumes that the destination file is a valid boot extension file and uses the destination file as its source. If the source is defaulted and the destination file is an offset number, INSTL quits.

Successful completion is indicated by the message:

```
INSTL END.  aaaaaa IS YOUR BOOT EXTENSION FILE.
WARNING: BOOT FILE MUST
BE AT CYL 0, HEAD 0, SECT 0.
```

where *aaaaaa* indicates the actual file name.

Error Messages

BOOT SOURCE FILE NOT TYPE 1.

The source file must be type 1. The program terminates after the error is found.

DESTINATION FILE TOO SMALL

The destination file must be at least as large as the source. The program terminates after the error is found.

INSTL end. *aaaaaa* is your boot extension file.
warning: boot file must be at cyl 0, sect 0.

These two messages are given at the successful completion of the program when the destination is a file. The destination file must be at cylinder 0, sector 0 in order to be accessed by the ROM bootstrap code. Note that it must also start at head 0 if the disk is formatted in cylinder mode. The message is given whether or not this condition is found.

INSTL end. Your boot extension has been installed
at boot block *xx* on lu *yy*.

This message is given at the successful completion of the program when the destination offset is a boot block number.

LU *xxxx* IS NOT A DISK LU.

The destination file can be created only on a disk. The program terminates after the error is found.

Caution Base page links may be overwritten by driver parameters.

This warning is issued if INSTL suspects that locations 140B-154B of the source BOOTEX are base page links. INSTL writes the driver parameters of the target disk into these locations of the destination BOOTEX. Thus, any base page links in these locations are destroyed. This can only occur if a new BOOTEX is being built with a BTXL supplied by you that requires substantially more page links than the HP-supplied BTXL.

IO (Display I/O Configuration and Status)

Purpose: Displays the system I/O configuration and status.

Syntax: `IO [-`*flags*`] [`*args*`] [`*listfile*`]`

flags is a string of single alphabetic characters preceded by a dash; for example, −CX selects both the C and X flags. *flags* select the style of report IO displays. Valid flags are:

- `C` Show configuration, not status, information. If this flag is not present, device status information is displayed.
- `D` For the device status report, D inhibits looking up device descriptions in file /SYSTEM/DEVICES, using the generic device type descriptions instead.
- `L` Sort the select code configuration listing by LU within the select code.
- `S` Organize the configuration report by select code, not LU.
- `X` Display extended configuration information for each device.

args takes one of three forms, depending on the flags selected. If the S flag is not selected, it is of the form:

firstlu [*lastlu*]

or

symlink_file

where:

- *firstlu* is the first LU to be displayed.
- *lastlu* is the last LU to be displayed. If *lastlu* is not given, only *firstlu* is displayed. If neither *firstlu* nor *lastlu* is given, all LUs are shown.
- *symlink_file* is the name of a symbolic link file that points to an LU. Information for that LU will be displayed.

LUs may be given as an octal value suffixed with “B”.

If the S flag is selected, *args* is of the form:

- *sc* the select code of the interface card to display. This may be given as an octal value suffixed with “B”. If not given, all select codes are shown.

listfile is the name of the file to which the listing is to be sent; if none is specified, the listing is displayed at the scheduling terminal.

Three styles of reports can be selected from the above syntax. The relevant portions of the syntax for each report style are reproduced here.

1. To display device status:

```
IO [-D] [firstlu [lastlu]] [listfile]
```
or
```
IO [-D] symlink_file [listfile]
```

2. To display I/O configuration organized by LU:

```
IO -C[X] [firstlu [lastlu]] [listfile]
```
or
```
IO -C[X] symlink_file [listfile]
```

3. To display I/O configuration organized by select code:

```
IO -CS[LX] [sc] [listfile]
```

Description:

The device status report includes the LU number of the device and a device name corresponding to the device type specified at generation time, or the description from file /SYSTEM/DEVICES if present, or "not assigned" if the LU is not defined.

The status of the device is shown as follows:

idle — The device is not handling a request and is free to process one. If suffixed by the message

```
lu n in node list is busy
```

the device is "busy" because another device in its node list is busy with a request (node lists are defined at generation time for LUs that cannot be simultaneously accessed).

down — The device was downed. If it was downed by the operator while processing a request, the information about the request appears the same as for "busy" devices. If downed by RTE due to a device error, a brief description of the error as given by the extended status of the device appears.

busy — The device is processing a request. Information about the request is displayed in one of the following formats:

```
{buffered | unbuffered} io_type for program
```

for buffered or unbuffered program I/O, where *io_type* is one of the following:

read	for read requests
write	for write requests
control *n*b	for control requests (*n* = function code)
io	for multi-buffered requests

and *program* is the name of the program that issued the request.

```
class #n io_type by program
```

for class I/O, where *n* is the class number and *program* is the program that requested the class I/O, but not necessarily the program waiting on the class get.

```
system io_type
```

for system requests.

If the request is queued on the interface driver pending completion of another request, the message

```
queued:  interface is busy with lu n
```

appears, where *n* is the LU of the device occupying the interface.

If the system is trying to abort the request, the message

```
request is being aborted
```

appears. This usually means that the device is not responding to the abort request.

If a write request has exceeded the upper buffer limit of the device, the message

```
buffer limit exceeded
```

appears. Programs attempting to access the device are buffer limit suspended on the LU until the number of words of data waiting for the device goes below the lower buffer limit. The buffer limits can be examined with the −CX options.

If the device is locked to a program, the message

```
locked to program
```

appears, where *program* is the locking program. No other program may normally access an LU that is locked.

The I/O configuration report types are:

`CI> io -cs`	This report lists for each I/O interface in the system: – the I/O select code for this interface (or zero if none) – the type of interface (name and octal interface type) – each device connected to this interface, sorted by HP-IB address, if applicable, and then by LU number (but see the −L option) The device information includes HP-IB address, LU number, name, and device type.
`CI> io -cs 27b`	The same as above, but only for select code octal 27.
`CI> io -c 8 9`	This report shows for LUs 8 and 9 the device name, interface select code, HP-IB address, DVT number and address, and interface type.

If the −X option is used (for example, "`io -cx 100`"), extended configuration is listed for each device, including:

queuing	'Priority' indicates requests are queued in order of the priority of the requesting program; 'FIFO' indicates requests are queued in order of arrival.
timeout	The number of centiseconds that a device has to respond to a request before timing out, or 0 if there is no timeout for this request.
priority	The interface priority of requests for this device, meaningful only for interface drivers that queue requests by priority (not FIFO).

buffered/ unbuffered	Write requests to buffered devices are copied to SAM and processed without I/O-suspending the requesting program. If buffered, the buffer limits are displayed.
buffer limits	When the number of words of buffered data for the device goes above the upper buffer limit, any programs that attempt to access the device are buffer limit suspended on the LU until the number of data words buffered goes below the lower buffer limit.

More configuration information may be given for certain device types, including MAC/ICD and CS/80 disks.

The device status report uses localized device descriptions from file /SYSTEM/DEVICES if present, rather than the generic device type descriptions. The format of this file is:

lu devicename description
or
`*` *comment*

where:

lu	is the LU number of the device. The LU numbers must appear in increasing order in the file.
devicename	is a character string not containing spaces or commas that names the device (not currently used).
description	is any string up to 21 characters that describes this device.

For example:

```
8    tape1    7970 tape drive
```

Responses to the "`More ...`" prompt can be preceded by a number from 1 to 32767, called *n* below:

<space>	list another page, or *n* lines if *n* is given
<return>	list the rest of the information without pausing
A or Q	abort the listing
+	list 1 line, or skip *n* lines and list 1 line if *n* is given
P	set # lines per page to *n* and list a page
Z	suspend IO, restart with the RTE GO command

IO

Examples:

The first two examples below show both the user input and report response.

```
CI> io 1

 lu device name     status

  1 terminal        busy:  class #25 read by FST/1

 Thu, Apr 28, 1988, 12:03 PM
```

This displays the status of LU 1.

The following entry displays the status of LUs 100 through 104.

```
CI> io 100 104

 lu device name     status

100 terminal        up
101 terminal        busy:  class #39 read by CI/101
102 terminal        busy:  buffered write for IO/102
103 terminal        busy:  class #32 read by CI/103
104 terminal        busy:  class #36 read by PROMT locked to LOGON

Thu, Apr 28, 1988, 12:04 PM
```

The following examples show user input only:

`CI> io`	(Show the status of all LUs in the system)
`CI> io 24`	(Show the status of LU 24 only)
`CI> io 95 110`	(Show the status of LUs 95 to 110)
`CI> io /dev/dat`	(Show the status of the LU associated with the symbolic link /DEV/DAT)
`CI> io –cs`	(Show the configuration for all select codes)
`CI> io –c 42`	(Show the configuration of LU 42)
`CI> io –c 51 85`	(Show the configuration of LU 51 through 85)
`CI> io –cs 27b`	(Show the configuration of select code 27 (octal))

IS (Compare Strings or Numbers)

Purpose: Compares two character strings or numbers.

Syntax: `IS` *string1* `<`*rel operator*`>` *string2* `[`*option*`]`

string1 is a numeric or character string.

rel operator is a 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 is a numeric or character string.

option specifies special comparison instructions. Possible values are:

`-i` Integer comparison. A suffix of B following *string1* or *string2* in either upper or lowercase indicates an octal value. A leading minus 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. 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> is 1024 eq 2000B -i          (The two strings are compared as integers)

CI> 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.)
```

KTEST (ksh-style Condition Evaluation Command)

Purpose: Evaluates the expression *expr* and returns a true or false exit status.

Syntax: KTEST *expr*

Description:

KTEST evaluates the expression *expr* and, if its value is true, returns a zero (true) exit status; otherwise, a non-zero (false) exit status is returned. KTEST also returns a non-zero exit status if there are no arguments. The following primitives are used to construct *expr*:

```
−a file       : True if file exists.
−d file       : True if file exists and is a directory.
−x file       : True if file exists and is a type 6 file.
−f file       : True if file exists and is an ordinary file. (Not a directory, not a symbolic link
                and not a type 6.)
−h file       : True if file exists and is a symbolic link.
−r file       : True if file exists and user has read access.
−s file       : True if file exists and is not empty.
−w file       : True if file exists and user has write access.

f1 −nt f2     : True if file f1 is newer than file f2 (update time).
f1 −ot f2     : True if file f1 is older than file f2 (update time).
f1 −st f2     : True if file f1 has the same update time as file f2.
f1 −nc f2     : True if file f1 has a newer create time than file f2.
f1 −oc f2     : True if file f1 has a older create time than file f2.
f1 −sc f2     : True if file f1 has the same create time as file f2.
f1 −na f2     : True if file f1 has a newer access time than file f2.
f1 −oa f2     : True if file f1 has a older access time than file f2.
f1 −sa f2     : True if file f1 has the same access time as file f2.
f1 −ef f2     : True if file f1 and file f2 exist and refer to the same file.

−z string     : True if the length of string is zero.
−n string     : True if the length of string is non-zero.

string = pattern  : True if string matches pattern.
string != pattern : True if string does not match pattern.
string1 < string2 : True if string1 comes before string2, based on ASCII value of their
                    characters.
string1 > string2 : True if string1 comes after string2, based on ASCII value of their
                    characters.

n1 −eq n2     : True if the integers n1 and n2 are equal.
n1 −ne n2     : True if the integers n1 and n2 are not equal.
n1 −gt n2     : True if n1 is greater than n2.
n1 −ge n2     : True if n1 is greater or equal to n2.
n1 −lt n2     : True if n1 is less than n2.
n1 −le n2     : True if n1 is less than or equal to n2.
```

A compound expression can be constructed from these primitives by using any of the following, listed in decreasing order of precedence.

```
( expression )    True if expression is true. Used to group expressions.
! expression      True if expression is false.
```

expression1 && expression2	True, if expression1 and expression2 are both true.
expression1 || expression2	True, if either expression1 or expression2 is true.

Pattern Matching:

The pattern matching notation used by the "=" and "!=" primitives allow the following patterns.

Pattern	Match
a	Matches character "a"
−	Matches any character.
@	Matches any character zero or more times.
[ai−k]	Matches any of the characters "a", "i", "j", "k"
[!ai−k]	Matches any characters but "a", "i", "j", "k"

Examples:

The following command will compile a source file only if the file specified in $LOCK does not exist and either the source file or include file is newer than the relocatable.

```
if ktest ! ( -a $lock ) && ( prog.ftn -nt prog.rel || prog.ftni
   -nt prog.rel )
then
   ftn7x prog.ftn 0 -
fi
```

The following example prints a file if it begins with "a", "b", or "c", the file is neither a directory nor a type 6 file, and the user has read access to the file.

```
if ktest $file = [abc]@ && ! ( -d $file || -x $file ) && -r $file
then
   lp $file
fi
```

Caution

All operators and primitives must be delimited by spaces (or commas). KTEST cannot test patterns that contain spaces or commas.

KTEST cannot determine permission rights for remote files that are open exclusively to other programs. When this occurs, the return status of the KTEST −r and −w primitives are based upon the remote file's non-owner access permissions.

The −a, −f, −x, −r, −s, and −w primitives cannot be evaluated for a FMGR file without opening the file. These primitives return FALSE for a FMGR file if the file cannot be opened for any reason.

The −ef primitive returns FALSE when either argument is a FMGR file.

Non-existent files are considered older than existing files.

FMGR files do not have time stamps and are considered older than any FMP file. There is no distinction between a non-existent file and a FMGR file when comparing time stamps.

LI (List Files)

Purpose: Lists files in paged format to the terminal.

Syntax: `LI [-`*flags*`]` *filemask*

flags specify output format. The options are described below.

filemask is a file descriptor mask for the files to be listed.

Description:

After listing each screenfull ("page") of the file, a "`More...`" prompt is given, which lets you read that page before moving on to the next. Additionally, you may enter commands at the prompt to skip forward or review backward. Commands and options are described in the sections that follow.

The "`More...`" prompt by default also shows the percentage of lines in the file that were viewed thus far (that is, the fraction of the total number of lines in the file that the current line represents) if this is known. For FMGR and device files this is not known until the end of the file is reached. The "*%*" command (moves to a percentage through the file) is still legal for these files, but LI first reads to the EOF so that the percentage can be found.

LI is loaded as a VMA program by default. You may also load the LI program as an EMA program, or as a non-EMA/VMA program, by following the instructions given in the LI.LOD file. To load LI serially reusable as a memory-resident program in a memory-based system, you must load LI as an EMA, or as a non-EMA/VMA program. If LI is linked as a non-EMA/VMA program, certain features of LI are not available; these features are noted as "requires VMA" in the following LI command descriptions.

LI Flags

Flags are strings of characters preceded by a dash (−) and can be entered together, as

```
-nhx
```

or separately, as

```
-n -h -x
```

When a flag uses the "next parameter" as an argument, the next unconsumed parameter is consumed as the argument; thus

```
-s 10 -e 15
```

and

```
-se 10 15
```

both set the starting and ending lines to 10 and 15, respectively.

NO TAG summarizes the flags.

LI Flags Summary

Flag	Description
A	Lists ASCII text (defaults for file types 0, 3, and 4).
W	Lists octal words (defaults for all other file types).
O	Lists octal bytes.
I	Lists signed integer words.
B	Lists binary words.
H	Lists hexadecimal bytes.
D	Lists ASCII text with display functions around special characters.
N	Lists line/record numbers.
S *ln*	Sets starting line to list at *ln*.
E *ln*	Sets ending line to list at *ln*.
X	Suppresses prompting at the end of the file; quits listing the file when the EOF is reached.
$	Always prompt at the end of file, even if the file is less than one page long.
M	Folds long lines. Lines longer than 79 characters are treated as multiple lines for pagination.
T	Truncates trailing blanks on text listings.
F	Forces type 1 access, lists blocks in octal words by default.
C	File has FORTRAN-style carriage control characters in column 1; LI by default sets the honesty bit so printer drivers do not treat column 1 as carriage control.
Q	Quiets file access; does not record access time.
L *fl*	Diverts listings to file *fl*. *fl* may be preceded by a tilde (~) to overlay an existing file, or a plus (+) to append to an existing file.
Y	Lists each file that matches *filemask* without asking.
R *rsz*	Sets maximum record size, if more than 512 characters are wanted.
pgsz	Sets number of lines per page to *pgsz* (1..32767). If *pgsz* is zero, does not paginate (lists without prompting).
> */cmds/*	Executes initial command string *cmds* at the start of each file. For example, −> /'start'1+s'end'1−e1.l/ sets the bounds of the file to the lines between the strings "start" and "end" and starts listing from the top bound. The delimiter surrounding commands may be any character except a space or comma. Backquotes (‘‘) should also surround the string to keep CI from inserting commas, and so on.
P */str/*	Redefines the 'More...' prompt. *str* is a string of characters delimited by any character except a space or comma. Backquotes (‘‘) should also surround the string to keep CI from inserting commas, and so on. Within *str*, the following string substitutions occur: %f file name %l current line numbers %p percentage through the file %w window or page number %% percent

LI

LI Commands

Commands entered at the "`More...`" or "`End...`" prompts can be preceded by a number from 1..2147483647 (this value is referred to as *n* in the LI command summary below). The listing commands are summarized in NO TAG.

Trailing arguments like *m* and *rex* are prompted for interactively. In the "`>`" runstring flag, they are entered directly after the command, as in

```
CI> li -> /f!zow!ka1.ua/ yow
```

which finds string "zow" (delimited by exclamation points (!)) in file "yow", marks that line with "a", goes to line 1, and lists yow until the line marked with "a" is encountered.

When more than one file is selected, LI prompts with

```
File:  file, list? [Y]
```

before listing each file (the −Y flag turns this prompt off). When prompted, you may enter one of the responses shown in NO TAG.

Regular expressions are the same as those in EDIT/1000; see the documentation for EDIT/1000 for use and examples. In brief, the expressions are shown in NO TAG.

Backward movement is allowed for device (as opposed to disk) files, but LI cannot back up beyond the number of lines that fit inside an internal buffer.

LI Commands Summary

Command	Description
Space or L	Lists the next page or the next *n* lines if given.
Return	Lists the rest of the file or goes to line *n*.
A or Q	Aborts list. 'A' moves to the next masked file. 'Q' quits the entire listing.
+	Lists the next line or skips forward *n* lines.
−	Skips backward 1 line or *n* lines from the top line in the window.
B	Skips backward 1 page or *n* pages from the top line in the window.
G or .	Goes to line *n*. 'G' lists a page. A period (.) lists 1 line.
$	Lists the last window.
%	Goes to a line that is *n* percent through the file.
'*rex*'	Searches for the next occurrence of regular expression *rex* from the current window or from line *n*. A null string searches for the last string entered. The trailing quote is not used in interactive mode; simply terminate the pattern with <return>.
F/*rex*/	Same as '*rex*' but with user-defined delimiters for startup commands. In interactive mode, the delimiters are not used; the pattern is terminated with <return>. The delimiters may be any character except space or comma. The delimiters cannot be the same as those used to surround the startup commands string.

NO TAG. LI Commands Summary (continued)

Command	Description
‘*rex*‘	Searches backward for regular expression *rex* from the current window or from line *n*. A null string searches for the last string entered. The trailing quote is not used in interactive mode; simply terminate the pattern with <return>.
@/rex/	Show all lines containing pattern from the current window or from line *n*. The delimiters may be any character except space or comma.
K*m*	Marks top window line or line *n* with *m* which must be an alphabetic character from A to Z.
:*m*	Goes to line marked with character *m* and lists *n* lines.
U*m*	Lists until the line marked with *m* is encountered; lists no more than *n* lines.
P	Sets page size to *n*, if given, and lists a page.
Oc	Toggles or resets the setting of runstring flags, including listing modes (a, b, h, i, o, w).
S	Sets the starting file line to the window top. LI will not back up farther than this line.
E	Sets the ending file line to the window bottom. LI will not advance past this line.
#*f*	Moves to file number *n*, if given, or prompts for file number *f* (requires VMA).
#+[*i*]	Moves forward *i* selected files, or 1 file, if not given (requires VMA).
#−[*i*]	Moves backward *i* selected files, or 1 file, if not given (requires VMA).
#?[*f*]	Shows a window of selected files starting at file *f*, or around the current file, if not given (requires VMA).
=	Displays the file name and current line number on the screen.
N*file*	Adds a new file name to the list of files to be displayed (requires VMA) and moves to this new file.
R	Removes (purges) the file being listed.
? or H	Displays help information.
Z	Suspends LI operation. You can restart with the system GO command.

LI Responses

Response	Meaning
Y or <space> or <return>	List the named file.
N	Do not list the file.
S	List the file and set the “y” option (to suppress the prompt).
A	Do not list the file; abort the listing.
#	Show selected files or move to another file (see description of the # command in NO TAG).
R	Remove (purge) this file.

Expressions Summary

Expression	Meaning
.	Matches any character.
@	Matches any character zero or more times (same as ‘.*’).
^x	Anchors the pattern to the beginning of the line.
x$	Anchors the pattern to the end of the line.
[ai−k]	Matches any of the characters ‘a’, ‘i’, ‘j’, and ‘k’.
[^ai−k]	Matches any character except ‘a’, ‘i’, ‘j’, and ‘k’.
x*	Matches zero or more occurrences of pattern x.
x+	Matches one or more occurrences of pattern x.
x<5>	Matches 5 repetitions of pattern x.
a:b	Matches a word boundary between patterns a and b.
*	Matches the character ‘*’.

LNS (Create Symbolic Link; VC+ Only)

Purpose: Creates a symbolic link to a file or a directory.

Syntax:

```
LNS [-FIQV] file symlink_file
LNS [-FIQV] file1|mask1 [ file2|mask2 ... ] dest_directory/
LNS [-FIQV] file1|mask1 [ file2|mask2 ... ] dest_mask
LNS [-FIQV] directory symlink_directory
```

`-F`	(forced) overwrite existing files when creating symbolic links.
`-I`	(interactive) issue a prompt requesting confirmation for each symbolic link that would overwrite an existing file. This option only has an effect when used with the −F option.
`-Q`	quiet mode; inhibit error reporting.
`-V`	verbose mode.
`--`	specifies the end of options; required if *mask* begins with a hyphen (−).
file *mask* *directory*	file names, directory names, or masks that are to be used as the contents of the symbolic link files to be created.
symlink_file	is the name of the symbolic link file that is to be created.
dest_directory/	if specified, LNS creates symbolic links in *dest_directory* to all of the files specified in the "*file1*|*mask1*, ..." arguments.
dest_mask	if specified, LNS generates the corresponding destination file descriptor to create the symbolic link for each of the files specified in the "*file1*|*mask1*, ..." arguments.

Description:

LNS creates symbolic links between files, between directories, or between files and devices. You may specify any FMP file or any logical unit that is to be used as the contents of the symbolic link file. For symbolic links to FMP files, *file* must be in hierarchical format (that is, */dir/file*).

Once created, the symbolic link file contains a file descriptor that points to the file given as the source argument. The symbolic link can be created in any existing directory. The symbolic link file may not be a FMGR file.

If a mask is specified as the first argument after the options, the destination must be a directory or a destination mask.

Remote files may be specified with the use of the DS transparency syntax.

If the name of a symbolic link file being created conflicts with the name of an existing file, the −F option must be specified to overwrite the existing file.

If the symbolic link being created has the name of an existing file (or symbolic link) and you do not have write access to the file, LNS prompts for confirmation to purge the file. When prompting for confirmation, LNS displays the file name and its protections and requests for confirmation of the removal of the file. If the response begins with "y", the existing file is purged and the symbolic link is created.

When specifying existing directories in the argument list, a directory may optionally be specified with a trailing slash (*dirname/*) or with the type extension (*dirname*`.dir`). The trailing slash or the type extension is necessary when a file name without a type extension exists that has the same name as the directory.

When creating symbolic links to directories, the type extension .DIR must exist in both the name of the symbolic link and also in the name of the directory referenced in the symbolic link.

Symbolic links to FMGR directories, FMGR files, and FMGR type 0 files are not supported.

Return values:

$RETURN1	The number of symbolic links that could not be created.
$RETURN2	The number of symbolic links that were successfully created.

Examples:

```
CI> lns subdir/name fname
```
(Create a symbolic link to SUBDIR/NAME in the file FNAME in the current working directory; listing the file FNAME will now list the file SUBDIR/NAME)

```
CI> lns /progs17/@:::6 /progs18/@:::6 /programs/
```
(Create symbolic links to all type 6 files in /PROGS17 and /PROGS18 in the /PROGRAMS directory)

```
CI> lns /system.dir sys.dir
```
(Create a symbolic link to the /SYSTEM directory that will look like a subdirectory SYS in the current working directory; a directory listing of the SYS/ subdirectory will display the contents of the /SYSTEM/ directory)

```
CI> lns -if 8 /devices/magtape
```
(Create file MAGTAPE in the devices directory that points to LU 8; if the file already exists, prompt the user for confirmation to replace the existing file)

```
CI> lns -- -prog::dir1 /dir/
```
(Create links in the /DIR directory to all files in the /DIR1 directory that match the mask "`-prog`")

```
CI> lns /mydir/tests/@>bucko ./
```
(Create links in the current directory to all files in the /MYDIR/TESTS.DIR directory on the remote system bucko)

LS, LL, LSF, LSX (List Directory Contents)

Purpose: Lists the contents of a directory. These commands support redirection.

Syntax:
```
LS [-CDGLNOPRTUXYZcfl1 ] mask1 [ mask2 ... ]
LL [-CDGLNOPRTUXYZcfl1 ] mask1 [ mask2 ... ]
LSF [-CDGLNOPRTUXYZcfl1 ] mask1 [ mask2 ... ]
LSX [-CDGLNOPRTUXYZcfl1 ] mask1 [ mask2 ... ]
```

`-C` use the create time of the file for sorting (−T) or listing.

`-D` if an argument is a directory, list only its name (not its contents); often used with −L to get the status of a directory.

`-G` same as −L, except that only the group is listed (owner is omitted). If both −L and −G are specified, the owner is NOT listed.

`-L` list in long format, which includes file attributes, owner, group, size in bytes, and the update time for each file (see File Attributes below). Note that the field used by the HP-UX `ls` command to specify the number of links to a file is always set to 1 on RTE-A. If the time of the last update is set to a date that is more than six months prior to the current date, or any time in the future, the year is substituted for the hour and minute of the update time. For remote files, the owner's UID and the group's GID numbers are displayed.

`-N` use the owner's UID and group's GID numbers rather than the associated character strings.

`-O` same as −L, except that only the owner is listed (group is omitted). If both −L and −O are specified, the group is not listed.

`-P` if a file to be listed is a directory, include a slash (/) after the file name.

`-R` reverse the order of the sort. This produces a listing in descending order or oldest first (−T).

`-T` sort according to the time modified (latest first); then sort alphabetically.

`-U` use the access time instead of the update time when using the −T or −L option.

`-X` output the listing in multiple column format; sort across instead of down the page.

`-Y` display the file type in the first character of the file attributes. Only RTE-A file types 1 through 9 are reported.

`-Z` display the file name as a namr. Ignored when used with multiple column output.

`-1` list file names in single column format regardless of the output device. This option overrides `-\c`.

`-\c` list output in multiple column format; sort down columns. The `-1` option takes precedence over −c.

`-\f` the following characters are included after each file name for the given types of files:

`/` directory

`*` RTE type 6 file or CI command file

`@` symbolic link file

`-\l` if the argument is a symbolic link, list directory attributes of the file or directory to which the link points rather than the link itself.

`--` marks the end of the options. This is only required when the mask begins with a hyphen (−).

mask1 ... one or more file masks for which to display file or directory information.

Description:

LS reports the name of any file matching the mask and any other information that is requested. If no mask is supplied, the contents of the current working directory is listed. By default, the listing is sorted in ascending order.

The output of LS can be redirected to an output file by specifying either "`>`filename" or "`>>`filename" in the runstring. The output file specified must be delimited by commas and is position independent. If the file already exists, it will be overwritten. To append to a file, "`>>`filename" can be used. If the file does not already exist, it will be created.

If an output file is not specified, LS breaks the output into screen pages. Paging is disabled when an output file is specified. Multiple redirection strings may occur in the runstring; however, only the last redirection is executed.

Any mask starting with the "`>`" character must be specified with either a relative or absolute path, for example, "`ls ./>@`" or "`ls /dir/>@`".

The output format is dependent on whether the output is defaulted to the terminal LU, and is also controlled by the options. The default format for the terminal LU is to display the directory listing in multiple columns, with entries sorted vertically by column. If an output file is specified, one entry per line is listed. To list multiple columns to a specified output use the −\c or −X option. LS uses the $COLUMNS environment variable to determine the length of the output line. If this variable is not set, a default of 80 columns is used.

Note that the options to LS are case sensitive. To specify a lowercase option, you must quote the option character to prevent CI from upshifting the character. For example, to specify the l option, use the following command string:

```
CI> ls -\l mask
```

Because the default behavior of CI is to case fold the entire command line, the options of this version of LS are the inverted case of the equivalent options for the HP-UX `ls` command. Therefore, the most commonly used options do not require quoting.

Shorthand formats

LS can be invoked with different names to achieve commonly used formats.

`lsf`	is equivalent to "`ls -\f`"
`lsx`	is equivalent to "`ls -X`"
`ll`	is equivalent to "`ls -LY`"

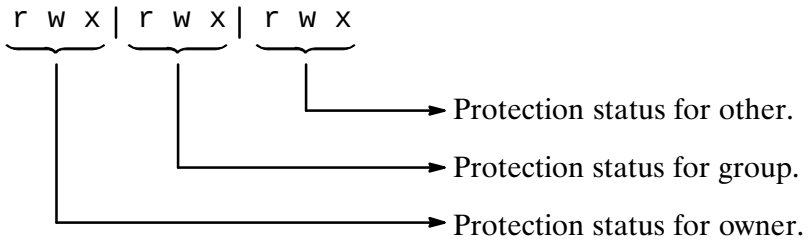
The shorthand versions can be achieved with the use of proto-ID segments or by creating symbolic links to LS.RUN. For example, to create a proto-ID name of LSF from LS.RUN, use the CI command "`rp ls lsf d`". To do this using symbolic links, use the LNS program as follows: "`lns ls.run /programs/lsf.run`". Note that shorthand formats cannot be implemented using the CI ALIAS command because LS.RUN must be invoked with the shorthand name.

File Attributes (−L option)

The file attributes displayed in listings produced by the long format consists of 10 characters. The first character indicates the entry type:

`d`	Directory
`l`	Symbolic link
`-`	Ordinary file
<*n*>	RTE file type (−Y option only)

The remainder of the characters identify protection status for owner, group, and others as follows:



Protection status letters are defined as follows:

`-`	Permission denied
`r`	Read permission
`w`	Write permission
`x`	File is either an RTE type 6 file or a CI command file (text file with a .CMD type extension).

Return Value:

LS will return 0 in $RETURN1 if all files were listed successfully; or greater than 0 if LS was aborted for any reason. $RETURN2 will contain the number of files found matching the mask.

Examples:

Print a long listing of all the files in the /PROGRAMS directory. List the most recently modified (youngest) file first, followed by the next older file, and so forth. For symbolic links, use the update time of the file pointed to by the link. Append the output of the LS program to the file OUTFILE in the current working directory.

```
CI> ls -l\lt >>outfile /programs/
```

Note that CI will upshift the entire command line. To protect the lowercase l from being upshifted, the character must be preceded by a backslash (\).

MC (Mount Disk Volume)

Purpose: Mounts a disk volume and makes its contents available to the system.

Syntax: `MC` *lu*

lu is the LU number of the disk volume to be mounted. Must be a positive number.

Description:

The MC command mounts a disk volume to the file system, thus making it accessible to users of the file system.

If the disk volume has a valid FMP or FMGR directory, the volume is mounted; otherwise, 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 disk volumes are mounted, unless there are duplicate global directory names on two or more volumes. If a global directory on the newly mounted disk volume has the same name as a previously mounted global directory, the new directory is inaccessible. To access the new directory you must rename the previously mounted directory, then dismount the new disk volume and remount it. Disks can be mounted from a BOOTEX command file; this is the recommended procedure for disks containing system files and other commonly used programs.

This command can be used to mount FMGR cartridges. It recognizes FMGR directories without any user specification. The cartridge mounted will be placed at the bottom of the cartridge list.

MERGE (Concatenate Many Files into One)

Purpose: Collects input files and sends their data to a single composite output file. This allows you to form a library of binary relocatable files to be searched by the linker and to concatenate a program and all user-written subroutines.

Syntax: `MERGE [-`*options*`] [`*fromfile* *fromfile*`...] [`*destfile*`]`

options is one or more of the following, entered anywhere in the runstring, preceded by a dash (−):

- `-L` suppress the listing of file names being merged.
- `-D` remove Debug records from a relocatable file as it is merged.
- `-O` overwrite existing files without asking for user confirmation (if MERGE is linked to ASK by default).
- `-V` verify that an existing output file is to be overwritten.
- `-Z` suppress zero-length records normally output between files.

fromfile is one of the following:

- A command file containing a list of names of files to be merged. To be a command file, *fromfile* must have an .MRG or .MERG type extension; alternatively, it can be an FMGR file starting with "*" (for example, *MERGM), or a device LU. Masks may be used in a command file to indicate the files to be merged. If only one *fromfile* is entered and it is of type 0, 3, or 4, it is assumed to be a command file for backward compatibility.
- A file to be merged into *destfile*. To force the merging of a file that would otherwise be a command file, precede the file name with a backslash (\) (as in "\MergeThis.merg"). If you do this from CI, note that you must precede the file name with two backslashes, rather than one, because CI uses the backslash as a quoting character.
- A mask of files to be merged. For example, the mask "@.ftn" merges all the files with the extension "ftn" on the working directory. A mask may also be preceded by a backslash, which allows you to enter masks that begin with a dash, as in "\−q@," which specifies all files with a "q" in the second character. Note that two backslashes must be entered from CI, because CI uses the backslash as a quoting character.

You may specify multiple *fromfile*s in a runstring, each of which is interpreted as above.

If a *fromfile* is an interactive LU or you do not specify one, MERGE prompts for the input data files as follows:

```
Merge file:
```

Enter a file name or mask to be merged. MERGE continues prompting until you enter /E, press RETURN, or press control-D at the prompt.

destfile is the file or LU that receives the merged files. It may safely be the same as the first input file. If you name a file that does not exist, it is created. An existing file may be overwritten without notification unless you use the -V option (described below). The *destfile* is the last file name in the runstring.

Description:

If a command file is specified in the runstring (for example, "these.mrg"), the names of the files to be concatenated are read from it until the program encounters an EOF.

A zero-length record is written to the output after each input file, unless you selected the -Z option. This creates a subfile structure that is useful to some utilities and the FMGR ST command.

MERGE always removes index records (created by LINDX) from a file that it merges because the indexes are no longer valid for the new, merged file. To create a new index, run LINDX on the file after MERGE completes its operation.

Debug records (created by a compiler such as FTN7X or MACRO) may also be removed with the -D option. If you are not running the Debug/1000 program against the merged file, removing Debug records creates a smaller destination file.

It is legal (and useful) to make the output file identical to the first input file. For example, suppose that program source TRYIT.FTN has relocatable object code TRYIT.REL and requires the subroutines in TRYLIB.REL.

```
CI> merge tryit.rel trylib.rel tryit.rel
```

TRYIT.REL now contains the main program and the subroutines.

MERGE recognizes a BReak command prior to reading the file name to be added to the output file when the command device is a file or non-interactive LU. The BR command is entered as follows:

```
CI> br,merge
```

Examples

`CI> merge`	(Ask for input and destination files from the terminal)
`CI> merge –dl util.mrg util.lib`	(Create the library UTIL.LIB, using the list of files in UTIL.MRG. Remove Debug records, and do not list the files merged.)
`CI> merge @.mac.e big.mac`	(Merge all the macro sources everywhere into file BIG.MAC)
`CI> merge –lv a b.mrg c megafile`	(Merge file A, the files listed in B.MRG, and file C into file MEGAFILE. Do not list the files merged, and verify that an existing MEGAFILE may be overwritten.)

METER (Display CPU Usage)

Purpose: displays information about system and user processes.

Note Session accounting must be turned on in the boot command file in order for METER to work.

Syntax: `METER [LU] [`*commands*`]`

commands is one of the following:

`?`	display Help screen
`>`	set marked session
`+`	speed up refresh
`–`	slow down refresh
(*space*)	exit
`A`	display All processes
`D`	sort in Decreasing order
`E`	exit
`H`	sort by Histogram value
`I`	sort in Increasing order
`L`	sort by LU (session number)
`M`	display only Marked processes
`N`	sort by Name
`O`	sort by Owner
`P`	sort by Priority
`S`	sort by State
`T`	sort by Total CPU
`U`	display only User processes (system utility bit not set). METER determines whether a process is a user or system process by checking the system utility bit, which is set when a program is loaded with the LINK command SU.

You may enter a command when the cursor is at the bottom of the list on your screen. (If the cursor is anywhere else, command entry is ignored.) Note that you should avoid typing whenever the screen is being refreshed.

Commands may be entered in upper or lowercase except for the H, L, P, and S commands. These commands have both primary and secondary sort values. The primary sort values are indicated by uppercase characters, and the secondary sort values by lowercase characters.

For example, if you want to display programs first in increasing order of priority and then by the value of their histograms, you should enter these commands (not necessarily in this order):

1. Enter a lowercase “h” to specify the histogram value as the secondary sort value.
2. Enter an uppercase “P” to select priority as the primary sort value.
3. Enter lowercase “i” or uppercase “I” to specify that the sort is in increasing order.

If a process is in the marked session, a ">" appears in the left column of the listing. You may use the > command to change the marked session. The session running METER is the default marked session.

The + and - commands control the frequency with which METER refreshes its display. The default is a refresh every four seconds. You may vary the length of time between refreshes from one to 64 seconds. When you press +, the speed of the refresh is twice as fast; when you press -, the refresh is twice as slow.

METER repeatedly updates information until you stop the program by pressing the space bar, entering uppercase "E" or lowercase "e," or by entering the BR command at the break mode prompt.

The amount of CPU time that METER reports for a program is actually collected by the accounting system. The accounting system attributes the CPU usage of a system utility program to the parent program waiting for it, if any. For example, if program AGIT is making extensive use of FMP calls that cause D.RTR to be scheduled, METER shows that D.RTR is not using any CPU time, but AGIT is. This allows the true cause of performance problems to be more readily identified.

Sorting and Displaying Process Information

When you run METER, your screen displays a list of the top eighteen processes after sorting, with the following information about each process:

- process name
- LU or session number associated with the process
- logon name of the owner session
- current priority
- current state of the process
- total CPU seconds that have been used
- percent of the CPU that the process used since the last update
- a histogram of the percentage of CPU used

METER recognizes several commands that let you specify how you want the processes sorted and displayed. If you do not specify how you want the processes sorted, they are sorted in decreasing order by the percentage of CPU time used. If two percentages are the same, the current state of the processes are used to determine the order.

Example of METER Output

```
    METER   Use E or space to exit, ? for help.  Tue Jul 22, 1986  5:41:13 pm
       User programs sorted in decreasing order at a 4 second refresh rate.
    Primary sort key is the histogram.   Secondary sort key is the state.


 Name      LU  Owner    Priority    Current State          CPU        Histogram
                                                       Total  %  1 2 3 4 5 6 7 8 9
  LINK    106  DAVE          90        scheduled        5     9 ▌
 >METER     1  SCOTT         39        scheduled        4     2 ▌
  CI      106  DAVE          51     prog wait ss       15     0
 >CI        1  SCOTT         51     prog wait ss       15     0
```

Loading METER

To load METER, use the load sequence:

```
CI> link,#meter
```

MI2AB (Memory Image to Absolute Binary)

Purpose: Converts a type 1 file to a type 7 file.

Syntax: `MI2AB [`*input_file output_file*`]`

input_file is the file name of a type 1 file containing the system.

output_file is the file name of a type 7 file to be created or overlaid. It may also be a device LU.

If the output file does not already exist, MI2AB creates it (default size is 256 blocks); if the file exists, it is overlaid.

If no parameters are specified, MI2AB prompts you as follows:

```
This program copies a memory image (type 1)
file to an absolute binary (type 7) file.

Please enter the input file and output filenames.
Format: input::directory, output::directory
```

Description:

MI2AB allows you to load an RTE system into memory from a mini-cartridge, since the mini-cartridge bootstrap loader requires absolute binary format. (The system created by the generator is placed in a type 1 file.)

The system size for RTE-A is obtained from words 1 and 2 of the system file: word 1 contains the number of complete 32K blocks in the system files and word 2 contains the remaining number of words after the 32K blocks.

Break Detection

The Break command is entered as follows:

```
CI> br,mi2ab
```

When you enter BR, the following message is displayed:

```
Break Flag Set
```

When it detects a break, MI2AB closes the input and output files and terminates. Note that the output file is incomplete if the break flag is set during a conversion.

Error Messages

If the input file is larger than 32K blocks, MI2AB prints the following warning message:

Input is greater than 32K; Converting only 32K.

The output file is created, if necessary, and 32K blocks of the input file are converted.

MI2AB can generate the following error messages:

Break Flag Set

The break flag is tested in each major loop within the program. When the break flag is detected, the program closes the input and output files and terminates.

Input File Error *xxxxxx*
Output File Error *xxxxxx*

Any error that results from calling a file management routine in either the input file or output file produces this message. The FMP error code is reported and the files are closed.

MO (Move Files)

Purpose: Moves files from one directory to another on a given disk volume, and renames files.

Syntax: `MO` *file1 file2*

file1 is the source file descriptor. Refer to the CR command syntax description for the definition of file descriptor. May be masked to move a group of files. Refer to the "File Masks" section in Chapter 3 for the mask syntax.

file2 is 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 file name 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:

- The files are MOVED, not COPIED. Therefore, after you use the MO command, the file will no longer exist in its old location.
- The file contents are not moved, only the directory entry is moved. This is much faster, particularly for large files, and more reliable because the data is not altered.
- Files cannot be moved across disk 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.

For VC+ only, when the variable $QUIET_CMD is set to ON, the message

```
Moving FILE1 to FILE2 ... [ok]
```

is not displayed.

Examples:

The following example moves all the files that were not accessed since June 1983 into the archive subdirectory of the backup directory.

```
CI> mo @.@.a-8306 /backup/archive/@.@
```

The following example causes the subdirectory to become a global directory. A file that 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.

```
CI> mo /myglobal/mysubdirectory.dir /mynewglobal
```

MPACK (File Compacting and Disk Pack)

Purpose: Compacts files, packs a disk, and reports file block and extent usage.

Syntax: `MPACK [`*options*`] `*mask*`|`*lu*` [`*options*`]`

mask is any legal FMP mask form (including a standard file descriptor). However, if you use the Pack (P) or Visual Mode (V) option, you must specify an LU number.

options one or more of the options described in the following section.

Description:

When MPACK compacts files, it removes unused blocks and extents from the files. When it packs a volume, all of the disk's free space becomes one large block. This provides you with faster file data access and more efficient disk space usage. MPACK reports on "wasted" blocks (blocks that were allocated to a file but remain unused) and extent usage.

MPACK only works with hierarchical files and hierarchical file volumes.

MPACK Options

MPACK provides compacting, packing, and logging options. All options start with "+" and may be entered anywhere in the runstring, but only once. The options are summarized in NO TAG and discussed in the sections that follow.

Note that when you use MPACK, paging output to the screen occurs, but only if you are not doing any file or disk manipulation and you have not selected logging. In other words, you are not prompted every page of output to continue the listing if logging, file compacting, or disk packing is selected.

Compacting and Reporting Options

The +R, +T, and +C options specify how a file is to be compacted. These options allow you to remove unused ("wasted") blocks from files and extents. The +D option causes directories to be compacted. The +B option can be used with the +R, +T, or +C option to adjust file sizes to be a multiple of the blocks per bit value of the disk volume.

The +En, +Wn, and +Wn% options use the block and extent information to qualify the selection of files for reporting or compacting, and are therefore referred to as "qualifiers."

You must enter a mask in the runstring to let MPACK know which files to select. If a file that matches a specified mask also fits the qualifiers, reporting or compacting is performed. If you do not enter any compacting or packing options in the runstring, only reporting occurs.

MPACK Options Summary

Options		Description
Compacting and Reporting Options		
+R		Remove extents from files
+T	[*n*[%]]	Truncate wasted blocks from files
+C	[*n*[%]]	Remove extents and truncate wasted blocks
+B		Adjust file size to a multiple of blocks per bit value
+E	*n*	Select files with *n* or more extents
+W	*n*	Select files with *n* or more wasted blocks
+W	*n*%	Select files with *n*% or more wasted blocks
+A		AND the +E and +W qualifiers
+D		Include directories in the compacting
+Q		Quiet mode; only report totals
Packing Options		
+P		Pack the disk LU
+OK		OK to overlay data during pack
+V		Visual mode; display progress of pack
Logging Option		
+L	*file*	Log output to a file

Remove Extents from Files (+R)

The +R option allows you to recreate files that currently have extents, so that the resulting files have no extents. Type 1 and type 2 files cannot have their extents compacted, since extent sizes may contain some significance for those types. The only exception to this rule is for directories; MPACK removes extents from directories whenever possible. A directory's main size never exceeds 64 blocks, so extents may still remain after being compacted. The +D option must be specified for MPACK to remove extents from the directories.

The format for using +R is:

```
mpack mask [options] +r
```

Truncate Unused Blocks from Files (+T)

The +T option lets you truncate types 3, 4, 5, and 7 files with wasted blocks up to the EOF. When you enter +T, all the unused blocks are truncated from the file. If you enter +T*n*, *n* number of unused blocks remain in the file after truncation. If you enter +T*n*%, *n* percent of unused blocks remain in the file after truncation.

Since truncating never adds blocks to a file, if the number of unused blocks is smaller than the amount specified to remain, the file is unchanged.

Since truncating never shortens an extent, if truncation occurs within an extent, the entire extent is left unchanged and wasted blocks still exist.

Note that if you specify both +T and +R, +R is performed first so complete truncation can occur.

The format for using +T is:

```
mpack mask [options] +t[n[%]]
```

Compact Files (+C)

The +C option behaves like the +R and +T options combined. When you specify +C, you may not use either +R or +T in the same runstring. The +C*n* and +C*n*% usage is the same as that for the +T option.

The format for using +C is:

```
mpack mask [options] +c[n[%]]
```

Adjust File Size to Blocks per Bit Value (+B)

The +B option can be used with the +R, +T, or +C options. When this option is specified, file sizes are adjusted to sizes that are multiples of the disk volume's blocks per bit value. This eliminates wasted blocks in the bit map leaving them allocated to the file as unused blocks. Also, any subsequent extent will also be a multiple of the blocks per bit. The +B option only effects files that are having extents removed or files that are being truncated.

The format for using +B is:

```
mpack mask [options] +b
```

See the *RTE-A System Design Manual* for more information about disk management.

Extent Count Qualifier (+E)

The +E option lets you select any file whose extent count is equal to or greater than *n*.

The format for using +E is:

```
mpack mask [options] +En
```

Wasted Block Count Qualifier (+W)

The +W option lets you select any file whose unused block count is equal to or greater than *n*.

The format for using +W is:

```
mpack mask [options] +Wn
```

Wasted Block Percent Qualifier (+W%)

The +W% option lets you select any file whose unused block percent is equal to or greater than *n*.

The format for using +W% is:

```
mpack mask [options] +Wn%
```

ANDing Option (+A)

By default, the +E, +W, and +W% qualifiers have an ORing functionality when entered together in a runstring. In other words, a file is selected if it matches any one of the qualifiers. The +A option causes an ANDing functionality. In this case, files must match all of the qualifiers in order to be selected.

The format for using +A is:

```
mpack mask [options] +a
```

Include Directories (+D)

By default, when MPACK is reporting file block and extent usage, MPACK does not report information on directory files. Also, when MPACK is compacting files (+C, +T, +R), MPACK does not alter any directory files. If the +D option is included, MPACK treats the directory files in the same manner as all other files.

The format to report file block and extent usage for files and directories is:

```
mpack +D mask
```

The format to compact both files and directories is:

```
mpack +C +D mask
```

Quiet (+Q)

When a file is selected, MPACK produces an informational listing for each file and a total of the results. The +Q option suppresses the individual file information and displays only the totals. This is useful when only the final results are of interest.

The format for using +Q is:

```
mpack mask [options] +q
```

Packing Options

Disks almost always develop multiple unused data areas scattered throughout the disk. When this happens, you may find that although the total amount of free space on the disk would be sufficient for your needs, there is no single area large enough for the task you are doing.

When you use the +P option, MPACK "pushes" all the files on a disk together to replace scattered areas of free space with one large, contiguous data area. The only exception is on the last track, where the volume header must remain, so the space following the volume header remains separate. The +P option is described below.

Except for the volume header, MPACK can and may move everything on the disk to a new location. In order to do this, the LU cannot be connected to any paths, working directories, RP'd files, or anything else. Since MPACK must have the disk all to itself, the disk LU being packed will be dismounted and locked by the program. This means that during the pack operation, the disk is unavailable to all other users. When MPACK finishes packing the disk, the program remounts the LU.

Because MPACK is moving around file data and adjusting pointers on the disk, it should never be OF'd while in process. If you need to abort the program, use the BR system break command.

MPACK uses the volume's bit map to determine how the pack operation should proceed. Many precautions are taken to preserve data integrity. Occasionally, when some data needs to be moved but there is no free area large enough to accommodate the data, MPACK cannot fully pack a volume. MPACK continues to do what it can, but at the end of its pass through the bit map, all the unused areas may not be successfully combined.

Since at least a partial pack takes place when this happens, a new free area may be created during the packing process that is large enough for the data that was not moved. MPACK makes a second pass to see if a full pack is now possible. Additional passes are taken until a full pack occurs, or nothing was moved on the last pass. MPACK reports on each pass, to let you know what is happening and why the operation is taking longer than usual.

Normally, MPACK does not copy data onto itself. However, the +OK option enables MPACK to overlay data. When you select +OK, only one pass is ever done or required for a full pack to occur. See the discussion of the +OK option below for more information.

The +V option is available to enable visual mode such that MPACK displays a graphical representation of a bit map.

Pack the Volume (+P)

This option causes the volume to be packed. You must enter the LU number as the mask. If you also enter compacting options in the runstring, compacting occurs before packing. This provides for the maximum return of continuous free disk space.

The format for using +P is:

```
mpack [options] +p lu
```

OK to Overlay Data (+OK)

The +OK option gives MPACK permission to copy data onto itself if it cannot be moved to another area on the disk. The danger of having this corrupt your data only occurs if the system goes down or if MPACK is aborted during an overlay. To help preserve data integrity in such situations, whenever an overlay occurs, MPACK reports on which blocks are being overlaid, and where and when the overlay is done. This lets you know if you need be concerned about data integrity, and what steps to take to correct any problem that arises.

The format for using +OK is:

```
mpack [options] +p +OK lu
```

Enable Visual Mode (+V)

The +V option enables visual mode such that MPACK displays a graphical representation of a bit map. The bit map is displayed using ASCII characters to represent portions of the disk LU. The at-sign (@) represents portions of the disk that are completely allocated. The underscore (_) represents portions of the disk that are completely free. Portions of the disk that are partially filled are represented by an asterisk (*). Each character represents the same number of blocks on the disk. The resolution depends on the size of the display being used and the size of the disk LU.

When the +V option is used without the +P option, the bit map is displayed for the disk LU supplied.

When the +V option is used with the +P option, the graphical representation of the bit map is dynamically altered to reflect the progress of MPACK as the disk is being packed.

The format for using +V is:

mpack [*options*] +v *lu*

or

mpack [*options*] +v +p *lu*

The following is an example display using the +V option:

```
CI> mpack +v 47

LU : 47   Total blocks : 259824     Free blocks : 117948
Elapsed time :    0:00:00              Max free : 91678
@**@@**@**@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*@
@@@@**@@*@@@@@@@@@@@*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*@@@@@@*
**@@@@@@*@*@***@*@*@@*@@**@*****@*_******@***___*****@@@@@@@@@*@***@
@@*@*____________*@*@@@**@@**@@@@@@@@@@@@*__*@@@@@**@@@@@*@@@@@@@@@*
*@@@@@@@@@**@@@@@@@@@**@@@@@@*__*@@*@@@@@@@@***___*@@**_**@*_*@@@**@
@***@@@@@*@@@@**@@@@@@@@@@**@@@@@@*____*@@@@@@@@*____**__*@@@@***@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@*__**@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@**@@@
@@@*________*@@@*___________*@@@@@@@@@@*____________________________
____________________________________________________________________
____________________________________________________________________
____________________________________________________________________
____________________________________________________________________
__________________*
Each character represents 288 blocks.
```

Logging Option

MPACK provides a logging option that you can select to direct output to a log file. Compacting and packing operations both support logging.

Logging (+L)

+L causes all the output that is displayed on the screen to go to the log file. When you specify logging, output to the screen is not paged (in other words, the listing does not stop every 22 lines with a "More..." prompt).

If the file name already exists, MPACK fails to run. Log files are not overwritten.

The format for using +L is:

mpack *mask* [*options*] +l <*log filename*>

Examples

Example 1: Remove extents from all files on the working directory.

```
mpack @ +r
```

Example 2: Truncate wasted blocks from files on disk LU 20.

```
mpack +t 20
```

Example 3: Remove extents and truncate wasted blocks, leaving 10 unused blocks remaining, for any file in the working directory or below that has at least two extents or 30 unused blocks.

```
mpack +c10 @.@.s +e2 +w30
```

Example 4: Remove extents and truncate wasted blocks from every file on LU 15 that has 100 wasted blocks and the wasted block percent is equal to or greater than 20 percent of its block usage, and then pack the disk.

```
mpack +c 15 +w100 +w20% +a +p
```

Example 5: Display the bit map of disk LU 10.

```
mpack +v 10
```

Example 6: Pack disk LU 10 and enable visual mode.

```
mpack +p +v 10
```

MV (Move/Rename Files/Directories; VC+ Only)

Purpose: Moves or renames files and directories.

Syntax:

```
mv [-F|-I] [-QV] file1 dest_file
mv [-F|-I] [-QV] file1|mask1 [file2|mask2 ...] dest_directory/
mv [-F|-I] [-QV] file1|mask1 [file2|mask2 ...] dest_mask
mv [-F|-I] [-QV] directory1/ [directory2/ ...] dest_directory/
```

`-F`	any existing destination file is purged before each move without prompting for confirmation. Only write access to the directory is required.
`-I`	issue a prompt requesting confirmation for each move that would overwrite an existing file.
`-Q`	quiet mode; inhibit error reporting.
`-V`	verbose mode.
`--`	end of options; required if the mask begins with hyphen (−).
file1 ... *mask1 ...* *directory1/ ...*	one or more files or directories to be moved.
dest_directory/ *dest_mask*	the directory or destination file mask to which the file or directory is to be moved. When files are moved to *dest_directory*, the files are moved into the directory. When a source mask is specified (*mask1 ...*) or when two or more files are to be moved, the destination must be a directory or a destination mask. When specifying directories in the argument list, a directory may optionally be specified with a trailing slash (*dirname/*) or with the type extension (*dirname*`.DIR`). The trailing slash or the type extension is necessary when a file name without a type extension exists that has the same name as the directory.

Description:

The MV command moves files to new or existing files, or into an existing directory. MV can also move directories into an existing directory.

MV can be used to move a file within the file system or rename an existing file. Directories may not be moved across disk volumes. If a file is "moved" to a different disk volume, the file is copied to the new location and the original file is purged.

When moving a file to a new destination, if the destination file already exists and you have write access to the file, the existing file is replaced by the file being moved.

If a destination file already exists and you do not have write access to the file, MV prompts for confirmation to remove the existing file. When prompting for confirmation, MV displays the file name and its protections and requests for confirmation of the removal of the file. If the response begins with "Y", the existing file is purged and the move is completed.

Return values

$RETURN1	The number of files that could not be moved.
$RETURN2	The number of files that were successfully moved.

Examples

`CI> mv @.c @.ftn @.mac sources`	(Move all of the files with the type extentions .C, .FTN, and .MAC to the subdirectory SOURCES.DIR' in the current working directory)
`CI> mv testprog.@ prog.@`	(Rename all files with the name TESTPROG to PROG while preserving the original type extentions)
`CI> mv subdir/ /dir/`	(Move the subdirectory SUBDIR.DIR to /DIR/SUBDIR.DIR)
`CI> mv subdir/@ /dir/`	(Move all of the files in SUBDIR.DIR to the /DIR/ directory)

NOTIFY (Send a Message to a Terminal)

Purpose: Sends a one-line message to another user's terminal.

Note The NOTIFY utility is installed with Mail/1000 by the INSTALLMAIL.CMD file of Mail/1000.

Syntax:
```
NOTIFY [-a] user message
NOTIFY ON|OFF
```

`-a` inhibits notifying alias logons; treats *user* as a logon name only.

user specifies who is to receive the message, and may name either:

- a logon name, in which case all sessions logged on under that name are notified.
- a session number, in which case only that session is notified.
- an alias name from /SYSTEM/NOTIFY.CF, in which case the message is sent to all users defined therein.
- the name "all", in which case the message is sent to all sessions logged on.

message	is your one-line message to be sent by the NOTIFY utility.
`on|off`	turns notification for your session on or off. If notification is turned off, then messages sent to your session are thrown away without being printed.

Description:

The NOTIFY utility sends a one-line message to another user's terminal, in the format:

```
Message from logon[>node] date-time...
your-message-here
```

The recipient of the message is specified as either a logon name or an alias name defined in the /SYSTEM/NOTIFY.CF file, which is described below.

In the example,

```
CI> notify joker 'I'm Batman.'
```

the NOTIFY utility will send that message to every session logged on as JOKER on this system, plus any alias logons for JOKER defined in NOTIFY.CF.

Defining Aliases for Notification

NOTIFY may send messages to aliases that are defined for:

- alternate logons for a user
- group distribution lists
- hosts in a DS network

Alternate Logons for a User

The /SYSTEM/NOTIFY.CF file defines alternate logons to also be notified when a message is sent to a user. For instance, if user MAGNOLIA often logs on under user name MANAGER, an entry may be defined that aliases MAGNOLIA to MANAGER, such that all messages destined to MAGNOLIA are also sent to anyone logged on as MANAGER. As shown below, aliases may be created that notify any of several possible logons MAGNOLIA may be using, on any of several hosts in her DS network.

Group Distribution Lists

Group distribution lists may be defined, which cause messages sent to the group name to be sent to each logon in the group. For instance, group LUNCHERS may be defined to include several users who normally go to lunch together. To notify each person in the group that you are ready for lunch you might enter:

```
CI> notify lunchers 'Lunch,anyone?'
```

Hosts in a DS Network

For hosts in a DS network, a DS node name or number may also be given in the alias definitions, such that users on other HP 1000 hosts in your DS network may be notified. The remote hosts must be executing the M1KSS RTE-A monitor program to receive the remote requests, and must be able to execute the NOTIFY utility to deliver the message to users on that host. (Like the NOTIFY utility, the M1KSS monitor program is installed as part of Mail/1000 installation.)

Alias Definition Format

Each line of the /SYSTEM/NOTIFY.CF file is either blank, a comment with a star (*) in the first column, or an alias definition in the form given below.

name`: alt` *aliasname* `[,`*aliasname*`...]`

Entries in the format above define alternate notification logons that should be notified when a message is sent to *name* on this host.

aliasname entries are of the following form:

logon`[>`*node*`]`

where *node* is the name or number of the host on your DS network where user *logon* logs on.

For example, the definition:

```
superman: alt clark, kent>dplanet
```

specifies that when a message is sent to Superman on this host, the message should also go out to anyone logged on as CLARK on this host and to anyone logged on as KENT on DS node DPLANET.

OF (Stop/Remove Program)*

Purpose: Terminate a program and optionally remove its program ID segment, or remove a prototype ID segment.

Syntax: `OF [`*prog* `[`*opt*`]]`

prog is the program name, up to five characters, with an optional session identifier or the name of a prototype ID segment if the D option is used.

opt is either of the following options:

`ID` Releases the named program ID segment.

`D` Releases the named prototype ID segment.

Description:

If the D option is NOT given, this command terminates the named program by stopping the program, flushing I/O, and releasing resources such as LU locks. However, files are not posted, and the ID segment is released. (If the program was restored previously with the RP command, the ID parameter must be used to free the ID segment.) All program attributes revert to those defined when the program was linked after the ID segment was released.

If the prog parameter is not specified and the startup program (usually CI) 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 OF command is executed on the last program. The only exception is that if the last program is a protected system program, the program that scheduled it will be stopped or removed.

If the D option is given, this command removes the prototype ID segment of the named program.

In a VC+ environment, a superuser can use this command to remove any program if the need arises. General users can only remove non-system programs in their own session and other sessions with the same user name.

OLDRE (Extended Record Converter)

Purpose: Translates extended relocatable record formats to non-extended formats for loaders that require them. This provides backward compatibility with loaders and generators that do not accept extended record formats.

Syntax: `OLDRE` *file_desc*

file_desc identifies the relocatable file that combines old format and extended format records to be translated. The relocatable file must be a type 5 (object program) file.

OLDRE Extended Records

The term "extended record" refers to the set of extended relocatable records that allow 16-character EXT and ENT names and other extended features that are supported by the Macro Assembler and by language compilers. Additional information can be found under Relocatability in the *Macro/1000 Reference Manual*, part number 92059-90001. For other languages, check the appropriate language reference manual for the record forms used.

Extended records are identified as 7 in the identifier field (bits 13-15) of record word 2, and with a sub-identifier in bits 6-12, as shown below:

SubIdent	Extended Record	Type
0	GEN	Generator information/command record.
1	XNAM	Extended name relocatable program. Superset of NAM.
2	XENT	Extended entry point. Superset of ENT.
3	XDBL	Extended define left byte address. Superset of DBL.
4	XEXT	Extended external reference. Superset of EXT.
5	XEND	Extended terminate program. Superset of END.
6	RPL	Contains code replacement information.
8	MSEG	Declare EMA memory segment size.
9	ALLOC	Combined ENT/EXT for FORTRAN block data.
10	DEBUG	Contains information for Debug/1000.
20	LOD	Contains information/command for Loader/Linker.

OLDRE Operation

OLDRE creates a scratch file in which the translated relocatable file is built. Each record is scanned and, if a non-translatable feature is encountered, the record is skipped and an appropriate message is issued.

The program displays the following message when the first extended record is encountered:

```
OLDRE: Converting new relocatable records to old format
```

OLDRE does not terminate on an error, but continues to translate the entire file. If the program encounters non-translatable code, the scratch file is purged after the translation attempt, and the original relocatable file is left unchanged.

If the relocatable file is not type 5, or the file does not reside on a mounted cartridge, OLDRE issues one of these messages and exits:

```
OLDRE: Input must be relocatable file – type 5
OLDRE: Relocatable must be a disk file
```

OLDRE also aborts if either a bad record checksum is encountered, the break flag is set, or an FM error occurs.

After successful translation, the original relocatable file is purged. The scratch file is truncated to the exact size required to hold the translated code, and renamed with the original file name. The resulting translated file can then be loaded to execute under the appropriate operating system.

The OLDRE exit message defines the success or failure of the translation process:

```
OLDRE: End OLDRE – No errors
OLDRE: End OLDRE – Relocatable file unchanged.
```

OLDRE processes Macro code at approximately 800 lines per second and FORTRAN code at approximately 133 lines per second.

Translation Results

The input instruction to OLDRE and the translated output are listed below.

Input	Output	Comment
NAM	NAM	
XNAM	NAM	Non-zero EMA size, SAVE size, or pure code size will cause an error. Program names longer than five characters will cause an error.
ENT	ENT	
XENT	ENT	An ENT to EMA or externals will cause an error. An entry point name longer than five characters will cause a warning.
EXT	EXT	
XEXT	EXT	Weak external will be translated to a strong reference. Names longer than five characters are warnings.
DBL	DBL	
XDBL	DBL	Byte externals and DDEFs to EMA will cause an error. If the memory area to be initialized is not absolute, common, or program relocatable an error will be issued.
ALLOC	ENT	Translation if ALLOC is to be labeled common in FORTRAN BLOCK DATA. Any DBLS to this ALLOC have external ID number deleted.

ALLOC	EXT	Translation for all other cases except ALLOC to SAVE (illegal). Any DBLs to this external are illegal. Only one symbol per record is legal and it must be a COMMON ALLOC. Note that only program type and name will appear in the NAM record for this module.
MSEG		Error. Will not be translated.
EMA	EMA	
RPL	ENT	
END	END	
XEND	END	
GEN	GEN	
LOD	LOD	
DEBUG		Ignored. No error or record output.

Program Restrictions

The following restrictions must be observed in preparing the source programs to ensure that the relocatable code can be successfully translated.

Macro

1. External, entry point, and variable names cannot exceed five characters. Names exceeding five characters are truncated to characters 1 − 4 and the last character.
2. The ALLOC, DDEF, and MSEG commands cannot be used.
3. The RELOC and ORG commands cannot be used with EMA or SAVE, or with pure code.
4. Two-word RPLs cannot be used.
5. DBLs and DBRs cannot refer to externals.
6. No more than 255 externals can be referenced in one module.
7. Negative and multiple relocatability cannot be used (for example, constructs such as DEF −W or DEF W+W, where W is not absolute).

Pascal

1. Level 1 procedure names and program names cannot exceed five characters, unless ALIASed to an external name of less than six characters. Names exceeding five characters in length are truncated to characters 1 − 4 and the last character.
2. No more than 255 level 1 procedures can be contained in one program.

OLDRE

FORTRAN

1. The SAVE command cannot be used.
2. Subroutine, function, program or variable names cannot exceed five characters. Names exceeding five characters are truncated to characters 1 − 4 and the last character.
3. Local EMA cannot be specified.
4. Labeled EMA common cannot be used.
5. Character data cannot be in labeled common.
6. No more than 255 common block names, external subroutines, or function calls can be contained in one module.

Error Messages

The following messages describe conditions encountered during the translation process and define errors that cause cancellation of the translation.

OLDRE: Local EMA, SAVE and CODE area illegal

OLDRE: New feature not available in old format

OLDRE: New feature in RPL illegal in old record

OLDRE: More than 255 externals found

The following message is treated as a warning and, if the only problem, does not cancel the translation:

OLDRE: Symbol name truncated to 5 characters

Note that in this case, the load may fail due to duplicate externals caused by the loss of the truncated characters.

OWNER (Display/Change Owner; VC+ Only)

Purpose: Displays or changes the owner of a directory or a subdirectory.

Syntax:
```
OWNER directory [newOwner]
OWNER luV [newOwner]
```

directory is the name of the directory or subdirectory. No wildcard characters are allowed.

*lu*V is a CI volume number followed by the character "V".

newOwner is the name of the new owner. This is 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 or CI volume. Only the current owner can assign ownership. Ownership is associated with volumes, directories, and subdirectories, but not with the 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 is 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 logon name is changed with the USERS program. Note that if a removable disk is moved to another system with different user accounts, ownership will not be correct.

The *newOwner* parameter must be a name usable for logon, that is, there must be a user account with that name on the system. The group account name in the *newOwner* parameter is made the associated group of the directory or volume. If a group account name is not specified in the *newOwner* parameter, the default logon group is assumed to be the group account. If NOGROUP is made the associated group, it is equivalent to not having an associated group, and group protection is thus turned off.

PATH (Display/Modify UDSP; VC+ Only)

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. This is used when echoing is not desired from a command file or when information is desired only in the return parameters and is not to be 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 (the maximum is eight).
- `-A` All UDSPs defined for the 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, without the −E parameter, sets the $RETURN variables and 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. If the −E parameter is specified, the $RETURN variables are set, but no information is displayed.

The second format displays or defines a specific UDSP or a specific entry of a UDSP, and sets the $RETURN variables. If the −E parameter is specified, the specific UDSP or specific entry of a UDSP is defined and/or the $RETURN variables are set, but no information is displayed.

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.

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. We recommend that you always make /PROGRAMS the last entry in the search path. Note that if you define UDSP #1, your executable file must have .RUN as the file type extension.
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. We recommend that you always make /CMDFILES the last entry in the search path. Note that if you define UDSP #2, your command file must have .CMD as the file type extension.
UDSP #3	Used by some subsystems to find library files.

UDSPs #4 through #8 can be used for your application programs. See the description of FmpOpen in the *RTE-A Programmer's Reference Manual* and the description of the #n directory specifier in Chapter 3, "Manipulating Files" in this 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. THEN and ELSE test this parameter.
$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:

PATH

```
CI> path                                (Display current UDSP information)

CI> path 1                              (Display UDSP #1)

CI> path –a                             (Display all UDSPs)

CI> path 0 /mine                        (Set home directory to /MINE)

CI> path 2 . /mine/cmdfiles /cmdfiles   (Set UDSP #2 to the following:
                                          (1) current working directory
                                          (2) /MINE/CMDFILES
                                          (3) /CMDFILES)

CI> path –e –f setpath.cmd              (Read PATH commands from the file
                                         SETPATH.CMD without echoing messages
                                         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)

CI> path –n:3 1                         (Display the third entry of UDSP #1)

CI> path –n:1 2 /groups/cmds            (Set the first entry of UDSP #2)

CI> path 3 !                            (Delete all entries of UDSP #3)

CI> path –n:2 4 !                       (Delete the second entry of UDSP #4)

CI> path –e –n:2 4 !                    (Return the contents of the second entry of
                                         UDSP #3 in $RETURN_S without echoing
                                         the name to the terminal)

CI> path –e 0                           (Return the name of the home directory
CI> wd $return_s +s                      without echoing it and then set the
                                         working directory to the home directory)
```

POLL (Polling Function)

Purpose: Executes a specified CI command synchronously with respect to user interaction and terminal timeouts.

Syntax: `POLL` *interval*`|OFF` *command*

interval`|OFF`	if a number (*interval*), it is the approximate number of minutes between executions of the poll command.
	if `OFF`, the poll function is turned off.
command	is any CI command or program to be executed at the poll interval.

Description:

Each time CI prepares to issue a prompt, it first checks to see if the current time minus the base time is greater than the poll interval. If it is, CI executes the poll command, sets the base time for the next poll interval to the current time, and issues the CI prompt.

The command to be executed and the poll interval are stored in CI variables $POLL and $POLLINT, respectively, so that you can see them by doing a SET command in CI.

Examples:

`CI> poll 1 dl`	(Execute the DL command and set the poll interval to one minute)
`CI> poll`	(Execute the previously set poll command and reset the base time)
`CI> poll 7`	(Execute the previously set poll command and reset the poll interval to seven minutes)
`CI> poll off`	(Turn off the polling function)

Notes:

1. $POLL can be altered with the SET command. This has the effect that the new command will be executed the next time the poll interval is exceeded.

2. $POLLINT cannot be altered with the SET command.

3. $POLL and/or $POLLINT can be deleted with the UNSET command. If either or both variables are deleted the polling function is turned off.

PR (Change Program Priority)*

Purpose: Changes priority of a restored program. It can also be used to display the priority of a program.

Syntax: PR *prog* [*priority*]

prog is the program name, up to five characters, with an optional session identifier.

priority is the program priority, a number between 1 and 32767. If omitted, the priority of the specified program is displayed, if an ID segment exists for that program.

Description:

The program whose priority is to be changed must be a restored program. The priority of system utility programs can be changed by superusers. Program priority determines when a program will run relative to other programs; lower numbers represent higher priorities and are more important. Priority values typically fall within 50 to 200.

Be careful when setting programs to priority numbers lower than 40, as such programs can interfere with normal system operations if they use system resources excessively. Your choice of program priority may also be influenced by the system priority fence and timeslice fence; refer to the *RTE-A System Design Manual* for more information.

PROT (Display/Change Protection; VC+ Only)

Purpose: Displays or changes the protection status of a file, directory, or volume.

Syntax:
```
PROT fileMask [newProtection]
PROT luV [newProtection]
```

fileMask is 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 file masking.

*lu*V describes a CI volume LU to display or change (for example, 15V).

newProtection defines the new protection status for the owner, members of the owner's group, and others.

The syntax for the *newProtection* parameter is:

```
owner[/group]/others
```

The slash is a required delimiter. The protection values are:

R = allow read access
W = allow write access

If both R and W are specified, they may be given in either order. If no protection value is given in a particular position, it disallows all access. If the group protection is not specified, it will remain unchanged.

As an alternative to the R and W symbols, a set of default symbols may be specified. These symbols will allow the current protection values to be transferred into the new protection setting. The symbols are:

U = user (owner)	Place current owner protection here.
G = group	Place current group protection here.
S = system (others)	Place the current other protection here.

Description:

If new protection is not specified, this command displays the current protection on the files that match the mask or on the CI volume. If new protection is specified, all files that match the mask or the CI volume will have their protection changed to the new protection.

When the current protection is displayed for a file mask, PROT actually executes the command "DL *mask* P", which shows the current protection for all files matching the mask (volume protection is displayed without DL).

When a global directory is created, the owner is set to the creator and the default protection is RW/R/R. When a CI volume is initialized, the owner is set to the one doing the initialization and the default protection is set to RW/RW/RW.

To change protection on a file or volume, the user must be the owner of the directory on which the file resides or of the volume itself. Protection of a CI volume restricts the displaying, creating, and purging of global directories on that volume.

PROT

For VC+ only, when the variable $QUIET_CMD is set to ON, the message

```
    Setting protection on FILE1 ... [ok]
```

is not displayed.

Examples:

`CI> prot /libraries/@.@`	(See protection for all files in /LIBRARIES)
`CI> prot message rw/rw/rw`	(Allow full access by everyone)
`CI> prot groupbox rw/rw/`	(Allow access only to owner and group members)
`CI> prot oldfile rw/r`	(Set access but leave previous group value)
`CI> prot myfile rw//`	(Only the owner can access the file)
`CI> prot myfile /rw/rw`	(Only the owner cannot access the file)
`CI> prot safe //`	(No one (except superusers) can access the file)
`CI> prot 15v`	(Display the protection for CI volume 15)
`CI> prot 15v rw/rw/r`	(Allow the owner and group members only to create global directories on CI volume 15)

Assume protection on FILE is currently rw/w/r:

`prot file r/g/s`	(Change owner access only (result is R/W/R))
`prot file u/g/rw`	(Change others access only (result is RW/W/RW))
`prot file u/s/s`	(Do not change owner and others access, but change group access to the same as others (result is RW/R/R))
`prot file s/u/g`	(Shuffle protection around (result is R/RW/W))

PS (Display Program Status)*

Purpose: Displays the status of the specified program or the currently executing program.

Syntax: `PS [`*program*`]`

program is the name or ID segment of a program for which status is required.

Description:

If you omit the program parameter, the status of the currently executing program is displayed. If the specified program does not exist, the following message is displayed:

```
No ID segment for this program.  Try RP,program.
```

If the program does exist or if you enter the command without the *program* parameter, the response is displayed in the following format:

prog `PR(`*priority*`) PC(`*addr*`) [M] [T] [L] [S]` *status*

where:

prog	Program name.
priority	Priority of the program.
addr	Memory address of the last instruction executed.
`M`	Program is memory-resident.
`T`	Program is time scheduled.
`L`	Program is being loaded from disk.
`S`	Program is being swapped to disk.

The valid codes for the status parameter are as follows:

`AB`	The program is being aborted.
`BL(`*lu*`)`	The program requested buffered output or Class I/O to the given lu, but the device already has requests exceeding its allotment of SAM.
`CL`	Either the program issued a Class I/O get (EXEC 21) and the queue for the requested class number is empty, or the program attempted to allocate a class number and none was available.
`DL(`*lu*`)`	The program was suspended because it made an I/O request to the given lu and the lu is down.
`IO(`*lu*`)`	The program is waiting for an I/O request to complete.
`LD`	The program made an EXEC 28 or EXEC 8 request to restore a real-time program or statement from the disk. The program will continue when the load is initiated.

LK(*lu*)	The program was suspended because it made an I/O request to the given lu and the lu is locked to another program.
MM	The program is waiting for System Available Memory (SAM) to become available. SAM is required for buffered output, Class I/O, and string passing.
OF	The program is dormant. (The program either was never scheduled, has run to completion, or has been aborted.)
PA	The program suspended itself with a PAUSE statement or an EXEC 7 call. You can reschedule the program with parameters (see the GO command).
QU(*progb*)	The program attempted to schedule *progb* with an EXEC 23 or EXEC 24 call, but *progb* was already active. When *progb* becomes dormant, it will be scheduled as the child of the program.
RN(*progb*)	The program is waiting for *progb* to release a resource number.
RN(-GLB)	The program is waiting for a resource that is locked globally.
RN(-NONE)	The program attempted to allocate a resource number and none was available.
SC	The program is scheduled (waiting for CPU time) and will execute when all higher priority programs are suspended or dormant.
SP	Spool/class buffer limit suspended.
SR	The program is suspended waiting for another program to exit a shared subroutine.
SS	The program was suspended with the SS command. You can reschedule the program with the GO command, but parameters cannot be passed to the program.
TM	The program requested a timed delay of its own execution (EXEC 12 call, with name = 0).
WT(*progb*)	The program scheduled *progb* with wait (EXEC 9 or EXEC 23) and will continue execution when *progb* completes.
XS	The program is waiting for XSAM to become available. XSAM is required when using signals or UDSPs.
XQ	The program is executing.

Examples:

```
CI> ps test1
TEST1 PR( 99) PC(    0) OF

CI> ps
CI PR( 51) PC(66330) XQ
```

PU (Purge Files)

Purpose: Purges files.

Syntax: `PU` *mask* `[OK]`

mask is 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 file masks.

`OK` is the optional parameter that instructs the program to purge the specified files without prompting.

Description:

A wildcard purge is one in which a mask is used to specify one or more files to be purged. It is a powerful capability, but should be used with great care in order to avoid purging files that you meant to keep.

Caution If a mask is not specified, no files will be purged unless the optional parameter `OK` is given, in which case all files will be purged. That is, `PU,,OK` purges all files in the current working directory.

If the file mask you enter specifies a single file, only that file is purged and the following message is displayed:

```
Purging <file descriptor>
```

FMGR files with a security code must be purged individually, with the security code specified.

When a group of files is to be purged, the program provides some checking to make sure you really want to purge all the files, by prompting you interactively to confirm each file that matches the mask:

```
Purging <file descriptor> (Yes, No, Abort, Stop Asking) ? [Y]
```

The program steps through the files, prompting you to respond for each file. This allows you to confirm the file selection or change your mind before the purge is done by entering one of the responses shown in NO TAG.

You can avoid having the program perform this checking by using the OK parameter, which indicates that the purge is set up as intended. OK causes all the files that match the mask to be purged without prompting or any other intervention. The program displays the file names on the log device as the files are purged, but no further confirmation is required.

If the input device is not an interactive device and the OK parameter is not specified, wildcard purges are not executed.

To purge a file, you must have write access to the directory that contains it. The file must not be an active type 6 file, the system swap file, an opened file, or a directory containing files.

When purging a file that is a symbolic link, the link is purged and not the file to which it points.

PU

PU Responses

Response	Action
Y or <cr>	Purge the file named.
N	Skip this file.
A	Abort the purge.
S	Purge all the files that match the mask (you will not be prompted again).

The PU command can be used to purge an empty directory. Note that the format of the command to purge a global directory is "`PU /GLOBAL`". The command "`PU ::GLOBAL`" or "`PU /GLOBAL`" will purge all the files on the GLOBAL directory, but not the directory itself. In this case, the form /GLOBAL is not the same as ::GLOBAL and does not produce the desired results.

If there are non-empty subdirectories under the GLOBAL directory, you can purge them and their contents by repeatedly entering the following command:

```
CI> pu /global/@.@.s
```

For VC+ only, when the variable $QUIET_CMD is set to ON, the message

```
Purging FILE1 ... [ok]
```

is not displayed.

Examples:

```
CI> pu /goal/file1
```
(Purge FILE1 in directory GOAL)

```
CI> pu @.temp ok
```
(Purge all files in working directory with file type extension .TEMP)

```
CI> pu /joe/
```
(Purge all files in global directory JOE)

```
CI> pu /joe
```
(Purge global directory JOE)

```
CI> pu /test/two.dir
```
(Purge subdirectory TWO. Note that the file type extension .DIR is required here to avoid confusion with files named TWO)

PWD (Display Working Directory)

Purpose: Displays present working directory.

Syntax:
```
PWD
PWD [-p]   (VC+ Only)
```

Description:

The PWD command displays the current working directory. The −P option is only available when $VISUAL is set to any of the ksh-style command line editing modes (EMACS, GMACS, or VI). This option causes the PWD command to display the physical name of the current working directory. See the CD command for more information regarding logical versus physical naming of the current working directory.

When $VISUAL is set to anything other than a ksh-style editing mode, PWD reports the physical name of the current working directory. The −P option is ignored in this case.

RESIZE (Set $LINES/$COLUMNS Variables; VC+ Only)

Purpose: Sets $LINES and $COLUMNS environment variables by querying an HP terminal for its physical screen size.

Syntax: `RESIZE`

Description:

RESIZE uses HP terminal escape sequences to query a terminal for the size of its physical screen. This is the size of the area that can be viewed without scrolling the screen either vertically or horizontally. If it gets the size successfully, the size is returned in $RETURN2 and $RETURN3. If, in addition, an environment variable block is accessible, RESIZE will set the $LINES and $COLUMNS environment variables that are used by various utilities in order to format their output. Typically, RESIZE must only be run when a session is started or when the session's terminal (or terminal emulator) screen size changes.

Return Values:

$RETURN1 = 0 command executed successfully. $COLUMNS, $LINES, $RETURN2 and $RETURN3 are set to the values obtained from querying the terminal.

= 1 error when querying terminal. $COLUMNS and $RETURN2 are set to 80. $LINES and $RETURN3 are set to 24.

= 2 error when setting environment variables. $COLUMNS and $LINES are not set. $RETURN2 and $RETURN3 are set to the values obtained from querying the terminal.

= 3 error when querying terminal and setting environment variables. $COLUMNS and $LINES are not set. $RETURN2 is set to 80 and $RETURN3 is set to 24.

$RETURN2 The width of the physical screen.

$RETURN3 The height of the physical screen.

Example:

```
CI> resize                              (Set $COLUMNS and $LINES)
CI> if is $columns < 80 –i              (If $COLUMNS is less than 80,
THEN > then                               print out a message)
FI > echo 'This is a narrow screen'
FI > fi
```

RETURN (Return from Command File)

Purpose: Returns to previous level of command file nesting or to interactive mode.

Syntax: `RETURN[,`*return1*`[,`*return2*`[,`*return3*`[,`*return4*`[,`*return5*`[,`*return_s*`]]]]]]`

return1 is an integer return status indicating success or failure of command file. Zero means success; nonzero means failure. If omitted, return1 is set to zero.

return2 - 5 are four integer values made available to return additional status information. Each omitted parameter is set to zero.

return_s is a string of up to 256 characters for VC+, 80 characters for non-VC+; if omitted, it is set to null.

Description:

RETURN lets you exit 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; otherwise, 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:

```
CI> return,0,2,3,4,5, 'Command file successful'
```
(Exits a command file and specifies 5 integer values and a string)

```
CI> return
```
(Exits a command file using the default return values ($RETURN1 through $RETURN5 are set to 0, $RETURN_S is set to a null string))

```
CI> return,,,,,,'Successful Completion'
```
(Exits a command file, returning a value in only the variable $RETURN_S. $RETURN1 through $RETURN5 are set to the default value of zero.)

RM (Remove Files or Directories; VC+ Only)

Purpose: Removes files or directories.

Syntax:
```
rm [-F|-I] [-QV] file|mask ...
rm [-F|-I] [-QV] -R directory/ ...
```

`-F` purge files or directories without prompting for confirmation, regardless of the permissions of the file.

`-I` issue a prompt requesting confirmation before removing each file.

`-R` recursively purge the entire contents of any directory that is specified as an argument and then purge the directory itself.

`-Q` quiet mode; inhibit error reporting.

`-V` verbose mode.

`--` signifies the end of the options; required if *mask* begins with a hyphen (−).

file|*mask* ... is one or more files to be purged.

directory/ ... is one or more directories to be purged. The −R option is required when purging directories and their contents. A directory may be specified with a trailing slash (*dirname/*) or with the .DIR type extension (*dirname*`.dir`). The trailing slash or the type extension is necessary when a file name without a type extension exists that has the same name as the directory.

Description:

The RM command purges files and/or directories. You must have write access to the directory of a file to purge a file, but do not need write access to the file itself.

If you do not have write access for a file to be removed, RM prompts for permission to purge the file. When prompting for permission, RM displays the file name and its protections and requests for confirmation of removal of the file. If the response begins with "Y" the file is purged; otherwise, the file remains.

When purging a file that is a symbolic link, the link is purged and not the file to which it points.

Return Values:

$RETURN1 The number of files that could not be purged.
$RETURN2 The number of files successfully purged.

Examples:

```
CI> rm -rf /oldprogs
```
(Purge all of the files in /OLDPROGS.DIR and then purge the directory /OLDPROGS.DIR)

```
CI> rm file1 file2 subdir/@
```
(Purge FILE1 and FILE2 in the current working directory and then purge all of the files under the SUBDIR.DIR subdirectory)

```
CI> rm subdir/
```
(Purge the subdirectory SUBDIR.DIR; SUBDIR.DIR must be empty)

RN (Rename File, Directory, or Subdirectory)

Purpose: Renames a file, a directory, or a subdirectory.

Syntax: `RN` *file1 file2*

file1 is the source file descriptor; can be masked to operate on more than one file (refer to the "File Masks" section in Chapter 3).

file2 is the destination file descriptor; can be masked to allow the system to generate destination names.

Description:

The RN command changes the name, file type extension, or any combination of the above of *file1* to those for *file2*. The new name must not already exist in the directory. You must have write access to the directory.

Directories and subdirectories can be renamed. This command does not move files into a different directory. If the directory field of the destination file name is blank, the source directory is used. If source and destination directories are different, an error message is displayed. In this case, use the MO or MV command.

For VC+ only, when the variable $QUIET_CMD is set to ON, the message

```
Renaming FILE1 to FILE2 ... [ok]
```

is not displayed.

Examples:

`CI> rn foo joe`	(Rename file FOO on the working directory to JOE)
`CI> rn foo.txt @.ftn`	(Change the file type extension of FOO from .TXT to .FTN)
`CI> rn @.src @.ftn`	(Change all files with file type extension .SRC to .FTN)
`CI> rn Joe/SMITH FOOBAR`	(Rename file SMITH in subdirectory JOE to FOOBAR)

RP (Restore Program File)

Purpose: Establishes a program or prototype ID segment.

Syntax: `RP` *file* [*newname* [*options*]]

file is the 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.

newname is the new program name to be used instead of file name, up to five characters.

options is a character string that contains "C", "D", "P", "T", or "CP" (that is, both C and P) to select the following options:

- `C` Create a clone name if the specified or assigned name is already assigned to an RP'd program.
- `D` Create a prototype ID segment, NOT a program ID segment, for the program file.
- `P` Create a permanent program ID segment that will not be released when the program terminates. This is the default if *no* options are given.
- `T` Create a temporary program ID segment that will be released when the program terminates. This is automatically added to any other option that is given.

Description:

If the D option is NOT given, 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; for example, the WS, VS, or SZ command. If newname is not specified, it is derived from the file name.

If the D option is given, the RP command creates a prototype ID segment that is used to quickly create program ID segments when that program is scheduled.

If the T option is used to RP a program, either explicitly or by default, run the program using the :IH suffix. For example:

```
CI> RP,PROG,MYPROG,C
CI> MYPROG:IH
```

The RP command performs the following steps when searching for the program to restore:

1. If the specified or defaulted program name already exists as a "system utility":

   An error is issued. A system utility is one that is loaded using the LINK SU command to ensure that only one copy of the program may be present in the system at one time.

2. If a directory path is given in the specified file descriptor:

   The named directory is searched for the specified type 6 file. If the file is found, it is selected for restoring, and CI proceeds to step 5. If the file is not found and no file type extension was specified in the file descriptor, the directory is searched for a type 6 file with the given name and a .RUN type extension. If the file is still not found, an error is returned.

3. If steps 1 and 2 above do not apply, CI searches for an existing program that matches the first six characters in the given file descriptor, searching in your session and in the system session. If this search fails then the list of "prototype ID segments" in XSAM is searched for an entry with the specified name. If either an existing program or a prototype ID segment is matched then:

   a. If a program name is matched and the program is "permanently" restored, the ID segment is selected for "cloning". CI will proceed to step 5 to create a new ID segment with the same attributes as the existing ID segment, but with a unique program name and with status "dormant". Note that programs that were RP'd with the T (temporary) option are not selected for cloning. Also note that programs loaded as "system utilities" are never cloned; this case is handled in step 1.

   b. If an entry in the prototype ID segment list is matched, that proto-ID is selected for cloning, in much the same manner as when an existing ID segment is cloned. CI then proceeds to step 5.

4. If steps 1 and 2 above do not apply and step 3 fails to find a match, CI searches for a type 6 file, using the following algorithm to determine the order of directories searched. Note that if a directory is specified in the file descriptor then that case is handled in step 2.

   If User Defined Search Path (UDSP) #1 is defined for your session then that path is used to find the file. A .RUN file type extension is assumed if none is supplied. Each directory named in the UDSP is searched in order until the type 6 file is found. If the file is not found then an error is issued. Refer to Chapter 3 for a description of UDSPs.

   If UDSP #1 is not defined, CI uses the default directory search list to find the type 6 file:

   a. If your session has a working directory defined, that directory is searched for the file. If not found and no type extension was specified in the file descriptor, the working directory is searched again with type extension .RUN defaulted.

   b. If your session does not have a working directory, each mounted FMGR cartridge is searched for the type 6 file.

   c. If the file is still not found, global directory /PROGRAMS is searched, with type extension .RUN defaulted. If the file is not found, an error is issued.

5. If one of the above steps found a type 6 file, existing program, or prototype ID segment to use in creating the new program, CI tries to find a name for the new ID segment. If the specified or defaulted program name does not exist in both your session and in the system session then that name is used. If the specified or defaulted program name already exists then:

   a. If the C option is not specified, an error is issued.

   b. If the C option is specified, CI generates a unique name for the program by preserving the first three characters of the name and replacing the last two characters with ".A", ".B", and so forth, until a name that is not already taken is found.

RS (Restart Program)

Purpose: Restarts a program.

Syntax: `RS [`*program*`[/`*session*`]] [`*pram*`*5]`

program is the 5-character name of the program that is to be OF'd and restarted. If not specified, the default is your session's first program.

session is the session in which the program resides. Defaults to your session.

*pram**5* are parameters to be passed to the program. This can be five parameters or one long character string. The maximum runstring length, including the RS command and delimiters, is 256 characters.

Description:

The RS command can be used to restart a program that is not executing properly (for example, a program that is hung on a downed device). The command is especially useful if your session's first program (usually CI) is in this state because if you were to OF this program and it was the only one in the session, your session would be terminated.

Only a superuser can restart a program in another session or a system utility.

The RS command is available only in CM.

RU (Run Program)*

Purpose: Schedules a program for execution and waits for its completion. Note that the CI and base set versions of this command are different. See the following section "Base Set Version of the RU Command" for information on the base set version.

Syntax: `[RU]` *prog|file* `[`*pram*5*`]`

`RU` is 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 can be confused with a command file (see the sections on the TR command and predefined variables).

prog|file is the 5-character program name or a file descriptor that identifies a type 6 file. If the program name is suffixed with :IH (for example, PROG1:IH) then cloning of the program is inhibited, that is an existing program.

*pram*5* are the parameters to be passed to the program. The maximum runstring length, including the implied RU and delimiter, is 256 characters. This can be five numeric parameters or a character string.

Description:

If the program is not restored, CI restores it and frees the program ID segment after it finishes running. CI modifies the program name if necessary to make it unique when it restores the program. The last two characters will be changed to .A, .B, and so forth.

We recommend that all type 6 file names use the .RUN file type extension.

If you are executing more program files than command files, 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 procedure described below is used to select the program to execute.

1. If the specified program name or type 6 file descriptor names an already-restored program that is loaded as a "system utility":

   The existing program is chosen for execution. A system utility is one that is loaded using the LINK SU command to ensure that only one copy of the program may be present in the system at one time. If the system utility is busy (that is, non-dormant) then CI will "queue suspend" on the program until it becomes available.

2. If a directory path is given in the specified file descriptor:

   The named directory is searched for the specified type 6 file. If the file is found, it is chosen for execution. If the file is not found and no file type extension was specified in the file descriptor, the directory is searched for a type 6 file with the given name and a .RUN type extension. If the file is still not found, an error is returned.

3. If steps 1 and 2 above do not apply then an existing program with the specified name is searched for in your session and in the system session. If this search fails then the list of "prototype ID segments" in XSAM is searched for an entry with the specified name. If either an existing program or a prototype ID segment is matched, then:

   a. If a program name is matched and the program is either: (i) dormant and "permanently" restored, or (ii) dormant "saving resources", then that program is selected for execution. If the program was RP'd with the T (temporary) option and is either dormant (not saving resources) or busy, it is not selected either for execution or for "cloning", as described below; CI falls through to step 4 in this case.

   b. If a permanently-restored program name is matched but the program is busy then:

      (i) If the program was suffixed with :IH, an error is reported.

      (ii) If :IH was not given, a clone of the ID segment is made. A new ID segment is created with the same attributes as the existing ID segment, but with a unique program name and with status "dormant". This new program is selected for execution. Note that programs loaded as system utilities are never cloned; this case is handled in step 1.

   c. If an entry in the prototype ID segment list is matched, a new ID segment is created from the prototype ID segment, in much the same manner as when an existing ID segment is cloned. This new program is selected for execution.

4. If steps 1and 2 above do not apply and step 3 fails to find a match, CI searches for a type 6 file, using the following algorithm to determine the order of directories searched. Note that if a directory is specified in the file descriptor, that case is handled in step 2.

   If User Defined Search Path (UDSP) #1 is defined for your session, that path is used to find the file. A .RUN file type extension is assumed if none is supplied. Each directory named in the UDSP is searched in order until the type 6 file is found. If the file is not found, an error is issued. Refer to Chapter 3 for a description of UDSPs.

   If UDSP #1 is not defined, CI uses the default directory search list to find the type 6 file:

   a. If your session has a working directory defined, that directory is searched for the file. If not found and no type extension was specified in the file descriptor, the working directory is searched again with type extension .RUN defaulted.

   b. If your session does not have a working directory, each mounted FMGR cartridge is searched for the type 6 file.

   c. If the file is still not found, global directory /PROGRAMS is searched, with type extension .RUN defaulted. If the file is not found, an error is issued.

For example, if a working directory exists and UDSP #1 is undefined, the search sequence for program EDIT specified in "RU,EDIT" is as follows:

1. Search for a restored (RP'd) EDIT that is dormant.
2. Search for a permanently restored, busy, and clonable EDIT.
3. Search for EDIT in the proto-ID list.
4. Search for EDIT in the working directory.
5. Search for EDIT.RUN in the working directory.
6. 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 that is dormant.
2. Search for a permanently restored, busy, and clonable EDIT.
3. Search for EDIT in the proto-ID list.
4. Search for EDIT in FMGR disk cartridges.
5. Search for EDIT.RUN in directory PROGRAMS.

Parameters passed to the program can be integer, octal, 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. Specify octal numbers by immediately following the number with the letter b; for example, 30b.

Refer to the *RTE-A Programmer's Reference Manual* for more information about parameter passing and parsing.

Base Set Version of the RU Command

The RU and XQ commands are base set commands, meaning they may be entered at the "`System>`" or "`RTE:`" prompt, and may be executed programmatically using the MESSS routine. However, the base set versions of these commands differ substantially from the RU and XQ commands provided by CI, which may be executed programmatically using the FmpRunProgram routine.

The syntax of the base set commands is:

`RU,`*program*`[/`*session*`],`*parameters*
`XQ,`*program*`[/`*session*`],`*parameters*

Both commands schedule the named program without waiting for the program to complete, that is, they both behave like the XQ command.

The program to be scheduled must already be RP'd; the base set commands do not RP type 6 files as do the CI versions.

A session number may be specified to schedule programs in sessions other than your own or in the system session if you have sufficient capability. For example, "`xq ucsf/1`" runs program UCSF in session 1.

SAM (Show the Status of System Available Memory)

Purpose: Displays information about the status of System Available Memory. If the system contains Extended System Available Memory (XSAM), SAM reports on it as well.

Syntax: `SAM [AL] [XS] [`*lu*`|QU]`

`AL`	specifies a full listing. Using SAM with and without AL is described in detail below.
`XS`	returns information about XSAM in the return parameters, as described in the section called "Returned Values" below.
lu	is the LU to list output to (the default is 1). To specify LU 0, use the QU option.
`QU`	specifies quiet mode, and causes output to go to the bit bucket (LU 0).

Running SAM without the AL Parameter

When you run SAM without the AL parameter, the utility lists the total amount of memory, the number of free words, the number of free blocks, the average free block size, the largest free block size, and the percentage of total memory that is free space.

If the utility detects any missing or overlapping blocks, it reports the octal address and decimal length of each problem area along with a summary of the number of words missing and overlapping. This process is done once for SAM and if the system contains XSAM, once for XSAM.

A sample of output when a block of SAM is missing from a system without XSAM follows:

```
 SAM Analysis
   12       19  ** Unknown Block **
   71       88  ** Unknown Block **

32615 words free of 32763 total (99%)
1 blocks free; largest is 32615 words; average size 32615 words

** 2 errors: 107 unknown words

  XSAM Analysis

4021 words free of 4093 total (98%)
1 blocks free; largest is 4021 words; average size 4021 words
```

Running SAM with the AL Parameter

When run with AL, SAM prints a complete listing of memory blocks; lists the octal address, decimal length, and usage; reports missing or overlapping blocks, and prints summary information. If the system contains XSAM, this is also done for XSAM. A sample of the output follows.

```
   SAM  Analysis
   Octal  Decimal
   Address Length     Type

     3     4    Spooling for LU 1 from LU 8 to LU 2
     7     4    Spooling for LU 1 from LU 6
    13    13    String Passage for SAA
    30     7    String Passage for MUSTR
    37    17    Completed Class I/O for MUSTR on class # 32
    60     8    String Passage for SAM
    70    21    Free Space
   115   144    Pending Class I/O on LU 105
   335 32545    Free Space

 32566 words free of 32763 total (99%)
 2 blocks free; largest is 32545 words; average size 16283 words

    XSAM Analysis
   Octal  Decimal
  Address  Length    Type

     3    20    UDSP for session 158
    27    52    UDSP for session   1
   113    52    UDSP for session 105
   177    48    Signal Control Block for PROG
   257     4    Timer Block for PROG
   263  3917    Free Space

 3917 words free of 4093 total (95%)
 1 block free; largest is 3917 words; average size 3917 words
```

Returned Values

The SAM utility returns status information to the calling program through five return parameters, as follows:

$RETURN1 = # of errors (zero if none)
$RETURN2 = Total number of words of SAM
$RETURN3 = Total number of free words
$RETURN4 = Total number of free blocks
$RETURN5 = Size of the largest free block in words

The XS option causes SAM to return information about XSAM in the five return parameters. If XS is not specified, information about SAM is returned. The parameters have the following meaning:

$RETURN1 = Number of unknown or overlapping blocks found (0 if none)
$RETURN2 = Total number of words of memory
$RETURN3 = Number of free words
$RETURN4 = Number of free blocks
$RETURN5 = Size of the largest free block in words

Loading SAM

To load SAM, use the link sequence:

```
CI> link,sam.lod
```

SCOM (File Comparison)

Purpose: compares two input files and identifies their differences by matching sequences of lines and flagging those lines that are unique to a single file. By specifying options, you can configure a listing of only those lines unique either to one or to both files, or only those lines that are common to both, with or without line numbers.

Syntax: `SCOM` *file1 file2* `[[+|~]`*listfl*`]` `[`*options*`]` `[`*rematchlns*`]` `[`*maxchars*`]` `[`*difflimit*`]`

file1
file2 the files to compare. Type 1 or 6 files force a binary block compare (see the discussion of the "bb" option, below); type 2 or 5 files assume a binary record compare (but see the discussion of the "tc" option, below).

listfl is the listing file. The default is the terminal. The entry +*listfl* appends to an existing file, and ~*listfl* overlays an existing file.

options is one or more of the following (additional information about some of the options is presented in the following sections):

`F1`	Report lines unique to *file1* (default)
`F2`	Report lines unique to *file2* (default)
`BO`	Report lines common to both files
`NN`	Suppress line numbers on report
`NH`	No heading on listing (parameter information)
`NT`	No trailer on listing (number of mismatches)
`TB`	Trailing blanks are significant
`IB`	Trailing blanks are insignificant
`D`*x*	Wildcard; *x* matches any character in the other file
`C`*x*	Ignore lines with *x* in column 1 but otherwise blank when rematching
`IT`	Ignore Edit/1000 time stamps
`IC`	Ignore compile times in relocatable records for binary record compares
`ET`	Create an Edit/1000 transfer file to convert *file1* to *file2*
`ER`	Same as ET but add an Edit "ER" command to the end of the transfer file
`BR`	Binary record-by-record compare
`BB`	Binary block-by-block compare
`TC`	Force text compare on binary files

rematchlns indicates the number of consecutive lines that must match before a mismatch is ended; that is, the files are considered to be "rematched". The default is 3 lines.

maxchars is the maximum number of characters per record in both *file1* and *file2*. For hierarchical files, this is automatically set to the greater of the two maximum record lengths as given in the directory. For FMGR file system files, it is defaulted to 256; for binary block (bb) compares, it is forced to 256. The maximum value is 1024.

difflimit is the maximum number of differences to report. If the limit is reached, the following message appears, and the comparison terminates as if complete:

```
* Mismatch limit reached
```

If *difflimit* is not specified, all the differences found are reported.

SCOM can be run programmatically or from your terminal.

F1, F2, BO

If none of these options is specified, options F1 and F2 are used by default for the standard differences report.

NN

This option suppresses line numbers on report.

NH

When this option is specified, no heading (parameter information) appears on the listing.

NT

When this option is specified, no trailing information (number of mismatches) appears on the listing.

TB

This option is the default for the ET, ER, BB, and BR options. You can specify IB to override TB in ET, ER, BB, and BR.

IB

When this option is specified, trailing blanks are ignored when matching occurs. This is the default for the normal report (ET, ER, BB, and BR not given).

D*x*

The D*x* option allows you to specify a wildcard character that matches any character in the comparison file that is in the same line position as the wildcard. This can be any ASCII character, including a blank, except a comma, which is used as a runstring parameter delimiter. This option is useful in creating a mask file for comparison, filling the wildcard fields with the D option character.

As an example, entering d~ causes all tildes (~) in one file always to match the corresponding character in the other file, even if it is past the end of the other line.

C*x*

The C*x* option forces SCOM to ignore specified lines in searching for a rematch of the files. If an otherwise blank line has the C option character in column 1, the line is not considered in the search for a rematch. This option is useful for ignoring blank comment lines in a file.

x is typically c or * for FORTRAN or Macro source files. It may also be a blank, forcing SCOM to ignore totally blank lines in a file. For example, c* causes blank lines with a star (*) in column 1 to be ignored when rematching after a mismatch.

IT

If both files contain an EDIT/1000 timestamp of the form <*nnnnnn.nnnn*>, where *n* = 0..9, different times do not cause a mismatch; in other words, the *n* values can differ and still "match".

IC

XNAM checksums, compile times, language processor revision codes, and comment strings and XEND checksums are forced to be equal when you specify this option.

ET, ER

When these options are specified, the listing file is an EDIT/1000 transfer file that contains the necessary insert, replace, and kill commands to make file1 identical to file2. For example:

```
CI> scom ver2 ver1 ver2_1.edit er
```

The above command creates an EDIT/1000 transfer file named VER2_1.EDIT that contains the EDIT commands needed to modify file VER2 to the same text as VER1. To accomplish this, enter:

```
CI> edit –b ver2 tr ver2_1.edit
```

The F1, F2, BO, NN, D*x*, BR, BB, and IT options are not meaningful for this mode. If the files are binary, you must specify the TC option, otherwise an error occurs.

BR, BB

In a binary record compare, each record of file1 is compared, record-by-record, against the corresponding numbered record in file2, and differences are reported in octal format. This is assumed if either file is type 2 or type 5.

In a binary block compare, each block of file1 is compared, block-by-block, against the corresponding numbered block in file2, and differences are reported in octal format. This is forced if either file is type 1 or type 6.

For both the BR and BB comparisons, mismatching blocks or records are dumped in 12-byte groups, side-by-side in both octal and ASCII (if standard printing character) format. The octal representation for file2 bytes appears as "..." if this byte is the same as the byte for file1 listed

directly above. If only blanks appear in that field, the byte position is past the end of the record for that file. The last byte in each file is suffixed with a backslash (\). For example:

```
Record 12
  1    1:  377  377  000  000  000  000  000  000  000  000  000  000  000 |
 13     :  000  132  004  320  000  000  000  000  000  \                    | Z
       13:  ...  143  ...  000  ...  ...  \                                  | c
```

This says that in record 12 of both files, bytes 1-13, 15, and 17-18 match. Bytes 14 and 16 mismatch. Bytes 19-21 in file1 are past the end of the file2 record. Byte 14 is octal 132, ASCII "Z" in file1; that byte is octal 143, ASCII "c" in file2.

TC

This option forces a text compare on binary files.

The Compare Operation

SCOM compares the two input files and, when a mismatch is found, begins searching for a rematch. The files are considered rematched only when the number of lines (including blank lines) specified in *rematchlns* are identical.

Invalid options in the runstring are ignored, and SCOM executes using only the valid options. Any entry in the nmatch or nchar parameters other than a positive integer is ignored, and SCOM executes using the default values. SCOM does not issue a status message if any options are ignored; the run state is defined in the header of the comparison display.

Returned Values

Numeric parameters returned on exit are as follows:

$RETURN1 = 0 If comparison is complete, this is the number of differences found. 0 means the files are identical.

$RETURN2 = 1 if records were truncated, 0 if not

$RETURN1 = −1 If an FMP error occurred.

$RETURN2 = FMP error code

$RETURN3 = the number of the file on which the error occurred: 1 for file1, 2 for file2, 3 for listfile

$RETURN1 = −2 If SCOM error occurred.

$RETURN2 = 1 for runstring errors, others for internal errors

Status Interrogation

When SCOM executes a file compare, you may request the current status of the compare by entering break mode (striking any key on the keyboard) and using the BR command. SCOM responds with one of the following messages, followed by the "More ... " prompt:

```
Files presently match at file1 line XX and file2 line YY
ZZZ subsequent matching lines in each file compared
```

or

```
Files presently mismatch at file1 line XX and file2 line YY
ZZZ subsequent lines compare in each file without rematch
```

and

```
More...
```

SCOM Examples

Example 1:

```
CI> scom,test1,test2
Scom: comparison performed Sat Nov 21, 1992  4:32:17 pm
   file1 is TEST1
   file2 is TEST2
   options: f1f2;  rematchlns: 3;  maxchars: 36
-------------------------------------------------------------------------
-- beginning of file --
1           FTN,L
      1     FTN^^
2     2           PROGRAM FTEST
-------------------------------------------------------------------------
5     5           COMMON /MISC/ LUT,W1,W2,WBOTH
6                 LUT = 1
7     6           WRITE (1,10)
-------------------------------------------------------------------------
16    15    C
17                END
18          $
      16          END$
-- end of file --

3 differences were found.
CI>
```

In the above example, all defaults were taken. Column one represents file1 line numbers, and column two represents file2 line numbers. When the line contents differ, the line number appears in the appropriate file column, and the areas of difference are bracketed with dashed lines. By default, the F1 and F2 options are in effect, and only the differing lines are displayed. Around each area of difference, SCOM adds one preceding and one following line that are common to both files.

Example 2:

```
CI> scom,test1,test2,1,nn
Scom: comparison performed Sat Nov 21, 1992  2:38:58 pm
   file1 is TEST1
   file2 is TEST2
   options: f1f2nn;  rematchlns: 3;  maxchars: 36
-- beginning of file --
-----------------------------  file1 only  --------------------------
FTN,L
-----------------------------  file2 only  --------------------------
FTN^^
-------------------------------------------------------------------
      PROGRAM FTEST
      COMMON /MISC/ LUT,W1,W2,WBOTH
-----------------------------  file1 only  --------------------------
      LUT = 1
-------------------------------------------------------------------
      WRITE (1,10)
C
-----------------------------  file1 only  --------------------------
      END
$
-----------------------------  file2 only  --------------------------
      END$
-------------------------------------------------------------------
-- end of file --

3 differences were found.
CI>
```

In this example, the NN option suppresses line numbering. In this case, the areas that contain lines that differ are identified by the dashed line headers “file1 only” or “file2 only”. As in the previous example, SCOM adds the preceding and following lines that are common to both files around each area of difference.

Example 3:

```
CI>  scom,test1,test2,,f1f2bo
Scom: comparison performed Sat Nov 21, 1992  4:33:16 pm
   file1 is TEST1
   file2 is TEST2
   options: f1f2bo;  rematchlns: 3;  maxchars: 36
1           FTN,L
      1     FTN^^
-------------------------------------------------------------------------
2     2           PROGRAM FTEST
3     3           IMPLICIT INTEGER (A-Z)
4     4           LOGICAL W1,W2,WBOTH
5     5           COMMON /MISC/ LUT,W1,W2,WBOTH
6                 LUT = 1
-------------------------------------------------------------------------
7     6           WRITE (1,10)
8     7     10    FORMAT(”N = ”’)
9     8           READ(1,*) N
10    9           CALL ERROR(N)
11    10          END
12    11          BLOCK DATA MISC
13    12    C
14    13          LOGICAL W1,W2,WBOTH
15    14          COMMON /MISC/ LUT,W1,W2,WBOTH
16    15    C
17                END
18          $
      16          END$
-------------------------------------------------------------------------

3 differences were found.
CI>
```

In this example, the BO option is used to list all lines that are common to both files. When BO is specified, F1 and F2 are no longer defaulted and they must be specified as well to include the lines that differ.

Example 4:

```
CI>  scom,test1,test2,,nnf1f2bo
Scom: comparison performed Sat Nov 21, 1992  4:34:04 pm
   file1 is TEST1
   file2 is TEST2
   options: f1f2bonn;  rematchlns: 3;  maxchars: 36
------------------------------  file1 only  --------------------------
FTN,L
------------------------------  file2 only  --------------------------
FTN^^
-------------------------------------------------------------------------
      PROGRAM FTEST
      IMPLICIT INTEGER (A-Z)
      LOGICAL W1,W2,WBOTH
      COMMON /MISC/ LUT,W1,W2,WBOTH
------------------------------  file1 only  --------------------------
      LUT = 1
-------------------------------------------------------------------------
      WRITE (1,10)
10    FORMAT(”N = ”’)
      READ(1,*) N
      CALL ERROR(N)
      END
      BLOCK DATA MISC
C
      LOGICAL W1,W2,WBOTH
      COMMON /MISC/ LUT,W1,W2,WBOTH
C
------------------------------  file1 only  --------------------------
      END
$
------------------------------  file2 only  --------------------------
      END$
-------------------------------------------------------------------------

3 differences were found.
CI>
```

This example shows the NN and BO options together.

Example 5:

```
CI>  scom,test1,test2,,d^f1f2bo
Scom: comparison performed Sat Nov 21, 1992  4:35:42 pm
   file1 is TEST1
   file2 is TEST2
   options: f1f2bod^;  rematchlns: 3;  maxchars: 36
1     1     FTN,L
2     2           PROGRAM FTEST
3     3           IMPLICIT INTEGER (A-Z)
4     4           LOGICAL W1,W2,WBOTH
5     5           COMMON /MISC/ LUT,W1,W2,WBOTH
6                 LUT = 1
-------------------------------------------------------------------------
7     6           WRITE (1,10)
8     7     10    FORMAT(”N = ”’)
9     8           READ(1,*) N
10    9           CALL ERROR(N)
11    10          END
12    11          BLOCK DATA MISC
13    12    C
14    13          LOGICAL W1,W2,WBOTH
15    14          COMMON /MISC/ LUT,W1,W2,WBOTH
16    15    C
17                END
18          $
      16          END$
-------------------------------------------------------------------------

2 differences were found.
CI>
```

In this example, the ^ is specified as a wildcard character; thus, the first lines of the two programs now match. Note that the text of matching lines is always read from file1.

Example 6:

```
CI>  scom,test3,test4,,f1f2bo,1
Scom: comparison performed Sat Nov 21, 1992  4:42:45 pm
   file1 is TEST3
   file2 is TEST4
   options: f1f2bo;  rematchlns: 1;  maxchars: 16
1     1     ASMB,R,L,C
2     2           NAM TEST3
3     3     *
4           B     EQU 1
      4     A     EQU 0
-------------------------------------------------------------------------
5     5     *
      6     B     EQU 1
      7     *
-------------------------------------------------------------------------
6     8     START  HLT
7     9           END START

2 differences were found.
CI>
```

In this example, two similar files are compared with “rematchlns” set to 1 line, for rematch of identical lines. The difference between the files is the addition of two instructions (lines 4 and 5) to file TEST4:

```
4   *
5   A    EQU 0
```

In this case, when SCOM compares the files it does not detect the difference, since it finds that line 5 in each file matches. Because of this, SCOM cannot detect the more preferable rematch of line 4 in TEST3 and line 6 in TEST4.

This problem of extraneous identical lines causing erroneous rematches occurs most often with blank comment lines. Example 7 demonstrates a method of avoiding this problem.

Example 7:

```
CI>  scom,test3,test4,,c*f1f2bo,1
Scom: comparison performed Sat Nov 21, 1992  4:44:10 pm
   file1 is TEST3
   file2 is TEST4
   options: f1f2boc*;  rematchlns: 1;  maxchars: 16
1     1     ASMB,R,L,C
2     2           NAM TEST3
3     3     *
      4     A     EQU 0
      5     *
-------------------------------------------------------------------------
4     6     B     EQU 1
5     7     *
6     8     START  HLT
7     9           END START

1 difference was found.
CI>
```

In this example, the C* option specifies that all otherwise blank lines with * in column 1 are ignored when searching for a rematch in the files. The result is a more desirable rematch after the mismatch at line 4 in each file.

SCOM Error Messages

Error messages are only displayed at the log terminal; they are not included in the output file. Fatal errors that cause SCOM to abort are identified with (F) following the error description.

WARNING: RECORD TOO LONG

The program read a record whose length is greater than that specified by the MAXCHARS parameter. The record is truncated and the comparison may be invalid. This is only a warning and does not abort SCOM.

#1. USER SUPPLIED MAXIMUM RECORD SIZE IS TOO LARGE

The internal buffer space is insufficient to accommodate the specified MAXCHARS. Rerun SCOM and specify a smaller MAXCHARS value. (F)

#2. ILLEGAL INTERNAL SUBROUTINE PARAMETER

The subroutine was called with an invalid parameter; a utility self-check has failed. Reload SCOM and rerun. (F)

#3. CACHE DATA STRUCTURE CORRUPT

The internal check of the buffer data structure shows corruption; a utility self-check has failed. Reload SCOM and rerun. (F)

SET (Display/Define Variables)

Purpose: Displays all positional, user-defined, and predefined variables, or defines a user-defined or predefined variable.

Syntax:
```
SET [variable_name [= string_value]]
SET [-x|+x] [variable_name [[=] string_value]]   (VC+ only)
```

`-x`	if the user session has an Environment Variable Block, this option "exports" a defined variable or defines a new one as exported (also known as being in the environment). An exported variable is available to all copies of CI in the session, including CM.
`+x`	imports an exported variable or defines a new variable as imported (also known as being a local variable). An exported variable is defined only for the current CI. If neither the −x or +x option is given, the variable is defined as a local variable.
variable_name	is a string of letters, digits, and underscores, not starting with a digit. The maximum string size is 16 for non-VC+ systems and 32 for VC+ systems.
string_value	is a string terminated by the end of the command line or a ";" whichever comes first. The command line can be 255 characters. For non-VC+, the maximum length of *string_value* is 83 characters.

Description:

CI provides variables that you can define (positional and predefined variables) and also allows you to create variables (user-defined variables). The SET command displays or defines these variables.

If a variable is not specified, all positional, user-defined, and predefined variables are displayed. When displayed, the local variables are displayed first, followed by a blank line, then the exported variables are displayed.

Quoting preserves lowercase letters and spaces.

Variables entered into the EVB programmatically can have characters not allowed by CI. The SET command displays them even though CI cannot use them.

Examples:

```
CI> set filename = /scott/sam.ftn
                                   (Define a (local) variable called FILENAME)

CI> set auto_logoff=3              (Redefine the variable AUTO_LOGOFF)

CI> set back down                  (Define a local variable BACK)

CI> set greeting = 'How are you today?'
                                   (Define a variable GREETING that has
                                    lowercase letters and blanks)
```

```
CI> set –x visual emacs
```
(Define and export the variable VISUAL; does not affect the local copy, if any)

```
CI> set –x log
```
(Export the previously defined variable LOG; delete the local copy)

```
CI> set +x pect=ations
```
(Define a local variable PECT)

```
CI> set +x mine
```
(Import the variable MINE from the EVB and make it a local variable; assuming MINE was previously exported)

```
CI> set blank
CI> set –x blank
```
(Define a local variable BLANK to be an empty string, then export BLANK; note that this two-step process is the only way to define an exported empty string variable)

```
CI> set
```
(Show defined variables and their values)

```
CI> set +x
```
(Show local variables and their values)

```
CI> set –x
```
(Show exported variables and their values)

SPORT (Serial Port Analyzer)

Purpose: Displays the status of serial ports, which are driven by DDC00 in RTE-A and by DVC00 or DV800 in RTE-6/VM. This program verifies that a port was set up in the desired manner, and is useful for troubleshooting problems involving serial ports.

Syntax: `SPORT [`*port_lu* `[`*display_lu* `[`*waitflag*`]]]`

port_lu is the serial port LU for which the current status is requested. The default is the session LU.

display_lu is the LU to which the output is directed. Output can be directed to another LU, but not to a file. The default is the session LU.

waitflag allows you to specify a non-zero value to force a "wait" for a busy port; in other words, you can have SPORT "hang" on the port until the current request completes.

Description:

All parameters must be entered in the runstring, as the program is non-interactive. A help screen is displayed when you enter "`sport ?[?]`".

The default operation of SPORT is to first check the port LU to see if it is busy, down, or locked. If it is not, the program issues a Special Status Read (described in the *RTE-A Driver Reference Manual*) to obtain the current status of the port.

If the port is not available, SPORT invokes a subroutine called FakeSpStatus to read as much of the status as is available in the system tables. Because SPORT is used as a troubleshooting tool, this is usually the most desirable mode of operation. For the occasions when you want SPORT to "hang" on the port until the current request completes, you can set the *waitflag* parameter to any non-zero value.

Examples:

When the status is obtained from a special status read, the program formats the result into a display similar to the following example:

```
CI> ru,sport,31
Status for LU 31:                       930712.17563942
Device Driver:     DDC01 Rev. 6100    Driver type = 05
Interface Driver:  ID800 Rev. 6100
  Firmware:               Rev. 5.22
 CN20:  Primary Program:   PROMT (Enabled)
 CN40:  Secondary Program: HPMDM (Disabled)
 CN17:  000000B No user defined terminator
 CN22:   32767  Timeout = 327.67 seconds
 CN16:  000005B BRG0 range 5 (4800 9600 19200)
 CN30:  000330B Frame=8/1 None BRG0 9600 baud Port 0 SS at Logon
 CN31:  000000B
 CN33:  000000B
 CN34:  000002B HP Protocol
 DV20:  000077B Character mode
 DVT Address:  44043B; IFT Address:  53233B
```

When the status is obtained from a call to FakeSpStatus, the display is similar to the following example:

```
CI> ru,sport,32
IFT of LU 32 is busy: Static status report
Status for LU 32:                      930712.17563942
Device Driver:     DDC0? Rev. 255.255    Driver type = 05
Interface Driver:  ID80? Rev. 255.255
  Firmware:               Rev. 255.255
 CN20:  Primary Program:   PROMT (Enabled)
 CN40:  Secondary Program: HPMDM (Disabled)
 CN17:  000000B No user defined terminator
 CN22:   32767  Timeout = 327.67 seconds
 CN16:  000005B BRG0 range 5 (4800 9600 19200)
 CN30:  000330B Frame=8/1 None BRG0 9600 baud Port 0 SS at Logon
 CN31:  000000B
 CN33:  000000B
 CN34:  000002B HP Protocol
 DV20:  000077B Character mode
 Driver status bits set:  TimeOut
 DVT Address:  44227B; IFT Address:  53233B
```

On the first line of the display, the words “Static status report” are shown in half-intensity flashing video (on terminals so equipped) as a warning that this is synthesized information. Note that many of the fields now contain “odd values”, because the information is not available from the system tables.

Including SPORT in a User Program

The SPORT program functionality can be included in a user program by calling HpZPrintPort. For example:

```
CALL HpZPrintPort(port_lu,display_lu,waitflag)

integer*2 port_lu,display_lu,waitflag
```

where:

port_lu is the LU number of the port to display (in the range 1..255) in the format of the first word of the XLUEX control word.

display_lu is the LU number of the display LU (in the range 1..255) in the format of the first word of the XLUEX control word. If it is not in the valid range, it will default to the call LU.

waitflag specifies whether to get real or “fake” status in the event that *port_lu* is busy. If *waitflag* is 0, the true status is displayed. If *waitflag* is nonzero, a call to FakeSpStatus is made.

This subroutine prints out the port status (using a special status read) of *port_lu* to *display_lu*. If *port_lu* = 1, it is converted to the true LU number.

The subroutine verifies that the LU is not locked, not down, and not busy. Otherwise, a message is sent to the calling LU.

This is designed to be a simple addition to a program such as the following:

```
Ftn7x,Q,S
     program HpCrt
     integer prams(5),obuf(0:79)
     call rm par(prams)
     call HpZDefObuf(Obuf)
     call HpZPrintPort(prams(1),prams(2),prams(3))
     end
```

Loading SPORT

To load SPORT, invoke LINK on the SPORT.LOD command file. The only relocatable needed is SPORT.REL and the assumed library is $BIGLB.

SS (Suspend Program)*

Purpose: Suspends an active program.

Syntax: `SS [`*program*`]`

program is the 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 queued until the program is rescheduled. At that time the program is placed in the operator suspend state.

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.

If prog is not specified and the startup program (usually CI) 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 that if the last program is a protected system program, the program that scheduled it will be suspended.

Example:

```
CI> ss timer                    (Suspend program TIMER)
```

SYSTZ (Set Time Zone and Daylight Savings Time)

Purpose: Displays or sets the system's time-zone offset and the daylight savings time flag.

Syntax: `SYSTZ [-DSQ] [ [+|-]`*hours*`[:`*minutes*`] ]`

`-D`	Daylight savings time (DST) is in effect. This flag is ignored if the offset is not given.
`-S`	DST is not in effect. This flag is ignored if the offset is not given. This is the default setting if neither −S nor −D is given.
`-Q`	Quiet. Do not output the settings.

Description:

The RTE system clock maintains time relative to the local time zone. The offset of the local time from Coordinated Universal Time (UTC) is needed by RTE programs that must communicate time-of-day information with remote systems that store their time relative to UTC. In addition, they may need to know if daylight savings time (DST) is currently in effect.

SYSTZ sets the system's time-zone offset and DST flag when it is given an offset period. The offset is specified as the number of hours, and, optionally, minutes, that the current system time is offset from UTC. Typically, locales east of the prime meridian will specify the offset as a negative number, and locales west of the prime meridian will specify a positive offset. The offset can range from −12 hours to 12 hours.

By default, when the system's time-zone offset is set, SYSTZ clears the DST flag indicating that DST is not in effect. The −S (standard time) flag can be used to explicitly state that the DST flag should be cleared. The DST flag can be set by specifying the −D option when setting the system's time-zone offset. The setting of the DST flag does not interact with the setting of the offset.

SYSTZ only updates the offset and DST flag; it does not adjust the actual system time. SYSTZ must be performed whenever local time passes in or out of daylight savings time. You must also modify the system time using the TM command to reflect this change.

SYSTZ reports either the current settings, if an offset is not specified, or the new settings, if an offset is given. This can be overridden by using the −Q (quiet) flag.

Return values:

$RETURN1 =	0	command executed successfully. $RETURN2, $RETURN3, and $RETURN_S are valid.
	1	the time zone has not yet been set. SYSTZ needs to be run with an offset specified.
	2	error in the runstring. The usage message will be output.
	3	you must be a superuser to set the time zone.
$RETURN2 =	The offset from UTC in minutes.	
$RETURN3 =	0	DST is not in effect.
	1	DST is in effect.
$RETURN_S =	The reported settings in a format suitable for use as the runstring for a future invocation of SYSTZ.	

Examples:

```
CI> * Settings for Pacific Standard Time
CI> systz 8
Offset from Coordinated Universal Time is +08:00 (hours:minutes).
Daylight Savings Time is not in effect.
CI> echo $return1 $return2 $return3 $return_s
0,480,0,–S +08:00

CI> * Settings for Pacific Daylight Savings Time
CI> systz –d 7
Offset from Coordinated Universal Time is +07:00 (hours:minutes).
Daylight Savings Time is in effect.
CI> echo $return1 $return2 $return3 $return_s
0,420,1,–D +07:00
```

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 is the program name, up to five characters, with an optional session identifier.

size is the program size in pages for non-EMA/VMA programs or the EMA size for EMA programs.

When modifying the program size of a non-EMA/VMA program, the range is:

minimum required by program ≤ *size* ≤ 32

When modifying the EMA size of an EMA program, the range is:

2 ≤ *size* ≤ 1022 for Normal and Large model EMA programs

2 ≤ *size* ≤ 32733 for Extended model EMA programs.

Refer to "Models of EMA/VMA" in Chapter 4.

msegSize is the new MSEG size for EMA programs. The range is:

1 ≤ *msegSize* ≤ 30.

Description:

This command changes the amount of memory that the specified program can use when it runs. The program must be restored with the RP command and must be dormant.

Increasing the program size will help programs that use memory at the end of their partition for buffer or table space. Such programs include EDIT, LINK, and MACRO. To change the size permanently, use the LINK program.

The SZ command does not apply to CDS programs in the VC+ environment. For CDS programs, use the CZ or DT command for code or data, respectively. The SZ command also does not apply to VMA programs; use the VS or WS command to modify the VMA or working set size, respectively.

TM (Display or Set System Clock)*

Purpose: Displays or sets the system time. Note that the CI and base set versions of this command are different. See the following section "Base Set Version of the TM Command" for information on the base set version.

Syntax: `TM [`*month day year* `[`*hr*`[:`*min*`[:`*sec*`[AM|PM]]]]`

month	Jan to Dec
day	1 to 31
year	1976 to 2037
hr	0 (default) to 23
min	0 (default) to 59
sec	0 (default) to 59
`AM|PM`	defaults to the value appropriate for a 24-hour clock; AM if *hr* is a value from 0 to 11, or PM if *hr* is a value from 12 to 23.

Description:

This command displays the system clock in the format shown in the following example:

```
Mon Sep 27, 1988 11:37:13 am
```

Parameters can be entered as they would be printed; time can be specified in a 24-hour format if desired, with or without seconds.

Resetting the time affects programs scheduled with the AT command 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.

In the VC+ environment, only a superuser can set the system clock.

Base Set Version of the TM Command

The TM command is a base set command, meaning it may be entered at the "`System>`" or "`RTE:`" prompt, and may be executed programmatically using the MESSS routine.

The syntax for the base set version of TM is as follows:

`TM [`*hours min sec month day year*`]`

hours	0 (default) to 23
min	0 (default) to 59
sec	0 (default) to 59
month	1 to 12
day	1 to 31
year	1976 to 2037

The system clock is displayed in the following format:

```
11:27:13 SEP 27, 1988 MON
```

Time can only be specified in a 24-hour format. All parameters must be specified when setting the system time. Setting the system time has the same affect as when it is done from CI.

TO (Display or Modify Device Timeout)

Purpose: Displays or sets the timeout limit for a device.

Syntax: `TO` *lu* [*interval*]

lu is the LU number of the device.

interval is the number of 10-milliseconds intervals to be used as the timeout value for the device LU. The value can be in the range of $0 \leq interval \leq 65534$.

If *interval* equals 0, the timeout limit does not apply to this device.

Description:

The timeout value is displayed in the form:

```
Timeout for LU          # = interval
```

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 timeout value defined, the program is put into a downed device wait state and the device may be set down. This prevents an offline or downed 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 disk 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.

Default time-out values are contained in the driver code for each device. You can find these values in the *RTE-A System Generation and Installation Manual*. Some device drivers ignore or override the operator-specified time-out value.

If you set a timeout 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 downed.

To calculate the interval parameter, multiply the desired timeout value (in seconds) by 100.

When the system is rebooted, timeout values revert to those set at system generation time.

Examples:

To display the timeout value:

```
CI> to 6
Timeout for LU 6 = 500          (Set 5-second timeout)
```

To modify LU 6 timeout to 10 seconds:

```
CI> to 6 1000
Timeout for LU 6 = 1000         (New timeout value is displayed)
```

TOUCH (Update File Times; VC+ Only)

Purpose: Updates the create, access, or update time in a file's directory.

Syntax: `TOUCH [-amcb] [+b] [-r` *ref_file*`] [-t` *time*`] [--]` *file*`|`*mask* ...

`-a` update only the access time of the file.

`-b` clear the backup bit of the file.

`+b` set the backup bit of the file.

`-c` do not create the file if the file does not exist.

`-m` update only the create time of the file.

`-r` *ref_file* use the corresponding create, access, and/or update time of *ref_file* instead of the current time.

`-t` *time* use the *time* argument as the create, access, and/or update time rather than the current time. The *time* argument is a decimal number in the following form:

`[[`*CC*`]`*YY*`]`*MMDDhhmm*`[.`*SS*`]`

where:

CC	is the first two digits of the year.
YY	is the second two digits of the year.
MM	is the month in the range 01 to 12.
DD	is the day in the range 01 to 31.
hh	is the hour in the range 00 to 23.
mm	is the minute in the range 00 to 59.
SS	is the second in the range 00 to 59.

If neither *CC* nor *YY* is given, the current year is used. If *YY* is specified, but *CC* is not, *CC* is determined as follows:

If *YY* is 69−99 then *CC* = 19
If *YY* is 00−68 then *CC* = 20

`--` signals the end of the options. This is only required if the first mask specified after the options begins with a hyphen (−).

file`|`*mask* ... the file descriptor of the file whose create, access, or update time is to be changed.

Description:

By default, TOUCH updates the access time and update time of a file to the current system time. The status of the backup bit of the file is not altered unless the −b option is used. If a file does not exist, the file is created. The −c option can be used to prevent the creation of files.

The −t option can be used to set the access, create, or update time to a specific time.

The −r and −t options are mutually exclusive.

Return Value:

TOUCH returns the number of files for which the times could not be successfully modified (including files that did not exist and were not created) in $RETURN1.

TR (Transfer to Command File)

Purpose: Transfers control to a command file.

Syntax: [TR] *file* [*pram*9*]

TR	is 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	is the file containing the commands. Refer to the CR command for a definition of a file descriptor.
*pram*9*	is one to nine parameters that are used to replace occurrences of the positional 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 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 either to the terminal or to the command file, depending on whether the TR command was issued from the terminal or another command file, when the end of the file is reached or a RETURN command is encountered. A "TR,1" transfers control to the terminal and a RETURN command returns control to the file from which the "TR,1" command was issued. See the RETURN command description in this chapter.

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.

We recommend that you always use the .CMD file type extension in the command file name. This is required if you use a user-defined search path to find the file (see the descripton of the PATH command earlier in this chapter).

If you will be executing more command files than program files, 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 the working directory.
2. Search for MYCMD.CMD in the 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.

You can include comments in a command (transfer) file by placing an asterisk (*) in the first column of a line. When an asterisk (*) is the first character in a line, CI ignores the entire line.

Examples:

In the following example, COMP.CMD, a command file that compiles, links, restores, sizes, and runs program TEST4 and then removes its ID segment, is executed:

```
CI> 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> comp2 test4 28
```

where file COMP2.CMD contains the following commands:

```
* A command file to compile and link a program.
*
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; VC+ Only)*

Purpose: Unlocks a shareable EMA partition so that it can be used by other programs.

Syntax: `UL` *label*

label is a name that identifies a shareable EMA partition label, up to 16 characters.

Description:

The UL command is a useful tool when a program aborts or ends without unlocking the shareable EMA partition. The 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> ul d12
```

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

UNALIAS (Delete Alias; VC+ Only)

Purpose: Deletes one or more aliases from CI and/or the Environment Variable Block (EVB).

Syntax: `UNALIAS` *alias1*`[,`*alias2* ...`]`

alias1, *alias2* is one or more strings of up to 32 letters, digits, and underscores, not starting with a digit.

Description:

If the user session has an EVB, CI first looks for the local alias and deletes it. If there is no local alias with the given name, CI looks for an exported alias and deletes it.

Examples:

```
CI> unalias lin                    (Delete the alias LIN)

CI> unalias doc who                (Delete the aliases DOC and WHO)
```

UNPU (Unpurge Files)

Purpose: Recovers purged files.

Syntax: `UNPU` *mask*

mask is a file mask that specifies what files to unpurge. Refer to the "File Masks" section in Chapter 3 for a full description of file masks.

Description:

UNPU restores a purged file to active status. This can be done only if the directory entry of the specified file and the disk space of the file were not reclaimed by the file system. There are no guarantees as to how long these conditions may last. If there are multiple purged files with the same name, it is uncertain which one will be recovered. 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. Files recovered with the UNPU command retain the same attributes in effect when they were purged, including their time stamps.

FMGR files and directories cannot be unpurged.

For VC+ only, when the variable $QUIET_CMD is set to ON, the message

```
Unpurging FILE1 ... [ok]
```

is not displayed.

Examples:

The following command unpurges file CHARTER.TXT:

```
CI> unpu charter.txt
Unpurging CHARTER.TXT ...  [ok]
```

The following command attempts to unpurge AREA.FTN but is unsuccessful:

```
CI> unpu area.ftn
Unpurging AREA.FTN ...  [failed]
No such file AREA.FTN
```

UNSET (Delete User-Defined Variable)

Purpose: Deletes one or more CI string variables or functions from CI and/or the Environment Variable Block (EVB).

Syntax:
```
UNSET variable
UNSET variable1 [variable2 ...]        (VC+ only)
UNSET -f function1 [function2 ...]     (VC+ only)
```

variableN, *functionN* — A string of up to 32 letters, digits, and underscores, not starting with a digit. The variable or function must exist.

`-f` — specifies a CI function rather than a variable.

Description:

The UNSET command deletes a variable or function that you defined earlier in the session. Deleting unneeded, user-defined variables or functions frees space that CI can use for defining other variables. You cannot use the UNSET command to delete positional and predefined variables except for $DATC, $IFDVR, $LINES, $COLUMNS, and $EVB_SIZE.

If the session has an EVB, CI first looks for the local variable or function and deletes it. If there is no local version, CI looks for an exported one and deletes it.

Examples:

```
CI> unset ling                    (Unset the variable LING)

CI> unset ling times              (Unset the variables LING and TIMES)

CI> unset -f flink                (Unset the function FLINK)

CI> unset -f tardis space time    (Unset the functions TARDIS, SPACE, and
                                   TIME)
```

The following command attempts to delete the predefined variable $SESSION, which causes an error to occur:

```
CI> unset session
Cannot unset SESSION
```

UP (Up a Device)*

Purpose: Notifies the system that a specified device is available.

Syntax: `UP` *lu*

lu is the LU number of the device.

Description:

RTE-A downs a device when an error such as a timeout occurs. The LU remains unavailable until the UP command is given with that LU 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.

Example:

```
CI> up 8                                (Make LU 8 available)
```

VS (Display or Change VMA Size)*

Purpose: Displays the VMA size or changes the VMA size requirements of a restored program.

Syntax: `VS` *prog* [*vsSize*]

prog is the program name, up to five characters.

vsSize is the VMA size in pages. The range is 32 ≤ *vsSize* ≤ 65536.

Description:

The virtual space is the disk area used for paging data that does not fit in memory. Increasing this space may allow the program to process more data. Decreasing it cuts down on disk 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.

Examples:

```
CI> vs test4 200                          (Set VMA size to 200 pages)

CI> vs test4                              (Display VMA size of TEST4)
Last address= 20000 Min.Part.= 29 EMA/WS= 20 Mseg=1 VMA= 200
```

WC (Line, Word, Character Count)

Purpose: WC counts lines, words, and characters in the named file or files. It also keeps a total count for all named files. A word is a string of characters delimited by spaces or tabs.

Syntax: `WC [-`*options*`] `*file*`|`*mask* `...`

options is one or more of the following:

- `-l` Reports the number of lines in a file.
- `-w` Reports the number of words in a file.
- `-c` Reports the number of characters in a file.
- `-q` (Quiet) Inhibits error reporting.
- `--` Marks the end of the options. This is only required when the mask begins with a hyphen (−).

The l, w, and c options can be used in any combination to report a subset of lines, words, and characters. The default is −lwc.

file|*mask* is one or more files for which you want the line, word, or character count.

Description:

WC only reports information for variable record length files. An error is reported for file types 1, 2, or 6.

The character count for a file does not include the record length words before and after each record of an RTE variable record length file. The character count reported by WC does include an additional character for each line in a file. This is to account for the new-line character that is used as a record separator when files are transferred to HP-UX.

The output of WC can be redirected to an output file by specifying either >*filename* or >>*filename* in the runstring. The output file specified must be delimited by commas and is position independent. If the file already exists, it will be overwritten. To append to a file, >>*filename* can be used. If the file does not already exist, it will be created.

If an output file is not specified, WC breaks the output into screen pages. Paging is disabled when an output file is specified. Multiple redirection strings may occur in the runstring; however, only the last redirection is executed.

Return Values:

$RETURN1	= 0	WC executed successfully.
	> 0	The number of errors encountered.
$RETURN2		The number of files successfully read.
$RETURN_S		A string containing the count information for the named file. Or a string containing the total count information when more than one file is specified in the runstring.

The following two variables are only returned when a single output option is specified (−l, −w, or −c):

$RETURN3	The high order bits (31 through 16) of the line, word, or character count.
$RETURN4	The low order bits (15 through 0) of the line, word, or character count

Examples:

The following command string reports the number of lines and characters in the files matching the mask /HELP/WH@.

```
CI> wc -lc /help/wh@
352 15030 /help/wh
39 1615 /help/while
83 3238 /help/whosd
474 19883 total
```

The following command string counts the number of lines, words, and characters in the file FIELDS.

```
CI> wc fields
900 468 3825 fields
```

WD (Display or Change Working Directory)

Purpose: Displays or changes the working directory.

Syntax: `WD [`*directory* `[`*file*`|+S]]`

directory is the name of the new working directory. This may be a subdirectory name.

file is the command stack file descriptor associated with the name of the new working directory (.STK file type extension recommended). All subsequent posting of command stack contents will be to this file until another file is designated with another WD command. This must be a type 3 or type 4 file.

`+S` this parameter causes posting of the command stack contents either to the default file CI.STK or to a file previously specified. Then the command stack is reinitialized with the contents of CI.STK on the new working directory, or cleared if the file does not exist.

Description:

This command sets up the working directory that is 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.

In a non-VC+ environment, the WD command affects all users on the system. If one user changes the working directory, it is changed for all other users.

In the VC+ environment, the WD command changes the working directory associated with a session. It does not have to be owned by the user. The user does not have to have read or write access to the named directory, but setting the working directory to a read or write protected directory may cause programs (such as EDIT) problems when they try to create scratch files.

The second parameter provides the option of manipulating the command stack files. It lets you post the contents of the command stack to a file on a particular directory so that the same commands can subsequently be used. Refer to the Command Stack command description for examples of manipulating command stack files.

WH (System Status Reporting)

Purpose: Runs the system status program WH to report system information.

Syntax: `WH [-P] [`*option*`] [`*destlu*`]`

`-P` causes WH to generate "More..." prompts at the end of each screenful on long listings. This is useful when the terminal has insufficient screen memory to save the entire report. However, the system state might change significantly while WH is suspended at the prompt, causing the information displayed to become inaccurate or inconsistent.

option specifies the information to be displayed. If no option is entered, WH shows program information for this session or terminal. Options may be specified by entering a two-character abbreviation or a session number, in which case just that session is shown. Refer to NO TAG in the next section for a summary of WH options.

destlu specifies the LU to which the output is to be sent, if it is not the scheduling terminal.

Description:

A sample display of WH usage is shown below and further described in the following paragraphs.

```
CI> wh
Program                  DataPartition CodePartition
Name     Prio    PC    Seg  Size Status   Size Status  Program Status
-------------------------------------------------------------------
Session    44  User ELIZABETH
CI         51 14532          32 in                     waiting for WH
WH         5   5640          15 in          5          scheduled
-------------------------------------------------------------------
Sat Jan 25, 1992  12:10 pm
```

Note that WH executes while the state of the system is changing and, therefore, some reported information might be incomplete, inaccurate, or duplicated.

`Program Name` Programs are grouped by associated terminal LU or session number. With the default listing, only active programs associated with your terminal are listed. With the AC option, all active programs associated with all terminals or sessions are listed. With the AL or US option, all dormant programs are also listed. Dormant programs are programs that have been restored (RP'd) but are not executing. The US option lists program information for all sessions except for the system session.

`Prio` Priority of associated program. Lower values mean higher priority.

`PC` Program Counter; logical address of next instruction to be executed. This is an octal value, suffixed with a "c" if the address is in the CDS code partition.

If "-RTE-" is shown for the PC value, the program is suspended in an RTE kernel routine, as for a disk segment load from a downed or locked device.

The program's PC is updated when the request is complete.

Seg — Segment. For non-CDS programs, shows the overlay number currently in use by the program. For CDS programs, this is the CDS segment number.

DataPartition Size — Displays the size of the program in pages. This number includes EMA. For CDS programs, this refers only to the size of the data space.

DataPartition Status — Indicates where the program is in memory. For CDS programs, this refers only to the data space. Abbreviations used in this column are as follows:

`in`	– in dynamic memory.
`in` *n*	– in reserved partition number *n*.
`sw`	– swapped to the disk. The program is assigned to dynamic memory.
`sw` *n*	– swapped, but the program is assigned to partition *n*.
<*blank*>	– dormant program assigned to dynamic memory.
n	– dormant program assigned to partition n.

CodePartition Size — (For CDS programs only) Shows the amount of memory used by the code space of this program. This is equal to the number of code blocks times the size of the segments (in pages), plus one for the CST (code segmentation table).

CodePartition Status — (For CDS programs only) Indicates where the program's code is in memory. Abbreviations used are:

`in`	– code is in dynamic memory.
`in` *n*	– code is in reserved partition number *n*.
`sh`	– the code is shared and is in dynamic memory.
`sh` *n*	– the code is shared; resides in partition *n*.
<*blank*>	– program's code is not in memory, but is assigned to dynamic memory.
n	– program's code is not in memory, but is assigned to partition N.

Program Status — A description of the current activity of this program.

Following the program list, any critical status information is given (see below under the explanation of the WH, ST runstring). Critical levels are defined as follows:

- The largest free swap file block is less than 32 pages.
- The largest usable block of dynamic memory is less than 32 pages.
- There are fewer than 3 unused dynamic memory descriptors.
- One or more LUs are down.

WH Options

WH options are summarized in NO TAG.

WH Options Summary

Option	Description
AC	Shows all active (non-dormant) programs.
AL	Shows all restored programs (RP'd programs).
CL	Shows class number information.
D	Shows the prototype-identification list.
OP *prog*	Shows information about program *prog*.
PA	Shows memory partition information.
PL *progmask*	Shows programs with names selected by *progmask*.
RN	Shows resource number information.
SC	Shows scheduled and IO suspended programs.
SE	Shows all active sessions.
SH	Shows shared EMA information.
ST	Shows system status information.
US	Shows program information for user sessions.
session#	Shows programs in that session.

WH Option Examples

```
CI> wh                  (Display information about programs attached
                         to this session)

CI> wh se               (Display who is logged on)

CI> wh us               (Show what the users are running)

CI> wh 112 6            (Show session 112; send listing to LU 6)

CI> wh op ci/104        (Show program CI in session 104)

CI> wh pl ci@           (Show programs with names starting with CI )
```

WH, CL (Class Table Information)

The CL option displays class table information. A possible listing is:

```
         Assigned   Pending   Complete  Class
Class#   Ownership  Requests  Requests  Waiter
-----------------------------------------------------------------
  37     Global          0         0     CM
  38     PUTEM           0        20
  39     CI/47           1         0     CI/47
-----------------------------------------------------------------
  3 class numbers in use out of 50
```

`Class#`	The class number being displayed. At the end of the listing is the total number of class numbers used out of the total number defined at system generation time.
`Assigned Ownership`	Ownership of a class number may be assigned or global. The differentiation is made when the class number is allocated with a CLRQ request or through the initial EXEC request. If the class number is assigned to a program, it is deallocated when the program becomes dormant. Globally assigned class numbers must be explicitly deallocated.
`Pending Requests`	The number of I/O class requests that are pending (I/O not yet complete). For program-to-program class I/O, this number is always zero.
`Complete Requests`	The number of class buffers on the completed queue for this class number. These buffers are recovered by a class get call.
`Class Waiter`	If a program does a class get call and there are no completed buffers on the class number queue, the program waits. Note that only one program may wait for a particular class number at a time.

WH, D (Show Prototype ID Segments)

```
CI> wh d
```

List the ID segment prototypes (“proto-IDs”) in XSAM. These structures are used to create ID segments for programs without going to the type 6 file on disk. A proto-ID holds most of the information in the final ID segment, and much of the same information for RP’d programs is listed for proto-IDs. For example:

```
Proto-ID          DataPartition CodePartition
Name   Prio        Size Status   Size Status
-------------------------------------------------------------------
CI       51          32
MILES    4           20           164
-------------------------------------------------------------------
Sat Jan 18, 1992  12:16 pm
```

WH, SC (Show Scheduled Programs)

```
CI> wh sc
```

Display the programs that are scheduled, I/O suspended, or time scheduled to run within the next 5 seconds. The report format is the same as that shown when no WH option is given.

WH, PA (Memory Partition Listing)

A sample partition listing is shown below and the headings and fields are explained in the following paragraphs.

```
CI> wh pa

Ptn#   Page Range   Size   Occupant     Status            Priority
-------------------------------------------------------------------
  1     55-  86      32    D.RTR        saving resources         1
       87- 118      32    CI/47                                51
      119- 133      15    WH/47                                 5
-------------------------------------------------------------------
Sat Jan 18, 1992  12:16 pm
```

`Ptn#` — The Reserved Partition Number. A blank in this column indicates that this is a portion of the dynamic memory area. A program runs in a reserved partition only if it is assigned there by the “AS,program,n” command. “AS,program,0” makes an assigned program run in dynamic memory.

`Page Range` — The range of physical pages making up this block of memory.

`Size` — The number of pages in this block of memory.

Occupant	The name of the program or shareable EMA area occupying the memory block, or the message "bad" which implies that a parity error occurred in the block. If the memory is locked (it cannot be swapped out to disk to make room for other uses), that is stated (shareable EMA is always locked; EXEC 22 calls and unbuffered I/O calls can also lock memory).
Status	An optional description of partition status, including:

memory locked	The program has memory-locked itself with an EXEC 22 call (it cannot be swapped out to disk to make room for other uses).
io locked	The program is performing an unbuffered I/O call and is therefore memory-locked as above.
io + memory-locked	Both of the above are true.
saving resources	The program is dormant saving resources.
serially reusable	The program is dormant serially reusable.

Priority	Priority of the occupant program.

WH, RN (Display Resource Number Information)

The RN option shows resource number information. A sample listing is shown below and the column headings are described in the following paragraphs.

```
CI> wh rn

Resource#      Owner      Status
------------------------------------------------------------------
    1         Global       unlocked
    2         Global       unlocked
    3         Global       locked globally
    4         RN/102       locked to RN/102
------------------------------------------------------------------
4 resource numbers in use out of 30 available.
```

Resource#	The resource number involved.
Owner	The name of the allocating program or 'Global' if globally allocated (globally allocated resource numbers are not deallocated when the allocating program terminates).
Status	One of the following:

unlocked	The resource number is clear.
locked globally	The rn is locked, and can be unlocked by any program.
locked to *prog*	The rn is locked to program *prog* and can only be unlocked by that program.

WH, SH (Display Shareable EMA)

RTE-A provides a feature called Shared Extended Memory Area (SHEMA) to store data used by several programs. The SH option shows information about all the shareable EMA areas known to the system. A sample listing is shown below:

```
   CI> wh,sh
 Label  In System Count  Use Count  Location  Start Page  Size  Locked
-------------------------------------------------------------------------
 ABCDE        1              0          1         200       30     Yes
-------------------------------------------------------------------------
Sat Jan 22, 1989  12:27 pm
```

`Label`	Label of a specific EMA area used by one or more programs in the system. Shareable EMA usage, as opposed to local EMA usage, is specified by the LINK command SH,label.
`In System Count`	The number of programs in the system that use this shareable EMA area.
`Use Count`	The number of active programs in the system that use this shareable EMA area. The shareable EMA area is not deallocated (released from memory) until this count reaches zero, providing that the area is not locked.
`Location`	A number here refers to a reserved partition where the area is or will be allocated. “Dynamic memory” means the area is or will be allocated in the dynamic memory area.
`Start Page`	The starting physical page of the shareable EMA, if it is allocated.
`Size`	Size of the area in pages, if it is allocated.
`Locked`	If the area is locked, it will not be deallocated (released from memory) when the use count reaches zero. This allows a shareable EMA area to remain in existence with its data between executions of programs that use the area. The LKEMA subroutine locks an area; the ULEMA subroutine and the UL command unlock an area.

WH, ST (System Status Information)

When invoked with the ST option, WH prints out miscellaneous system resource and status information. One possible message is:

```
CI> wh st

Status Information
------------------------------------------------------------------
Largest block of free SAM =  6141 words
There are 199 unused dynamic memory descriptors out of 220 total
Largest usable block of dynamic memory =  713 pages
Largest free swap file area =  127 pages
Free ID segments = 52
Time-slice fence priority =  50
Background fence priority =  50
Security/1000 is ON
Down LU's :  4, 37
------------------------------------------------------------------
Sat Jan 22, 1989  12:08 pm
```

Numbers shown here will probably be different on your system.

SAM is System Available Memory, used for I/O buffering, class I/O, spooling, and string passage. See Chapter 8 of the *RTE-A System Design Manual* for information.

Dynamic memory descriptors are used to describe such parameters as location, size, and usage of individual blocks of dynamic memory. When no unused descriptors are available, a free block of memory cannot be subdivided.

The largest usable block of dynamic memory is the largest number of contiguous good pages in dynamic memory. A hard parity error resulting in a downed page can reduce this size. An attempt to run a program in dynamic memory that is larger than this size results in an SC09 error.

The largest free swap file area is the size of the largest program that may currently be swapped out to disk.

LUs are downed when a driver detects some error at the device level. After correcting the problem, inform the system that the device is again ready with the "UP,luNumber" command. If no LUs are down, this line is not printed.

Locking WH in Memory

If WH was loaded as a system utility and was RP'd before it was run, it will lock itself into memory, detach to the system session, and terminate saving resources the next time it is run. This guarantees that WH is available if the disk on which it resides goes down. To remove a memory locked WH, you must use OF when specifying the ID option.

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 or function.

Syntax:
```
WHILE command_list1
DO command_list2
DONE
```

command_list is 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 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), CI executes the DO branch. If the return status is non-zero (the command was unsuccessful), 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 reaches DO.

A WHILE-DO-DONE control structure can be nested in either another WHILE-DO-DONE or in 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 was just executed and CI is not executing commands as expected, make sure you entered a DONE command to terminate the control structure.

Example:

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. CALC is a user-written program that performs the specified math operation on the two integers and returns the result in variable $RETURN2.

```
set count = 0
WHILE IS $count lt 8
DO
   *
   * Increment counter and
   * retrieve result from $RETURN2
   *
   calc $count + 1
   set count = $return2
   *
   * Compile and link file if no errors
   *
   if ftn7x test_file$count.ftn 0 –
      then link test_file$count.rel
      else echo $return1 ‘errors in test_file ‘$count
   fi
DONE
```

WHOSD (Report Users; VC+ Only)

Purpose: Report users of a CI file or directory or a volume. This command supports redirection.

Syntax: `WHOSD [-t] [-m` *idmask*`]` *file*`|`*directory*`|`*lu*

`-t`	trace ID segments and proto-IDs back to a file name.
`-m` *idmask*	check only those ID segments and proto-ID segments whose names match the mask.
file`|`*directory*`|`*lu*	the file, directory, or volume of which you want to report the user.

Description:

WHOSD displays all of the users of a specified file, directory, or volume. Directories may not be purged while they are in use and CI volumes may not be dismounted from the system while they are in use.

A file is in use when it is open by any process, if it is an active type 6 file, or if the file is being used for the system swap area.

A directory is in use when it is being used as a working directory, when it is included in a user's UDSP, when the type 6 file of an active program resides in the directory, when a file is open in the directory, or when the system swap file resides in the directory.

The output of WHOSD can be redirected to an output file by specifying either "*>filename*" or "*>>filename*" in the runstring. The output file specified must be delimited by commas and is position independent. If the file already exists, it will be overwritten. To append to a file, "*>>filename*" can be used. If the file does not already exist, it will be created.

If the output file is not specified, WHOSD breaks the output into screen pages. Paging is disabled when an output file is specified. For example, to output to the terminal without paging, specify ">1" in the command line. Multiple redirection strings may occur in the runstring; however, only the last redirection is executed.

Return Values:

WHOSD returns, in $RETURN1, the number of users found for the specified file, directory, or volume. If an error is encountered, WHOSD returns a negative number.

Examples:

The following example shows all of the current users of LU 19.

```
CI> whosd 19
LU 19 is being used as the swap disk.
Directory – /HOMEDIR is being used by USER(115)
Directory – /MOREPROGS is being used by USER(115) in a UDSP
Program   – CI/115 is RPed from LU 19
Program   – DL/115 is RPed(proto) from LU 19
File      – /LOGS/NS_EVENT.LOG open to EVMON
```

In the following example, the /MOREPROGS directory is scanned for all uses. Any active program that resides on the same LU as /MOREPROGS.DIR is traced back to its type 6 file to determine if it resides in /MOREPROGS.DIR.

```
CI> whosd –t /moreprogs
Directory – /MOREPROGS is being used by USER10(115) in a UDSP.
Program   – CI/115 is RPed from /MOREPROGS/CI.RUN
```

Scan /MOREPROGS for working directories, UDSPs, and also check only those ID segments starting with the letter C:

```
CI> whosd –m c@ /moreprogs
Directory – /MOREPROGS is being used by USER10(115) in a UDSP.
Program   – CI/115 is RPed from /MOREPROGS/
```

The following example enables the trace option and logs all of the users of LUs 19 and 20 to the file USERS.LST:

```
CI> whosd –t >users.lst 19
CI> whosd –t 20  >>users.lst
```

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* [*wsSize*]

prog is the program name, up to five characters, with an optional session identifier.

wsSize is the working set size in pages. The range is:

$2 \leq size \leq 1022$ for Normal and Large model VMA programs

$2 \leq size \leq 32733$ for Extended model VMA programs.

Refer to “Models of EMA/VMA” in Chapter 4.

Description:

The working set is a number of pages in a VMA user’s partition that is used to hold a portion of the virtual memory space, including the page currently being accessed. Increasing the working set generally improves 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 or DT command.

XQ (Run Program Without Wait)*

Purpose: Schedules a program for execution without waiting for the program to terminate. Note that the CI and base set versions of this command are different. See "Base Set Version of the XQ Command" below for information on the base set version.

Syntax: `XQ` *prog|file* [*pram*5*]

prog|file is the program name, up to five characters, or a file descriptor that identifies a type 6 program file to be executed.

*pram*5* are the parameters to be passed to the program. The maximum runstring length, including the implied RU and delimiter, is 256 characters. This can be five numeric parameters or a character string.

Description:

The XQ command performs a "schedule without wait" operation. All other actions (comments and error handling) are identical to that of the RU command.

See the discussion on parameter passing and parsing in the *RTE-A Programmer's Reference Manual* for details about retrieving parameters as strings.

Base Set Version of the XQ Command

The XQ and RU commands are base set commands, meaning they may be entered at the "`System>`" or "`RTE:`" prompt, and may be executed programmatically using the MESSS routine. However, the base set versions of these commands differ substantially from the XQ and RU commands provided by CI, which may be executed programmatically using the FmpRunProgram routine.

The syntax of the base set commands is:

`XQ,`*program*`[/`*session*`],`*parameters*
`RU,`*program*`[/`*session*`],`*parameters*

Both commands schedule the named program without waiting for the program to complete, that is, they both behave like the XQ command.

The program to be scheduled must already be RP'd; the base set commands do not RP type 6 files as do the CI versions.

A session number may be specified to schedule programs in sessions other than your own or in the system session if you have sufficient capability. For example, "`xq ucsf/1`" runs program UCSF in session 1.

7

Command Editing

This chapter contains instructions for editing CI commands by using either the command stack editor or one of the visual editing modes (EMACS, GMACS, VI, or CSH). It also contains information about the CMNDO monitor which allows programs to easily enable the $VISUAL command editing functionality

/ (Command Stack Editor)

Purpose: Allows previously entered command lines to be displayed, edited, and re-entered as new command lines.

Syntax: `/[:][////|`*n*`][.`*pattern*`]`

`:` denotes auto-execute mode.

`////` represents the number of slashes that corresponds to the line number at which to start the frame; that is, "`///`" equals line number 3, "`////`" equals line number 4, and so on.

n is the line number at which to start the frame.

`.`*pattern* selects only lines that contain the pattern you specify. The pattern includes an optional anchor character, followed by one or more of the characters described below. The pattern syntax is:

`[^]{-|@|<`*char*`>|\<`*char*`>}...`

^	(optional) anchors the pattern to the start of the command line
-	matches any single character
@	matches zero or more characters
<*char*>	matches the character specified, both upper and lowercase
\<*char*>	matches the character specified (upper or lowercase) and allows "−" or "@" to be entered
\	quoting character

Description:

Note that although any number of slashes can be used to specify the line number at which to start the frame, entering one to four slashes (instead of /n) provides optimum speed. For line numbers higher than five, it is probably easier to enter a single slash followed by the desired line number.

Each command line entered is remembered in the "stack". The most recently entered commands are pushed onto the top of the stack, and the oldest commands fall off the bottom. Duplicate command lines are removed from the stack (unless $VISUAL is set without the NODUPES option). When you enter command stack commands, these command lines can be displayed and modified with the terminal editing keys and sent again to CI as new command lines.

The command stack commands are entered from the CI> prompt, displaying a frame of previous command lines and entering command stack editing mode. Examples of commands that switch to stack mode are shown in Table 7-1.

Table 7-1. Stack Mode Commands

Command	Description
/	Display the last frame of lines
//	Display the last line
///	Display the last two lines (and so on)
/*n*	Display a frame starting at line *n*
/.*pattern*	Display the last frame of lines containing *pattern*
/.^*pattern*	Display the last frame of lines starting with *pattern*
//.*pattern*	Display the last line containing *pattern*
/*n*.*pattern*	Display the last *n* lines containing *pattern*

The term "frame" in Table 7-1 refers to the maximum number of lines to be displayed in a command stack window; in CI, this is specified in the $FRAME_SIZE variable.

Additionally, you can include a colon (:) as the second character in any of the above command forms; this causes the first selected line to be automatically marked for execution and does not go into screen mode. For example, "`/:/`" re-executes the last line in the stack; "`/:7`" executes the seventh most recent command; "`/:/.edit`" executes the last line containing "`edit`".

The commands that select lines by patterns allow you to use the "`-`" and "`@`" characters in the pattern as single and multiple character wildcards, as in FMP masking. To specify either as a literal character, precede it with a backward slash (\). For example, "`/.edit \-b@.ftn`" finds all lines where EDIT was run in batch mode on any FORTRAN source; "`/.a@b-c`" matches lines "`axxbbc`" and "`gab!c`".

The command stack window is preceded by a banner that contains the starting line number of the window and the total number of lines selected for this display in inverse video, separated by a slash. For example, from CI:

```
CI> /3
--003/320-- Commands: [CI.STK::DEXTER]
edit glorp.ftn
ftn7x glorp 0 -,,s
link glorp.rel +de
```

This shows that the window starts at line 3 out of a total of 320 lines in the stack (CI also shows the name of the current stack file). If only those lines that contain a pattern are matched, the total line count reflects only those matching lines. For example:

```
CI> /3.edit
--003/324-- Commands: [CI.STK::DEXTER]
edit kaspritz.mac
? li;edit glink.mac
edit glorp.ftn
```

This shows that the window starts at selected line 3 out of a total of 24 lines selected by the pattern. To display the previous 20 lines that matched, use the ctrl-P editing command, as described below.

Once command stack mode has been entered, you may position the cursor to the desired line, modify it with the local terminal editing keys, and press <return> to execute the command. The editing commands shown in Table 7-2 are also recognized.

Table 7-2. Editing Commands

Command	Description
ctrl-A	Go to the start of the line where the cursor is positioned.
ctrl-D	Delete the current line from the stack.
ctrl-F	Display the following frame of selected lines.
ctrl-K	Mark current line for grouped execution in order of marking.
ctrl-P	Display the previous frame of selected lines.
ctrl-Q	Quit stack mode, start executing marked lines.
ctrl-U	May be entered instead of ctrl-Q on terminals using Xon/Xoff handshake protocol.
ctrl-Q ctrl-Q ctrl-U ctrl-U	Abandon stack mode, forget marked lines.
ctrl-Z	Go to the end of the line where the cursor is positioned.

In Table 7-2, “ctrl-A” denotes pressing the <ctrl> key and the letter “A” at the same time. The editing mode commands are recognized only when they are entered immediately before a carriage return. In addition, any of the stack display commands may be entered to display a new window.

The stack lines are displayed with display functions when needed. Lines that are longer than the screen width are continued by dots in the last two columns. To enter a new line in editing mode that is longer than the screen width, you must use the cursor control keys to put the dots in the last two columns. The cursor must then be placed back onto the first line of the command before selecting that line for execution or for executing one of the commands in Table 7-2 on that line.

A timeout in the stack mode read causes CI to exit stack mode and return to the normal CI prompt.

Command stacks can be saved in files. At logon, a file called CI.STK is searched for on the home directory first. If it is not found there, it 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 on the working directory 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, and see the examples given later in this chapter.

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 are 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.

Note that the stack subroutines are available for inclusion in user programs.

Examples:

Assume that the command stack contains the following and that the variable $FRAME_SIZE is equal to 20:

```
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
io 8
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> /
--003/320--  Commands:  [CI.STK::DEXTER]
? ex
dl
ru dl
ss spot.run
go spot.run
br spot.run
tm
wh
io 6
up 6
io 8
wh al
? wh
? pu
pu spot.1st
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. At this point, you can press the return key to return to CI without any further action or move the cursor 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, or you can use any of the stack mode commands described in the following section.

To display 20 lines beginning at command line 25 from the last command:

```
CI> /25
--025/320--  Commands:  [CI.STK::DEXTER]
? dl
dl ::system
dl
hello
?
? ex
dl
ru dl
ss spot.run
go spot.run
br spot.run
tm
wh
io 6
up 6
io 8
wh al
? wh
? pu
pu spot.1st
```

Note that the cursor is positioned at the top of the stack. If n is less than 20, the number of lines specified will be displayed. For example:

```
CI> /9
 --009/320--  Commands: [CI.STK::DEXTER]
wh al
? wh
? pu
pu spot.1st
pu spot.dbg
wh pa
edit testprog.ftn
co testprog.ftn backup.ftn
li testprog.ftn
```

You can achieve the same result by entering the following:

```
CI> //////////
```

The following command sequence displays the last command line:

```
CI> //
 --001/320--  Commands: [CI.STK::DEXTER]
li testprog.ftn
```

Posting and changing command stack files are done with the WD command. Following are examples of manipulating command stack contents and associated files:

Initial assumptions:	working directory	= /debbie
	associated command stack file	= /debbie/ci.stk
CI> wd,,+s	(Command stack contents posted to /DEBBIE/CI.STK)	
CI> 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 logoff or when the next WD command with the command stack option is executed.)	
CI> 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.)	

Changing the command stack display frame size is done with the SET command (described in Chapter 6) and predefined variable $FRAME_SIZE. An example follows:

```
CI> SET frame_size = 15          (Sets command stack display/frame size to 15)
```

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
io 6
up 6
io 8
```

To change the display size to 7 lines and then display the last 7 lines, enter the following:

```
CI> set frame_size = 7
CI> /
 --007/320-- Commands: [CI.STK::DEXTER]
go spot.run
br spot.run
tm
wh
io 6
up 6
io 8
```

$VISUAL Mode Command Line Editing (VC+ Only)

By setting the $VISUAL CI variable, you can select the desired command line editing mode. The supported modes are EMACS, GMACS, VI, and CSH. The EMACS, GMACS, and VI modes enable command editing functions much like the HP-UX ksh program. Setting $VISUAL to CSH enables a mode that provides some of the editing features of the HP-UX csh program. In the CSH mode, only the file name completion, command line directory lists, and command line control functions are available. The csh-style history substitutions are not available.

While editing the current line, the $VISUAL editing mode allows you to edit lines that are longer than your current screen width by scrolling through the line. For lines that are longer than the current screen width, the following symbols are displayed at the end of the line:

- `>` indicates that the line extends to the right.
- `<` indicates that the line extends to the left.
- `*` indicates that the line extends in both directions.

The current line is centered around the cursor as the cursor moves across the line. The default "viewing" width is 80 characters. Use the $COLUMNS variable to redefine the width.

Setting $VISUAL to EMACS, GMACS, VI, or CSH also has an effect on the functionality of the RTE command stack. By default, the RTE command stack routines do not insert duplicate lines in the stack. When setting $VISUAL, duplicate lines are allowed in the stack. To override this behavior, a "`,NODUPES`" can be added to the visual mode. For example, to use the VI editing mode without saving duplicate lines in the command stack, set $VISUAL to "`VI,NODUPES`".

The use of the $VISUAL editing modes is only supported when used with the HP 12040D 8-Channel MUX, the HP 12100A 4-Channel OBIO, or with a telnet psuedo terminal LU. FIFO mode is required when using the command editing features. When a port is not already in FIFO mode and command editing is enabled, the port will be reconfigured to enable FIFO mode after each prompt is issued. After the command line is terminated with the RETURN key, the previous state of the port is restored.

Performance Considerations

For optimal performance when using command editing, the port should be configured to use FIFO mode and Xon/Xoff handshaking. For example,

```
CI> cn,$session,33b,100000b    ! enable FIFO
CI> cn,$session,34b,101b       ! enable Xon/Xoff Protocol
                                 (Force type 5)
```

When using Xon/Xoff, the port must be in FIFO mode and the terminal's "G" (inhibit handshake) and "H" (inhibit DC2) straps must be set to allow the use of the standard RTE command stack and also the Edit/1000 screen mode function.

The required monitor program CMPLT should also be RP'ed and executed without wait in your WELCOME file as follows:

```
rp,/programs/cmplt.run
xq,cmplt
```

The CMPLT program performs the file name/command completion and the command line directory lists for the various command line editing modes. When a file name completion is requested, CMPLT is scheduled by CI to complete the file name. If CMPLT is already busy with another request, the system clones a copy of CMPLT for your request. To disable cloning, CMPLT can be linked as a system utility. In this case, when it is busy, the system will queue schedule CMPLT.

It is preferable to RP CMPLT as a permanent ID segment rather than a proto-id segment. When CMPLT is RP'ed as a permanent program and executed, it detaches into the system session and terminates serially reusable. Performance on global directory name completions or global directory lists is enhanced because CMPLT keeps a cache of the global directory names. When CMPLT is not RP'ed as a permanent program, the benefits of this cache are lost.

EMACS/GMACS Visual Editing Mode

This mode is used by setting $VISUAL to either EMACS or GMACS. The two modes are the same except for the usage of the ^T edit command (see the command description for the difference).

To edit the command line, move the cursor to the location you want to edit and insert or delete characters or words. All editing commands are control characters or escape sequences. A caret (^) followed by a character denotes a control character. For example, ^H is the notation for control-H. Note that the CTRL key and the 'H' are pressed simultaneously. The SHIFT key is *not* pressed.

The notation for escape sequences is M- followed by a character. For example, M-h (Meta h) is entered by pressing ESC (ASCII 033) followed by 'h'. M-H is the notation for ESC followed by SHIFT (capital) 'H'.

Edit commands may be entered at any location on the line. It is not necessary to enter the RETURN or LINE FEED key after an edit command, except where noted.

The following subsections describe the editing commands. The commands are grouped as follows:

- cursor motion commands
- deleting, killing, and restoring commands
- formatting commands
- marking commands
- searching commands
- other EMACS editing commands

Cursor Motion Commands

Note that a word is defined as a string of characters consisting of only letters, digits, and/or underscores.

Move forward (right) one character.	^F
Move forward one word.	M-f
Move backward (left) one character.	^B
Move backward one word.	M-b
Move to start of line.	^A
Move to end of line.	^E
Move forward to *char* on current line.	^]*char*
Move cursor backward to character char on current line.	M-^]*char*

Deleting, Killing, and Restoring Commands

Note that a word is defined as a string of characters consisting of only letters, digits, and/or underscores.

Delete previous character.	*backspace*
Delete one character forward.	^D
Delete one word forward.	M-d
Delete previous word (Meta-backspace). Delete previous word. Delete previous word (Meta-DEL).	M-^H M-h M-*del*
Delete from the cursor to the end of the line. If preceded by a numerical parameter whose value is less than the current cursor position, then delete from the given position up to the cursor. If preceded by a numerical parameter whose value is greater than the current cursor position, then delete from the cursor up to the given position.	^K
Delete from the cursor to the mark.	^W
Delete the entire current line. (*kill* is the user-defined kill character, as defined by the $KILLCHAR variable, usually ^G or DEL.)	*kill*
Yank (paste) last item removed (contents of delete buffer).	^Y

Formatting Commands

Transpose current character with next character (EMACS); transpose two previous characters with next (GMACS).	^T
Capitalize current character.	^C
Capitalize current word.	M-c
Change the current word to lowercase.	M-l
Line feed and print current line.	^L

Marking Commands

Set a mark (Null character).	^@
Set a mark (Meta space).	M-
Interchange the cursor and the mark.	^X^X
Put the region from the cursor to the mark into the delete buffer. Text and cursor position are unchanged.	M-p
Execute the current line (New line).	^J
Execute the current line (Return).	^M

Searching Commands

Fetch previous command. Each time ^P is entered, the previous command in the command stack is accessed.	^P
Fetch next command. Each time ^N is entered the next command in the command stack is accessed.	^N
Fetch the least recent (oldest) command stack line.	M-<
Fetch the most recent (youngest) command stack line.	M->
Reverse search for a previous command line containing *string*. If a parameter of zero is given, the search is forward. *string* is terminated by a RETURN. If *string* is preceded by a ^, the matched line must begin with *string*. If *string* is omitted, the next command line containing the most recent string is accessed. In this case a parameter of zero reverses the direction of the search.	^R*string*
Execute the current line and fetch the next line relative to current line from the command stack.	^O

Other EMACS Editing Commands

Define a numeric parameter, *digits,* to be taken as a parameter in the next command. The commands that accept a parameter are: ^F, ^B, *backspace*, ^C, ^D, ^K, ^R, ^P, ^N, ^], M-., M-_, M-b, M-c, M-d, M-f, M-h, M-l, and M-^H.	M-*digits*
The *n*th word of the previous command line is inserted on the line. If *n* is omitted, the last word of the previous command line is inserted on the line.	*n*M-., or *n*M-_
Attempt file name completion on the current word. Replaces the current word with the longest common prefix of all file names matching the current word with an @ appended. If the match is unique, a / (slash) is appended if the file is a directory and a space is appended if the file is not a directory.	M-ESC
List files matching current word pattern as if an @ were appended.	M-=
Multiply parameter of next command by 4.	^U
Escape next character. Editing characters and the user's *kill* character may be entered in a command line or in a search string if preceded by a \ (backslash). The \ removes the next character's editing features (if any).	\
Display version of the command editing subroutine.	^V
Insert an * at the beginning of the line and execute it. This causes a comment to be inserted in the command stack.	M-#

VI Visual Editing Mode

Within VI there are two modes: input mode and control mode. By default you are in input mode at the CI prompt. You can enter control mode by typing ESC (033). From control mode you can enter commands to move the cursor or modify the command line. Most control commands accept an optional repeat count prior to the command. You also can enter commands from control mode that will put you back into input mode.

Input Mode Edit Commands

Delete previous character.	*backspace*
Delete the previous blank-separated word.	^W
Escape next character. Editing characters, erase, or kill characters may be entered in a command line or in a search string if preceded by a ^V. ^V removes the next character's editing features (if any).	^V
Escape next character. Editing characters and the user's kill character may be entered in a command line or in a search string if preceded by a \. The \ removes the next character's editing features (if any).	\
Kill the entire current line. (User-defined kill character as defined by $KILLCHAR.)	*kill*
Enter control mode.	ESC

Control Mode Edit Commands

Cursor Motion Commands

The following commands move the cursor while in control mode. The optional [*count*] parameter causes a repetition of the command *count* times.

Move forward (right) one character.	[*count*]l
Move forward one alphanumeric word.	[*count*]w
Move to beginning of next word that follows a blank.	[*count*]W
Move to end of word.	[*count*]e
Move to end of the current blank-delimited word.	[*count*]E
Move backward (left) one character.	[*count*]h
Move backward one word.	[*count*]b

Move to preceding blank-separated word.	[*count*]B
Move to column *count*. Default is 1.	[*count*]|
Find the next character *c* in the current line.	[*count*]f*c*
Find the previous character *c* in the current line.	[*count*]F*c*
Equivalent to f followed by h.	[*count*]t*c*
Equivalent to F followed by l.	[*count*]T*c*
Repeats the last single character find command, f, F, t, or T.	[*count*];
Reverses the last single character find command.	[*count*],
Move to beginning of line.	0
Move to first nonblank character in line.	^
Move to end of line.	$

Search Commands

These commands access your command stack.

Fetch previous command. Each time k is entered, the next earlier command in the command stack is accessed.	[*count*]k or [*count*]−
Fetch next command. Each time j is entered, the next later command in the command stack is accessed.	[*count*]j or [*count*]+
The command number *count* is fetched. The default is the first command in the command stack.	[*count*]G
Search backward through the command stack for a previous command containing *string*. *string* is terminated by a RETURN or NEW LINE. If *string* is preceded by a ^, the matched line must begin with *string*. If *string* is null, the previous string is used.	/*string*
Same as / but search in the forward direction.	?*string*
Search for next match of the last pattern to / or ? commands.	n
Search for next match of the last pattern to / or ?, but in reverse direction. Search command stack for the string entered by the previous / command.	N

Text Modification Commands

These commands will modify the line.

Enter input mode and enter text after the current character.	a
Append text to the end of the line. Equivalent to $a.	A
Move cursor to the character position specified by *motion*, where *motion* is a motion command, deleting all characters between the original cursor position and new position, and enter input mode. If *motion* is c, the entire line is deleted and input mode entered.	[*count*]c*motion* c[*count*]*motion*
Delete the current character through the end of line and enter input mode. Equivalent to c$.	C
Delete the character under the cursor and enter input mode; the entered text replaces the deleted character. *count* specifies how many characters on the current line are replaced.	[*count*]s
Equivalent to cc.	S
Delete the current character through the end of line. Equivalent to d$.	D
Move cursor to the character position specified by *motion*, where *motion* is a motion command, deleting all characters between the original cursor position and new position. If motion is d, the entire line is deleted.	[*count*]d*motion* d[*count*]*motion*
Enter input mode and insert text before current character.	i
Insert text before the beginning of the line. Equivalent to the two character sequence 0i.	I
Place the previous text modification before the cursor.	[*count*]P
Place the previous text modification after the cursor.	[*count*]p
Enter input mode and replace characters on the screen with characters you type overlay fashion.	R
Replace the current character with c.	[*count*]rc
Delete current character.	[*count*]x
Delete preceding character.	[*count*]X
Repeat the previous text modification command.	[*count*].
Invert the case of the current character and advance the cursor.	~

Causes the *count* word of the previous command to be appended at the current cursor location and places the editor in input mode at the end of the appended text. The last word is used if *count* is omitted.	[*count*]_
Attempt file name completion on the current word. Replaces the current word with the longest common prefix of all file names matching the current word with an @ appended. If the match is unique, a / is appended if the file is a directory and a space is appended if the file is not a directory.	ESC

Other Edit Commands

Yank current character through character that *motion* would move the cursor to, where *motion* is a motion command, and put them into the delete buffer. The text and cursor are unchanged.	[*count*]y*motion* y[*count*]*motion*
Yanks from current position to end of line. Equivalent to y$.	Y
Undo the last text-modifying command.	u
Undo all text-modifying commands performed on the line.	U
Line feed and print current line. Has effect only in control mode.	^L
Execute the current line, regardless of mode. (New line)	^J
Execute the current line, regardless of mode. (Return)	^M
Equivalent to I*<cr>. Sends the line after inserting a * in front of the line. Useful for inserting the current command line in the command stack without executing it.	#
List the file names that match the current word if an @ were appended to it.	=
Kill the entire current line. (User-defined kill character as defined by $KILLCHAR.)	*kill*

CSH Visual Editing Mode

In CSH mode, only the file name completion, command line directory lists, and command line control functions are available. The csh-style history substitutions are not available.

Attempt file name completion on the current word. Replaces the current word with the longest common prefix of all file names matching the current word with an @ appended.	ESC
List files matching the current word pattern as if an @ were appended.	^D
Re-print the current command line.	^R
Erase the last word entered on the current command line.	^W
Kill the entire current line. (User-defined kill character as defined by $KILLCHAR.)	*kill*
Escape next character. Editing characters and the user's kill character may be entered in a command line. The \ removes the next character's editing features (if any).	\

CMNDO Monitor

The CMNDO monitor allows programs to easily enable the $VISUAL command editing functionality. CMNDO is a slave program that handles all interactive terminal I/O and command stack operations for a program. The exported environment variables, $VISUAL, $KILLCHAR, $COLUMNS, and $LINES, are used by CMNDO to allow a user's command line editing environment to be available in programs other than CI.

There is a separate monitor for each program that uses CMNDO. Because CMNDO requires additional ID segments, usage of the CMNDO monitor is controlled by the $CMNDO exported environment variable. $CMNDO can be set to TRUE or FALSE to control CMNDO usage in programs that support CMNDO. Command line editing with CMNDO can only be used in sessions that have an exported environment variable block.

To use CMNDO for all programs on your system that support CMNDO, set $CMNDO as follows:

```
CI> set -x CMNDO = T
```

Variables, such as $CMNDO_LINK, can also be used to control CMNDO usage from specific programs, in this case LINK. For example, $CMNDO_LINK can be set to TRUE so that only LINK uses CMNDO. The FST and FTP utilities can also optionally use the CMNDO monitor by setting the $CMNDO_FST and $CMNDO_FTP variables. (See the *RTE-A Backup and Disk Formatting Utilities Reference Manual*, part number 92077-90249, for more information on FST and CMNDO usage. For more information on FTP and CMNDO usage, see the *NS-ARPA/1000 User/Programmer Reference Manual*, part number 91790-90020, or the *ARPA/1000 User's Manual*, part number 98170-90002.)

To add CMNDO to an existing application, the CMNDO interface routines are documented in the *RTE-A • RTE-6/VM Relocatable Libraries Reference Manual*, part number 92077-90037.

8

Exception Condition Handling

This chapter describes situations that do not occur often but whose causes and the required corrective action may not be apparent.

Unusual File Access Errors

A number of unusual problems can occur when using files, as discussed in the following paragraphs.

Non-Standard File Names

Non-standard characters are those other than A - Z, a - z, 0 - 9, and underscore. The RTE-A file system assigns specific meanings to the characters period (.), slash (/), and colon (:). However, FMGR files have different file name requirements, as discussed below. In addition, the commands that 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 any program that uses FMP masking.

If a file name has the high order bit in a character set, it will print as a normal file name but cannot be manipulated or purged from CI unless an Fmp routine is used with the appropriate ASCII or non-ASCII characters.

Often, FMGR files have special characters in them. This was a naming convention, because 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. We recommend that you change all FMGR files with special characters to conform to the RTE-A file system convention as mentioned above, by renaming them from FMGR.

File Not Found

The RTE-A 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 receiving the "No such file" message. The RTE-A file system search procedure 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, FMGR disks are also searched. If the program only understands FMGR files (such as FMGR, FC and other subsystems), only FMGR disk directories are searched.

2. If no name is specified, the working directory is 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, LINK, and BOOTEX, look in special directories if the file is specified without a directory name. Programs using FMGR subroutines receive the "No such file" message when accessing files in a hierarchical 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 it 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 hierarchical 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 use directory XX. We recommend 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", program D.RTR is 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 is 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 up. Refer to the *RTE-A System Design Manual* for details.

OWNER, PROT, or WD Command Failures

The HP 92078A Virtual Code Package is recommended for effective use of the file ownership and protection attributes and for multiuser program development in general. Without this optional package, there is only one owner, to whom all files belong. This owner is a superuser, so file protection is meaningless. There is only one working directory, even if several people are using the system. This means the working directory set by the last user remains in effect, which may not be useful for other users.

If an account is purged and subsequently recreated, the directory/file ownership or associated group for that user or group, respectively, must be reassigned. Purging an account removes the user or group ID number used in the multiuser environment, and this number is used for directory/file ownership or associated group, respectively. This also applies to moving a disk volume from one system to another system with a different set of user and group accounts, causing incorrect ownership and associated group. This problem can also be corrected with the OWNER command.

Unusual Logon Errors

In a multiuser environment, you may receive error messages such as those shown below.

```
No such directory LOGONPROMPT::USERS

File read protected

Directory read protected
```

The first message indicates that the user information files were not initialized by the USERS program. It may also indicate that the volume containing the USERS directory was dismounted. Refer to Chapter 5 of this manual and to the *RTE-A System Generation and Installation Manual* for more information about logon problems.

Disk Volume Full

There are limits to how much data can fit on a disk. Flexible disks, such as mini- and micro-floppies, have limited storage space; other disks 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 disk space” is displayed.

This message means that there is not enough contiguous, free disk space on a particular volume. There may be adequate free space available on other volumes; or there may be enough total free space in the disk but in smaller pieces 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 shows how much space is available on a volume (refer to the description of FREES in this manual for details). 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 unneeded files to reclaim disk 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. Use the command DL @.@ +A to sort by time of last access to help you decide.

2. Purge files that are close to each other; "DL @.@ +L" can be used to sort by disk 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 MPACK program can be used to rearrange the files on the disk to produce more contiguous space. MPACK can be run concurrently with other operations. Refer to the description of MPACK in this manual for details.

You also can move a directory to another disk volume.

This involves the following five steps:

1. Create a directory on another LU.

2. Copy all subdirectories and files in the first directory to the new directory.

3. Purge all files and subdirectories in the first directory.

4. Purge the first directory.

5. Rename the new directory.

For example:

```
CI> crdir /newdir 47                  (Create new directory on LU 47)
CI> co /test/@.@.d /newdir/           (Copy files to directory NEWDIR)
CI> pu /test.dir.d                    (Purge all files and subdirectories;
     :                                 this command needs to be entered
CI> pu /test.dir.d                     until directory TEST is empty)
CI> pu /test.dir                      (Purge directory TEST)
CI> rn /newdir /test                  (Rename the new directory)
```

If you are moving directory PROGRAMS to another LU, enter the command RP,CIX.RUN before beginning the above steps. When PROGRAMS has been moved, you should OF the ID segment for CIX.RUN.

For a permanent solution, the system must be regenerated to change the way disk space is allocated. Having fewer, larger volumes reduces the number of times that a volume is full, though it will make the problem worse if and when it does happen. There are no disk 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 disk space.

For FMGR cartridges, disk 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. Refer to the description of FPACK in this manual for details.

Disk Volume Dismounted

One or more disk 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". Certain directories are required for the operating system operations. For example, if the directory PROGRAMS is not available, the DL, IO and WH commands stop working. Spooling and multiuser handling will have problems if directories SYSTEM and USERS disappear.

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. Use the IO program to find the dismounted volume (not the CL command). Look for disk devices. This assumes that the disk with IO on it has not been dismounted.

A disk 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.

Clearing Open Files

A CI or FMGR file may be left opened by a program that aborts or terminates before the file can be closed. A file left open in this way has an invalid open flag.

CI files left open are closed any time a program makes its first D.RTR call. The CI commands LI or DL, for example, call D.RTR. When called, D.RTR scans for all open flags in memory and closes the invalid ones.

FMGR files left open are closed by D.RTR any time D.RTR finds them while scanning FMGR cartridges to create, rename, and purge files, and to lock, pack, mount, and dismount cartridges. A common method to force D.RTR to clear open FMGR files is to use the FMGR DL command.

For more details about clearing open files, refer to the *RTE-A Programmer's Reference Manual*.

Parity Errors

High-speed computer memory is subject to intermittent errors. They are not very common, but can be bad when they occur. RTE-A detects these errors through parity checking or corrects them with Error Correcting Circuitry (ECC). If an error occurs that cannot be corrected by hardware, the RTE-A system handles the error.

If the error occurs in memory occupied by the operating system, operating system tables, or the program being swapped, or if the parity error handling module (PERR) was not specified at generation time, the system is halted. This is an unrecoverable error.

On all A-series systems except A900, if the error occurs in a program, the system aborts the program and reports an error. Other programs continue to run, and the offending program can be rerun later, usually without error.

On an A900 (because of the type of memory that is used), if the error occurs in a program, the system is halted. This is an unrecoverable error.

Most parity errors are not reproducible and are known as soft errors. Some errors are repeatable, and when one of these occurs, RTE-A blocks off the offending page (or partition if it is a reserved partition), making that page unavailable for use. This keeps other programs from using the bad memory location.

VCP Interrupt

Sometimes the computer halts and goes back to running the VCP program. If you see the system displaying the VCP prompt "VCP>" at the VCP terminal, check the octal display of the contents of various registers in the computer. Look at the last number printed, following the letter T. If this number is either 1020xx or 1030xx, where xx is any two digits, the system halted. If this is not the case, the BREAK key was accidentally pressed, and entering %R will return the system to the normal state. The VCP commands are described in the *RTE-A System Generation and Installation Manual*. There is a processor switch which can be used to prevent this occurrence. Refer to your computer's system installation and service manual for more information.

There are several reasons why system halts occur. The halt numbers are as follows:

102000: This normally occurs only at boot up. It indicates that the BOOTEX on the system must be used or the generator has allowed more driver partition pages than the system allows.

102003: This indicates a serious error such as a memory protect or unimplemented instruction error when no program was running. If you are developing your own drivers, this is a fairly common occurrence indicating a driver bug. If not, it probably means an error in the HP supplied software. Contact an HP service representative, recording what you did just before the problem and what values were printed out by the VCP.

102004: Power failed in a system not set up to handle power failure.

102005: This means a parity error in the system (see above). You probably have bad memory in the area used by the system. The A- and B-Registers indicate where the error occurred. It may be a soft error, allowing you to reboot, or a hard error, requiring changing the memory board.

103003: If this occurs immediately after the message “Boot process complete,” it probably means an I/O configuration card problem. Consult your computer reference manual. If it happens any other time, check if anyone was using Symbolic Debug on privileged code; if not, treat as any other halt.

Other halts: If you are developing privileged code that runs with memory protect disabled, you have probably corrupted the computer memory. Otherwise, contact your HP representative.

Missing System Programs

There are a number of protected system programs in the RTE-A system, for example, D.RTR, PROMT and LOGON. You cannot OF, suspend, or change the priority of these programs unless you are a superuser.

When a protected system program is OF’d or aborted, the associated subsystem will not work correctly. For example, OFing D.RTR is catastrophic, as all open file information is lost, and all mounted LUs are temporarily dismounted. Any programs that were accessing files no longer work correctly. The same applies to OFing PROMT or suspending LOGON; multiuser operation is not the same until the system is rebooted. Similar warnings apply to DS, Image and graphics programs. If one of these programs is OF’d, it must be restored, and the subsystem must be restored to the correct operating state.

In some extreme cases, suspending a program or changing priority has effects as disastrous as removing the program with the OF command. The general rule is never to change any attribute of a protected system program.

If CI is the only program in a session and it is terminated for any reason, the terminal is ready for the next logon. If CI is terminated and there are other programs in the session, CM is the interface for the terminal. CI can then be run, as follows:

```
CM> ci
```

and again be the main user interface.

The RTE Prompt

If you receive the following prompt instead of the expected program prompt, you are interfacing directly with the RTE-A system:

```
RTE:
```

This may occur in a single-user system. There may be a problem with the terminal handling program (CI or FMGR), or if this prompt appears right after booting the system, you may have forgotten to select a terminal handling program in your generation or Welcome file or both.

Enter "`ps,fmgr`" to see what FMGR is doing, or "`ds,`*lu*", for example, to display the I/O status of your system disk LU. You may learn that you have no FMGR or that an LU is down. In general, you may have to examine your generation listing, the Boot command file, and Welcome file to see why the problem occurred, and then fix the problem.

The following commands can be entered at the RTE prompt:

AS	DN	GO	PS	SZ	UP	XQ
BR	DS‡	OF	RU	TM	VS	
CD†	DT	PR	SS	UL	WS	

These commands are included in various system modules that must be relocated during system generation. If a system module is not relocated during generation, the commands provided by that module will not be available. Refer to the *RTE-A System Design Manual* for more information on the system modules.

The System Prompt

In the multiuser environment, problems can occur if CM is used to perform tasks other than the ones recommended. The RTE system responds by displaying the following system prompt:

```
System>
```

At this level, you can remove CM with the OF command only if you are a superuser. You may remove CM in such cases, because CM will return. In general, the system prompt indicates that a problem exists (your system is very busy or CM is currently busy). When the system prompt is displayed, any of the following commands can be entered:

AS	DS‡	OF	RU	TM	VS
BR	DT	PR	SS	UL	WS
CD†	GO	PS	SZ	UP	XQ

These commands are included in the various system modules that must be relocated during system generation. If a system module is not relocated during system generation, the commands provided by the module will not be available. Refer to the *RTE-A System Design Manual* for more information on the system modules.

---

† The CD system command is the same as the CI CZ command. Refer to the CZ command description in Chapter 6 of this manual for details on using the CD system command.

‡ The DS command can be entered at the FMGR prompt but not the CI prompt. Refer to the FMGR documentation in this manual for a description of the DS command.

Power Failure

If your system is equipped with battery backup, it can recover from power failures lasting up to several minutes. Battery backup maintains the state of memory, allowing RTE-A to recover where it left off when power failed.

The status of programs is maintained during power failures. All data in memory is intact, although the program does 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.

A

Error Messages

Most of the error messages caused by an operator action are simple and self-explanatory. However, some 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. Operator error messages are listed in alphabetical order, along with the explanation and suggestions for corrective action, in the “Error Messages” section.

Error Formats

Error messages have different formats depending upon the operation being performed. They can be grouped into the following categories:

Error messages and codes
I/O errors
Program abort errors
Parity errors
DS file access errors (only with optional DS-1000/IV)

Error Messages and Codes

Errors reported by CI in response to commands such as AS, RU, and SZ are in the form of brief descriptive messages. For example:

```
Illegal variable name <name>
Usage: RP file [progname]
```

There may be occasions 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
```

Some errors are reported in the form of an error code. These are reported in the form:

```
FMP error –59
```

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 occurred.

The error codes are listed and described under the heading FMP Error Codes in this appendix.

I/O Errors

Errors reported by RTE-A during an I/O operation are listed in the following format:

```
I/O Device Error on LU 55 The reason is:
Device not ready
```

The appropriate message is displayed on the second line. The error messages are listed below.

```
Addressing error
Data communication error
Device information specified at generation time is wrong
Device is write protected
Device not ready
Device timed out
Drive fault
End of tape detected
Group poll error
I/O request error
Request has been flushed
Serial poll error
Special driver defined error= 43
Transmission error
```

The last two error messages in the list are issued when the system does not recognize the error reported by the driver and the I/O request will not be retried.

Program Abort Errors

When a program is aborted, an abort message is displayed indicating the program name, the memory location, the reason it was aborted, and the contents of the registers. A sample program abort message is shown below.

```
PETST  aborted at address  26606 Reason is  MP   Current segment=     0
A=     3  B=  1401  X=     0  Y=     0  E= 1  O= 0  WMAP= 102100
Instruction=177777  Z=     0  Q=     0  CS mode=OF
MP = memory protect (I/O instruction or store/jump to protected memory)
```

The reason for program abort can be any of the abbreviations given below or an error code. The error code can be preceded with different mnemonics. A sample of each is listed below. Refer to the *RTE-A Programmer's Reference Manual* for a complete listing of the error codes and their explanations.

MP = memory protect (I/O instruction or store/jump to protected memory)
SR = privileged subroutine call error
UI = unimplemented instruction in user area
RQ = bad or too many EXEC parameter(s)
LD = disk tried five times to load program/segment and failed
SW = disk tried five times to swap program and failed

IO01 = Not Enough Parameters in EXEC call
IO04 = Illegal Buffer Address in EXEC call
IO14 = LU is down
IO24 = Spool file error; usually disk full.

SC01 = not enough parameters for EXEC schedule call
SC09 = program is too large to fit in memory

RN02 = undefined resource number
RN03 = cannot clear RN not locked to program or invalid RN

LU02 = Illegal LU

CL01 = Illegal Class Number or no class table

VM81 = not a VMA program
VM82 = VM address beyond program bound or VM file too small
VM02 = duplicate on VM filename

EM81 = not a EMA program
EM82 = EM address beyond program bound

CS00 = CDS software not installed
CS01 = Segment load requested by instruction in data segment
CS05 = CDS program is corrupt, internal error found

EV00 = invalid environment variable block.
EV01 = incorrect number of parameters.
EV02 = illegal parameter value.
EV04 = environment variable block is busy (no-suspend bit set).

Parity Errors

The parity error message format is illustrated below. A description of parity errors is given in Chapter 7 of this manual.

```
Parity Error occurred at physical page   132 page address   63
Pages actually marked bad (downed) =   2
PE = Parity Error in user code or data space
```

The parity error display indicates the page where the error occurred and provides a descriptive message (whether or not the error was caused by hardware or software).

Error Messages

Active working directory

This is reported by purge or dismount when you try to purge another user's working directory or when you try to dismount a disk containing a working directory. (FMP error −229)

Already 255 programs using SHEMA area

There are already 255 programs RP'd that access the shareable EMA area specified by the program. (FMP error −059)

Bad record length

There was an attempt to read or position to a record not written, or on update to write an illegal record length; check the position or size parameters. (FMP error −5)

Break flag detected

User entered a BR command, stopping the operation. (FMP error −235)

Can only run unshared

This is 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 OF'ing programs or by regenerating the system. (FMP error −241)

Cannot access account

A logon error occurred other than the normal user, account, and session limit errors. (FMP error −306)

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. (FMP error −212)

Cannot modify an active shared program code partition

The shared code partition is being used by an active program. Modification can only be made when all programs sharing the code partition have been completed. Reported by the RTE system.

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 so that this error will not be reported again with the current system. It does not fix the problem (if there is one) but it suppresses error messages if the program works. (FMP error −240)

Connection broken

The remote system monitor TRFAS was restarted since the connection was opened. (FMP error −308)

DCB is not open

You attempted to access an unopened DCB. Check the error code on open attempt. (FMP error −11)

Device I/O failed

You tried an illegal read/write on a type 0 file. Do not attempt to read/write or position a type 0 file that does not support the operation; check the file parameters. (FMP error −017)

Did not ask to read

Specify the R option in the open request. (FMP error −202)

Did not ask to write

Specify the W option in the open request. (FMP error −203)

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 disk or just store the ones you need. Normally this occurs when a disk is mounted before being initialized. (FMP error −103)

Directory is empty

No files are in the specified directory. (FMP error −51)

Directory is full

There is no more room in the file directory; purge files and pack the directory with the FMGR PK command if possible, or try another cartridge. (FMP error −14)

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. (FMP error −201)

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. (FMP error −211)

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. (FMP error −206)

Directory write protected

You are not allowed to write the directory containing the file, so you cannot change, purge it, or rename it. (FMP error −207)

Disk error

The disk is down; try again and then report it to the system manager of facility. (FMP error −1)

Disk I/O failed

This is reported when D.RTR tries to access an LU that is down or when any EXEC error occurs on an FMP disk access. (FMP error −242)

Disk is locked

The cartridge is locked. Initialize the cartridge if it is not initialized; otherwise, try again. (FMP error −13)

Disk is not mounted

The indicated volume was not mounted, so it cannot be dismounted and directories cannot be created on it. Mount the disk. (FMP error −214)

Disk LU is down

D.RTR tried to access a disk LU that is down. (FMG error −252)

Disk LU is locked

D.RTR tried to access a disk LU that is locked to another program. (FMP error −253)

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 illegal characters in a filename (> or [ ). (FMP error −220)

Duplicate directory name

A duplicate name was encountered. Check the destination of the directory being created. (FMP error −208)

D.RTR EXEC request aborted

D.RTR has tried something unreasonable, probably because of a corrupted cartridge list or a disk error. (FMP error −99)

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. (FMP error −102)

DS error DS08 (0), node 4

This generic error message displays when an unknown DS error occurs. These are in the DS manual.

DS is not initialized

DS has not been started with DINIT. (FMP error −310)

DS link is not connected

Hardware problem. (FMP error −311)

File already exists

A file already exists with the specified name; repeat with a new name or purge the existing file. (FMP error −2)

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. (FMP error −221)

File is already open

You made an attempt to open a file already open exclusively, or an FMGR file is open to seven programs. (FMP error −8)

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. (FMP error −204)

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. (FMP error −205)

Illegal file position

You attempted to read, write or position beyond the file boundaries. Check the record position parameters; the result depends on the file type and call. (FMP error −12)

Illegal interrupt from SC*xx*

This is reported when an interrupt occurs from a select code that is not in the system select-code table. Most likely a generation error occurred in select code specification. This is reported by the RTE system.

Illegal LU

You cannot access an undefined LU. (FMP error −18)

Illegal name

The filename does not conform to syntax rules; correct the name. (FMP error −15)

Illegal program file

This is reported if a program file is not a type 6 file, was not loaded with a snap file compatible with the current system, or is a non-transportable file from another system. (FMP error −238)

Illegal remote access

This message usually indicates an internal error. Either an invalid connection number was specified or an invalid request was routed to the DS transparency software. (FMP error −300)

Illegal use of directory

The directory was used when it should not be, such as in creating a file with a .DIR file type extension. (FMP error −230)

Incorrect password

The correct password was not entered. (FMP error −305)

Incorrect security code

There was an attempt to access a file without the correct security code. Use the correct code or do not try to access a file. (FMP error −7)

Input is not proper for this command

There may be a parameter missing or a bad number specified. Reported by the RTE system.

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. Reported by the RTE system.

LU has old directory

This LU has an old directory, and you did not tell FmpMount to re-initialize old directories. (FMP error −222)

Missing extent

A request was made for a file extent that was missing from the file. The file is probably corrupt. Purge the file. (FMP error −104)

More than 255 extents

An attempt to create more than 255 extents was made. Use a file with a larger initial size. (FMP error −46)

Must be a CDS program to use this command/parameter

This is returned for CD and DT commands when they are used on non-CDS programs; also for AS commands which specify the C parameter. Reported by the system.

Must not be remote

The specified file or directory must be on the local system. (FMP error −237)

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. (FMP error −218)

No free ID segments

The system cannot restore the program due to lack of ID segments. Try removing programs no longer needed. (FMP error −224)

No files selected

Nothing matches the mask supplied. (FMP error −50)

No room in shareable EMA table

Insufficient free XSAM exists to create SHEMA table entry. (FMP error −55)

No such account

No user has the name specified. (FMP error −304)

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. (FMP error −209)

No such cartridge

The specified cartridge is not mounted. Check the disk specification. (FMP error −32)

No such file

You tried to access a file that cannot be found. Check the filename or cartridge number. (FMP error −6)

No such group

The group name specified was not found. This is reported by the OWNER command when the associated group does not exist. (FMP error −254)

No such node

The local system does not recognize the node number or name specified. It may not be in the NRV. (FMP error −302)

No such user

Reported by the OWNER command when the user does not exist. (FMP error −233)

No TRFAS at remote system

Remote monitor TRFAS is not RP'd at the remote system. (FMP error −313)

No working directory

Returned by FmpWorkingDir when there is no working directory established and by some other calls when a filename is specified with no directory but no working directory exists. (FMP error −200)

Parameter is not proper for this command

Some parameter value entered is out of range. Reported by the RTE system.

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. (FMP error −37)

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. This is also returned by the XQ command when the program is running and cannot be cloned with a new name. (FMP error −225)

Program aborted

The given program was OF'd or aborted before it ran to completion. Returned by the RTE system.

Program name exists

You cannot RP a program with that name because there already is one. OF the old name with the ID parameter or choose another name. (FMP error −239)

Program name exists in another session

An attempt has been made to RP a program in the system session while a program of the same name is already RPd in another session. (FMP error −251)

Ran out of disk space

The disk specified for a disk file has insufficient room for file creation. This could occur when an extent is being created. (FMP error −33)

Remote system does not respond

The remote system is probably down or not running DS. (FMP error −312)

Security violation detected

The Security/1000 subsystem has detected an attempt by the user or program to perform a function for which there is insufficient capability. (FMP error −249)

String is too long

This is returned by FmpReadString and FmpWriteString when the passed string is longer than 256 bytes. (FMP error −231)

Symbolic link must not be remote

An attempt was made to access a symbolic link on a remote system and the remote symbolic link referred to a remote file. Create a symbolic link on the local system that refers to the destination of the remote symbolic link being accessed. (FMP error −262)

Symbolic link results in illegal path

Symbolic link interpretation yields a path name greater than 63 characters. Symbolic links that contain relative path names can be changed to absolute paths. (FMP error −261)

System cannot do CDS

This is reported when you try to RP a CDS program into a non-CDS system. (FMP error −245)

System common changed

This is reported when you try to RP a program with a system common defined differently from the current system. (FMP error −246)

System does not support symbolic links

At attempt was made to create a symbolic link on a system that does not support symbolic links. A CDS version of D.RTR is required and programs accessing symbolic links must be linked with either $SFMP or $SCDS. (FMP error −263)

System processes can only be changed by superusers

This message is returned when a non-superuser uses the superuser capabilities in commands such as OF, PR, or SS; it is also returned if action is taken by a non-superuser that may affect a protected system process. Reported by the RTE system.

That LU is not assigned on this system

The LU specified is invalid for a status request; this is usually not seen from CI. Reported by the RTE system.

The disk where the program resides is down

A hardware problem with the disk drive is preventing execution of the program.

Too many directories

D.RTR has no room to record this global directory; this may occur upon mounting or creating a directory. Closing some files or dismounting a volume should provide temporary relief; a long-term solution is to size D.RTR bigger, open fewer files, or have fewer global directories. You may be able to rename some global directories as subdirectories. (FMP error −215)

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, open fewer files, or have fewer global directories. (FMP error −213)

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. (FMP error −301)

Too many symbolic links in path

D.RTR traversed more than 8 symbolic links. This is probably a closed symbolic link loop. (FMP error −260)

Unable to schedule program <*name*> on interrupt to driver

The 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

This is returned when you ask for unavailable information about a FMGR file, such as time stamps. (FMP error −232)

Unpurge failed

The disk space or a directory entry occupied by the purged file has been reclaimed, so the file cannot be recovered. (FMP error −210)

User is not in group

The user is not in the group specified as the user group parameter for FmpSetOwner. (FMP error −255)

Value is too big

Check the command syntax for the valid range. Reported by the RTE system.

Value is too small

Check the command syntax for the valid range. Reported by the RTE system.

Value must be positive

Enter the proper positive value. Reported by the RTE system.

You do not own

You must be the owner of a file to change its protection information, and you must own a directory to change its owner. A superuser is required to performed the desired task. (FMP error −216)

FMP Error Codes

−001 Disk error!

The disk is down; try again and then report it to the system manager.

−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 Record size illegal

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 Incorrect 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 Must not be a device

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 is 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 Disk is 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 Size = 0 or illegal type 0 file access

Wrong type code supplied; attempt to create or purge type 0 file or create 0-length file; check size and type parameters.

−017 Device I/O failed

Attempt to read/write or position type 0 file that does not support the operation; check file parameters, namr.

−018 Illegal LU.

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 disk specification in call.

−033 Ran out of disk space

Disk specified for a disk file has insufficient room for file create. Could occur during a WRITF if an extent is being created.

−034 Disk is already mounted

Disk is mounted as an FMGR or hierarchical volume.

−035 Already 63 disks mounted to system

Only 63 disk LUs may by mounted at one time.

−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 offed 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 More 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.

−051 Directory is empty

The specified directory contains no files.

−053 Program assigned to bad partition

The program (for non-CDS programs) or the data partition (for CDS programs) is assigned to a reserved partition which is "bad" due to a parity error in the partition or a reserved partition which is undefined. Use the AS command to re-assign the program (or the AS command with the "D" option to re-assign the data partition) to a good partition.

−054 Partition too small for program

The program (for non-CDS programs) or the data partition (for CDS programs) is assigned to a reserved partition that is not large enough to hold the program or data partition. The program or data partition must be assigned to a larger reserved partition or dynamic memory.

−055 No room in shareable EMA table

Insufficient free XSAM exists to create SHEMA table entry.

−056 SHEMA assigned to non-existent partition

The shareable EMA area used by the program is assigned to a reserved partition which was not defined (by the AS or RV 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'd 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.)

−057 Partition too small for shareable EMA

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.

−058 Program assigned to SHEMA partition

The program (for non-CDS programs) or data partition of the program (for CDS programs) 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 or dynamic memory. 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.

−059 Already 255 programs using SHEMA area

There are already 255 programs RP'd that access the shareable EMA area specified by the program.

−060 Code & data assigned to same partition

Both the code partition and the data partition are assigned to the same reserved partition. They must be in distinct partitions. This error applies only to operating systems with VC+ Enhancement Package.

−063 Code assigned to non-existent partition

The code partition of the program is assigned to a reserved partition which is "bad" due to a parity error in the partition or a reserved partition which is undefined. Use the AS command with the "C" option to re-assign the code partition to a good partition or dynamic memory.

−064 Partition too small for code segment

The program's code partition is assigned to a reserved partition which is not large enough to hold it. The code partition must be assigned to a larger reserved partition or dynamic memory.

−068 Code assigned to shareable EMA partition

The program's code partition 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 or dynamic memory. This error could result if the first program that uses that shareable EMA area assigns the area to a reserved partition in which the code partition of a second program that accesses it is assigned to run.

−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 offed; system should be rebooted.

−103 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 disk, or just store the ones you need.

−104 Missing extent

A request was made for a file extent which was missing from the file. The file is probably corrupt. Purge the file.

−105 D.RTR must be sized up

D.RTR uses free space for open flags and global directories, and must be sized up when loaded.

−108 Illegal number of sectors/track

The disk LU being mounted has a defined number of sectors per track greater than 128.

−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 (in which case you should change the protection on the file), or it has a positive security code which needs to be specified correctly in the open call.

−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 Duplicate directory name

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

Disk space or a directory entry occupied by the purged file has been reclaimed, so the file cannot be unpurged. Not repairable.

−211 Directories not on 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 property

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 Disk 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. Superusers do not get this error; become a superuser to avoid this problem.

−217 Bad directory block

Tag fields in the directory do not match, indicating a corrupt disk 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 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.

−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 is busy

FmpRunProgram reports that the program named in the XQ command is busy.

−226 Program 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 run strings. 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 disk 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 FMGR 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 Break flag detected

An FMP routine detected a break sent by the BR command.

−236 You are not a 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 Changed RPL checksum

The program file was linked with a snap file that specified different microcoded instructions (RPLs), or the same instructions in a different order. If the program uses nonexistent instructions, it aborts with the message UI (unimplemented instruction), and the program file changes to prevent this error from being reported again on the current system. (This change to the program file does not solve the problem, but the program may work anyway.)

−241 Can only run unshared

Shared program cannot be run because there is no room in the shared program table. Use LINK to make the program unshared, OF programs to make room in the shared table, or regenerate the system with more shared program entries.

−242 Disk I/O failed

D.RTR got an EXEC error when attempting to access a disk LU.

−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 System can't do CDS

Tried to RP a CDS program on a non-CDS system.

−246 System common changed

Tried to RP a program that defines system common differently than it is defined on the current system.

−247 UDSP not defined

The UDSP is not defined because of one the following reasons:

- None of the entries in the UDSP have been defined.
- The requested UDSP number and entry has not been defined.
- The given UDSP number and entry is beyond the bounds defined for the account.

−248 Invalid directory address found

UDSP tables are corrupt.

−249 SECURITY VIOLATION detected

The Security/1000 subsystem has detected an attempt by the user or program to perform a function for which the user or program has insufficient capability.

−250 D.ERR not available

The system program D.ERR, which is used to generate FMP error messages, cannot be scheduled because D.ERR was not RP'd or it was OF'd. You should RP D.ERR.

−251 Program name exists in another session

An attempt has been made to RP a program in the system session while a program of the same name is already RP'd in another session.

−252 Disk LU is down

D.RTR tried to access a disk LU that is down.

−253 Disk LU is locked

D.RTR tried to access a disk LU that is locked to another program.

−254 No such group

Group name not found by FmpSetOwner.

−255 User is not in group

The user is not in the group specified as the user group parameter for FmpSetOwner.

−256 No such session

From FmpRpProgram.

−257 No such program

From FmpRpProgram.

−258 No SAM for Proto ID

From FmpRpProgram.

−260 Too many symbolic links in path

D.RTR traversed more than 8 symbolic links. This is probably a closed symbolic link loop.

−261 Symbolic link results in illegal path

Symbolic link interpretation yields a path name greater than 63 characters. Symbolic links that contain relative path names can be changed to absolute paths.

−262 Symbolic link must not be remote

An attempt was made to access a symbolic link on a remote system and the remote symbolic link referred to a remote file. Create a symbolic link on the local system that refers to the destination of the remote symbolic link being accessed.

−263 System does not support symbolic links

An attempt was made to create a symbolic link on a system that does not support symbolic links. A CDS version of D.RTR is required and programs accessing symbolic links must be linked with either $SFMP or $SCDS.

−264 Illegal file type

An attempt was made to access a type 12 file from a program or system that does not support type 12 files. The program must be linked with the $UFMP or $UCDS library.

−270 Update time already current

From FmpCopy.

DS Transparency Software

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 Too many sessions

Cannot log on the remote system because too many other sessions are already logged on.

−304 No such account

No user has that name.

−305 Incorrect password

The correct password was not supplied.

−306 Can't access account

A logon error occurred that was not one of the above three.

−307 Transfer is too long

A request was made to DSRTR to transfer more than 1024 words.

−308 Connection broken

The remote system monitor TRFAS was restarted since the connection was open.

DS/1000 Software Errors

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.

Native Language Support Utilities Errors

FMP Errors 401 − 410 are related to native language support utilities. If your system does not have the native language support utilities, the following errors may still occur if the message catalog file on the system for a utility is not of the same revision as the utility itself.

−401 Message number not found in catalog

−402 Message too big for message buffer

B

File Manager (FMGR)

FMGR is a program that manipulates files on FMGR directories. You can use FMGR to perform operations on FMGR cartridges that you cannot perform with CI, such as initializing directories.

FMGR used to be the standard command interpreter just as CI is today. FMGR can run programs and issue control requests in much the same way that CI can, except that FMGR requires commas as parameter separators.

FMGR works only with FMGR files, so it cannot access hierarchical file systems. This appendix discusses only those commands that are useful for manipulating FMGR files. Use CI for other operations.

This appendix covers the following topics:

- FMGR Control: Commands that let you perform FMGR operations.
- Disk Manipulation: Commands to handle FMGR cartridges.
- File Manipulation: Commands to copy, purge, and list FMGR files.
- Transfer Files: Commands to run FMGR non-interactively.
- Device Manipulation: Commands to down a device and change buffer limits.

A table showing other FMGR commands and their CI equivalents is provided at the end of this appendix, as well as a description of the COMND program.

FMGR Control

Using FMGR commands, you can designate a list device, send output to it, define a log device for receiving error messages, limit the system messages, and display explanations of error messages. FMGR control definitions are provided below, and the commands are described in subsequent sections.

List Device

The system list device is the device to which output is directed by default. The list device is often a line printer, but you can use the LL command (described later) to change the designation, directing the output to any device or disk file specified in the command. You may change the assignment of the list device as often as necessary.

The LI (List) command, described later, is used to send output to the system list device.

Log Device

The system log device receives FMGR error messages. It must support input as well as output, since the error messages may require a reply from the operator. The LO command, described later, lets you reassign the FMGR log device.

Severity Code

The severity code acts as a filter to screen out unwanted FMGR output (messages from the system). The SV command, described later, is often used in transfer files.

FMGR Errors

FMGR errors can be either positive or negative. They can be automatically listed to the terminal by specifying a severity code of 1000 or 1001 (see the SV command discussion).

FMGR Control Commands

NO TAG provides a summary of the FMGR commands, which are described in the sections that follow the table, along with command examples.

. FMGR Control Commands

Commands			Description
Information Commands			
??	Explain Error Codes	error#	Provides the meaning of an FMGR error code number
Listing Commands			
LL	Display/Change List Device	namr	Displays or changes list device assignment
LO	Log Device	namr	Assigns a log device to receive FMGR error messages
Configuration Commands			
SV	Display/Set Severity Code	severity	Displays or changes the current FMGR severity code

Explain Error Codes (??)

Purpose: Provides the meaning of an FMGR error code number.

Syntax:
```
?? [,error]
?? [,ALL]
```

error	The FMGR error number. If you do not enter the parameter, an explanation is provided for the most recent FMGR error.
ALL	The error codes and explanations are all listed to the list device.

?? Command Examples

The following examples illustrate the command entry and the system response.

Example 1: Explain the meaning of error code −006.

```
FMGR : ??,-6
FMGR -006 FILE NOT FOUND
```

Example 2: Explain the meaning of error code −102.

```
FMGR : ??,-102
FMGR -102 ILLEGAL D.RTR CALL SEQUENCE.
```

Example 3: List all the FMGR errors.

```
FMGR : ??,ALL
```

The system responds to this entry by sending a complete listing of all the FMGR error codes and their meanings to the list device.

Display or Change List Device (LL)

Purpose: Displays or changes the list device assignment.

Syntax: `LL[,namr]`

namr The new list device. This may be any existing file or LU number.

Description:

Namr should be a device that allows output; if you try to list on an input only device, FMGR terminates and issues the system error code IO07.

Certain FMGR commands (LI, CL, DL, PL, IO, ??) direct their output to the list device. The default list LU is your terminal.

LL Command Examples

Example 1: Change the list device from logical unit 6 (line printer) to logical unit 4 (user terminal).

```
LL,4
```

Example 2: Change the LU back to LU 6.

```
LL,6
```

Example 3: Change the list device to list output to existing file LISTF. Specify CRN to ensure that list is sent to the correct file.

```
LL,LISTF::13
```

Change Log Device (LO)

Purpose: Designates a log device to receive FMGR error messages.

Syntax: `LO[,namr]`

namr — The device to receive the FMGR messages. This may be any existing file or LU number.

Description:

If you do not enter a namr, the LU number of the current log device is displayed.

Namr should be a device that allows input as well as output, as you may need to enter a reply to one or more messages.

Display or Set Severity Code (SV)

Purpose: Displays or changes the severity code currently being used by FMGR.

Syntax: `SV[,severity[,global#][,IH]]`

severity — The new severity code, as follows:

- 0 Display error codes and echo commands on log device (default).
- 1 Display error codes on log device, inhibit command echo.
- 2 Display error code displayed only if error requires transfer of control to log device for correction; no command echo.
- 3 Same as 2.
- 4 If a FMGR command error is encountered, command file continues automatically; no command echo, no transfer to log device, and no command file abort occurs.
- 100x If the above severity codes are added to 100, additional error message information is provided when an error occurs.

global # — The optional G-type global number from 1-9 in which the severity code is placed.

IH — The optional parameter that inhibits echo of command entry.

Description:

If you omit the severity parameter, the current severity code is displayed.

The normal default mode is to echo each command as it is entered on the input device and to log all errors on the same device. During interactive operation, the severity code should be this default value, zero.

When there is no advantage to echoing commands at the console (for example, when commands are entered from a peripheral device or through a file), the severity code can be set to 1.

You can suppress messages that do not require action by setting the code to 2, which terminates any currently executing job when an error occurs.

Whenever you suppress command echo, any command that causes an error is displayed before the error code unless the error display is also suppressed.

SV Command Examples

Example 1: Display error codes and echo commands on the log device.

```
SV,0
```

Example 2: Display error codes and echo command on the log device, and display additional information about errors encountered.

```
SV,1000
```

Disk Manipulation

There are four important points to remember about disk manipulation:

- Disk logical units are logical subdivisions of physical disk drives.
- Cartridges are logical subdivisions of physical disk packs or flexible (floppy) disks.
- The information stored on a disk pack/cartridge is available to the system only when:
  - the disk pack is physically mounted on the disk drive, and
  - the cartridge is logically mounted on the logical unit.
- When a cartridge is mounted on a logical unit, there is an exact correspondence between the cartridge and the logical unit. At other times, the cartridge and the logical unit are independent entities.

The discussion of disk manipulation in this appendix consists of definitions of the various disk manipulation elements, followed by the FMGR commands used in manipulating disks.

Logical vs Physical

When you are managing the information stored on disk files, consider the difference between "logical" and "physical" entities. In general, logical refers to software, and physical refers to hardware. For example:

- Disk packs and disk drives are hardware, thus physical. Disk files and their structure are software, thus logical (they exist only in the memory of the computer).
- When a disk pack is placed on a disk drive, it is being physically mounted. When the operating system (software) is informed that the storage space on the disk pack is available to the system, the disk pack has been logically mounted.

The disk manipulation commands discussed later in this appendix deal with logical operations only.

Disk Logical Units and LU Numbers

The operating system divides the storage capability of a disk drive into one or more logical sections, and assigns a logical unit (LU) number to each section. Each logical unit is treated as a separate disk by the system, even though it may share the physical disk drive with several other LUs. Disk logical units can be of different sizes. There must be at least one disk LU for every disk drive on the system.

Cartridges and Cartridge Reference Numbers

The storage capacity on a disk pack (fixed or removable) or a flexible (floppy) disk can be divided into one or more cartridges. Cartridges are identified by cartridge reference numbers (CRNs), and they are closely related to disk LUs. A CRN is assigned when the cartridge is initialized (initialization is discussed in subsequent sections).

Data stored in disk files can be accessed by referring either to the LU number or to the CRN. The relationship between the disk LU and the cartridge (both of which are logical entities) is the same as that between the disk drive and the disk pack (physical entities). For example:

> Suppose you have two flexible disk drives that are assigned the logical unit numbers LU 13 and LU 14. You also have a flexible disk with a cartridge assigned the cartridge reference number CRN 123.
>
> If cartridge CRN 123 is mounted in disk LU 13, you can read a file from the flexible disk by referring to either LU 13 or CRN 123.
>
> If you move the flexible disk to the other drive and want to address it using the LU number, you now refer to it as LU 14. However, the CRN is still 123.

In other words, the cartridge reference number stays with the data on the flexible disk, while the logical unit number stays with the disk drive. This means that when you address files by the CRN, the LU being used should have the same configuration (see the section below on "Configuration of Logical Units/Cartridges").

CRNs are positive numbers from 1 to 32767. A CRN may also be assigned as two ASCII characters; in this case, the cartridge may be referred to by the numeric equivalent of the ASCII characters, and that number appears as the CRN when you list the cartridge information with the CL or DL commands, described later.

Since LU numbers fall within the range of allowable CRNs, the system needs some way of telling which type of number it is dealing with. In situations that may be unclear, LU numbers are entered as negative numbers and CRNs are entered as positive numbers in the namr parameter of the file descriptor, which is discussed below.

Configuration of Logical Units/Cartridges

Configuration refers to the way disk logical units or cartridges are arranged to cover the storage capacity of a disk drive, disk pack, or flexible disk.

The configuration of logical units assigned to a disk drive depends upon the type of disk drive involved. For example, since flexible disks hold a relatively small amount of information—about one megabyte on a two-sided disk—a single flexible disk LU is usually configured to cover the entire volume.

Flexible disks are portable, and a standard configuration lets you use a flexible disk in any flexible disk drive on the system. You can also use a flexible disk to transport programs and data between two systems. Because of their portability, flexible disks are frequently mounted and dismounted.

Disk packs ("hard" disks) are much larger than flexible disks both in physical size and storage capacity. Because of their large physical size, disk packs are not as easily removed, transported, and stored as flexible disks. In addition, some disk packs are not removable.

Because of their large storage capacity hard disks are almost always divided up into several disk LUs. These LUs seldom have any standard configuration. As a result, hard disks typically stay in one location and are not usually interchanged between drives, either on the same system or on different systems.

Disk LUs are configured during system generation. Cartridges are configured during initialization (initialization is discussed in subsequent sections). Note that a disk pack or a flexible disk can be successfully used on a disk drive only if the configuration of disk LUs on the disk drive is the same as the configuration of cartridges on the disk pack or flexible disk.

Mounting and Dismounting Cartridges

Mounting and dismounting cartridges are logical operations. Before you use the MC (Mount Cartridge) command (described later) to mount a cartridge logically, you must physically mount the disk pack or flexible disk that contains the cartridge on the disk drive. This makes the cartridge available to the system.

When you invoke MC, the system responds by establishing an exact correspondence between the cartridge and the LU on which it is mounted. That correspondence persists until you use the DC (Dismount Cartridge) command (described later) to logically dismount the cartridge; at that time, the LU and the cartridge regain their separate identities.

Note that you may not mount two cartridges with the same CRN at the same time.

The following example illustrates mounting and dismounting cartridges:

You have a flexible disk with its entire storage volume configured as one cartridge, with a CRN of 1234. You also have a flexible disk drive with its full storage capability configured as LU 17. You physically mount the flexible disk by inserting it into the drive; then you logically mount the disk by invoking the following command:

```
FMGR : MC, -17
```

This tells the system that the cartridge mounted in LU 17 is now available for use. The system searches the file directory of the cartridge to find the CRN, and then it establishes an equivalence between LU 17 and cartridge 1234. Directing a command to LU 17 is now the same as directing the command to cartridge 1234. This relationship continues until the cartridge is dismounted, using the DC command:

```
FMGR : DC,-17
```

or

```
FMGR : DC,1234
```

The system severs the relationship between LU 17 and cartridge 1234 and once again considers them separate entities. You may now physically dismount the flexible disk.

Cartridge Directory

The term “cartridge directory” refers to the directory of cartridges that are mounted on the system. The term “file directory” refers to the directory of files on a cartridge. Cartridge directories are described in this section, and file directories are described in the next section, “Cartridge File Directories.”

The cartridge directory is a system table that contains entries for all mounted cartridges and maintains the relationship between mounted cartridges and logical units.

Cartridges are listed in the system table in the order in which they were mounted. The last cartridge mounted is placed at the bottom of the cartridge directory. This is significant if the system is searching for a file but does not know in which cartridge it is located. In this case, the search begins with the first cartridge in the list and continues until the file is found or until all the mounted cartridges are searched.

The order of the cartridge directory is also used if a new file is created but no cartridge was specified to hold it. In this case, the file is placed on the first cartridge in the directory that has enough space to accommodate the new file.

To change the order of a search, you may use the DC and MC commands to dismount and then remount a cartridge. When you dismount the cartridge, it is removed from the directory; when you remount it, the cartridge is placed at the bottom of the directory since it is the newest entry.

Note that you may not dismount a cartridge that contains open files or program files that were restored. If you try to dismount such a disk, an FMGR error message is displayed. However, the cartridge is still moved to the bottom of the cartridge directory.

Cartridge File Directory

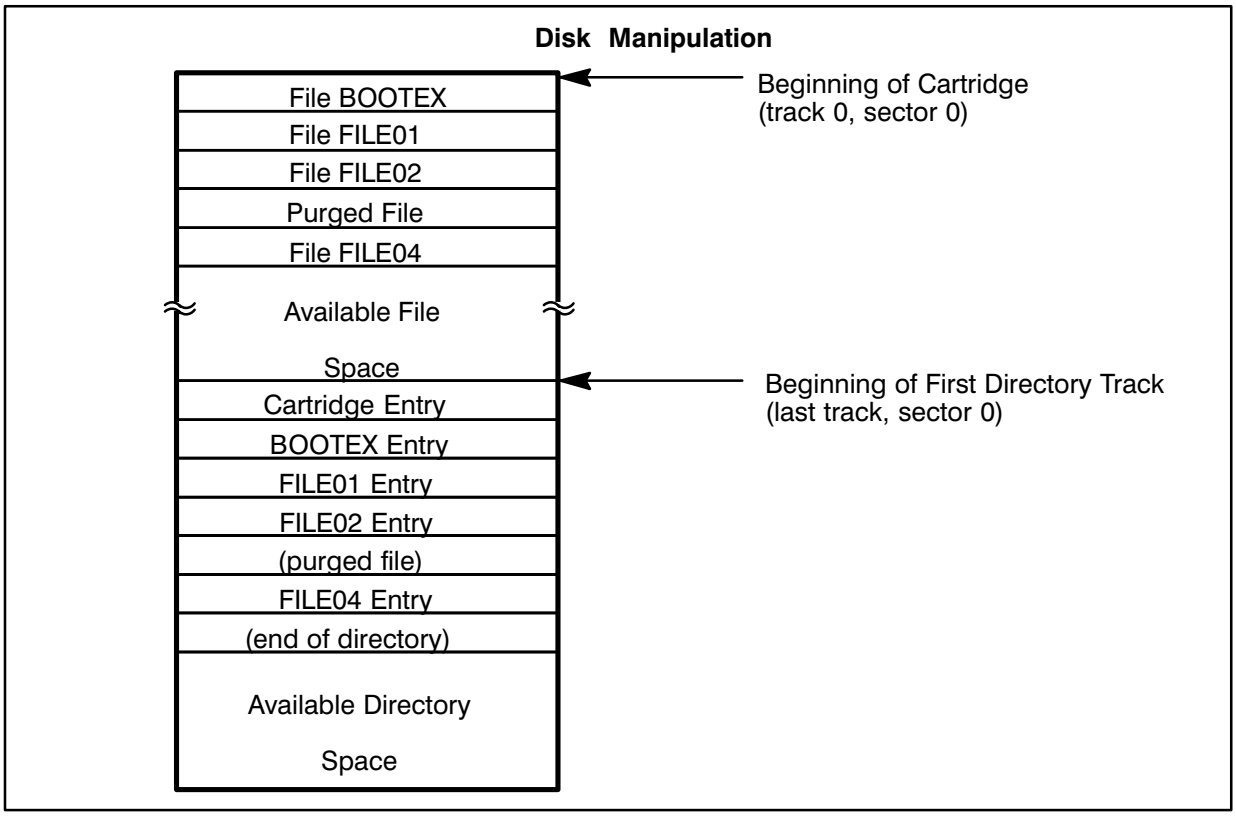
The term “file directory” refers to the directory of files on a cartridge. File directories are described in this section. Cartridge directories are described in the previous section, “Cartridge Directory.”

Files are located at the beginning of cartridges and directory entries at the end of them. Files begin at sector 0 of the logical first track of the cartridge. The first track is set using the IN (Initialize) command (described later), usually at track 0. NO TAG illustrates the cartridge structure.

The system always installs the boot extension file, BOOTEX, as the first file on a cartridge. It is a type 1 file, 768 blocks long.

The directory begins at sector 0 of the last track of the cartridge. If there is more than one directory track, the second directory track begins at sector 0 of the next-to-last track, and so on, moving toward the beginning of the cartridge. The first entry in the directory describes the cartridge, and succeeding entries describe files in their order on the cartridge.

When a new file is added to the cartridge, its size is checked. If it is the same size as a file that was purged, it replaces the purged file and its directory entry. If no files of the same size were purged, the new file is added after the last file, and a new entry is created at the end of the directory.



. Cartridge Structure

Cartridge Initialization

Initializing a cartridge builds or modifies the cartridge header information for a mounted disk cartridge. A cartridge must be initialized (using the IN command, described later) before it can be recognized by the file management system.

If you mount an uninitialized cartridge, the system locks the cartridge and it cannot be used with the File Management Package (including FMGR) until you initialize it.

Every cartridge must be initialized at least once, and may be re-initialized as often as necessary.

At initialization, the cartridge takes on the configuration of the LU on which it is mounted. Several of the parameters that describe the cartridge are specified, including the cartridge reference number, the cartridge label, the number of directory tracks, the location of bad tracks, and others. The parameters are stored in the cartridge entry of the file directory on the cartridge.

Master Security Code

The master security code (msc) controls access to security codes for all the files managed by the file management system. If you specify a master security code at system generation, you must subsequently specify it for new cartridge initialization and then for all re-initializations of the same cartridge. Be aware that this code is never printed or displayed by the system, so you must be particularly careful to remember it. The IN command can be used to change the master security code for the system.

Note that when the system is re-booted, the master security code reverts to the value set during system generation.

Re-initializing a Cartridge

You can re-initialize a cartridge in order to change the parameters in the cartridge entry, as often as desired.

Since FMGR locks a cartridge during initialization, make sure that all the files on the cartridge are closed and that all temporary (disk-resident) programs have released their ID segments before re-initialization. Otherwise, the cartridge cannot be locked, and the IN command is not executed.

During re-initialization, be careful about changing the first track parameter. If the first track number is increased, the system purges all files on the cartridge. (For more details, see "Purging All Files," below.) If the first track number is decreased, more track space is added to the cartridge, but the system is unable to use that space until the cartridge is packed with the PK command (described below).

If the starting track is decreased to zero, the system purges all files on the cartridge and installs the BOOTEX file at the beginning of the starting track. The starting track number is not changed unless you want to purge all files, as it is a waste of disk space to assign a starting track other than track zero.

Be careful when you change the number of directory tracks during re-initialization. More space is gained for file entries in the directory by increasing the number of directory tracks, as long as the newly added directory tracks don't use track space that is already assigned to existing files.

If the new directory tracks conflict with existing files, or if you decrease the number of directory tracks, the system purges all of the files on the cartridge. The system does give you another chance to say NO before the files are purged. See the next section for more detailed information about purging files.

Purging Files on a Cartridge During Re-initialization

A cartridge is purged of all files if you re-initialize the cartridge and any of the following changes occur:

- The first track number is increased.
- The first track is decreased to zero.
- The number of directory tracks is decreased.
- The number of directory tracks is increased and there is not enough room left on the cartridge for the new track(s).

In interactive mode, the file manager issues an FMGR 060 message in response to any of these occurrences. If this message displays, you must respond by entering YES or NO; otherwise, the message simply displays again.

If you answer NO, the IN command aborts. If you answer YES, or if the command is in a transfer file, the files are purged as part of the re-initialization. After the files are purged, the BOOTEX file is created at the beginning of the first track.

The following two command sequences purge all files from a cartridge. Assume that the cartridge was last initialized with the command given in the previous example.

```
FMGR  :  IN,SC,JS,JS,SORCE3,1,2
FMGR 060 DO YOU REALLY WANT TO PURGE DISK? (YES OR NO).  YES

FMGR  :  IN,SC,JS,JS,SORCE3,0,2
```

The first IN command increases the first track number, causing the system to try to purge all files on the cartridge. Before the files are purged, the system checks to see if you really want to purge the files by issuing the FMGR 060 message. If you respond by entering YES, the system proceeds to purge all the files.

The second IN command makes logical track 0 the first track of the cartridge. BOOTEX is then installed on that track.

```
FMGR  :  IN,SC,JS,JS,SORCE4
FMGR 060 DO YOU REALLY WANT TO PURGE DISK? (YES OR NO).  YES

FMGR  :  IN,SC,JS,JS,SORCE4,,2
```

In this command sequence, all the files on the cartridge are purged when the number of directory of tracks is decreased. The first IN command uses default values of 0 for the first track and 1 for the number of directory tracks. If you respond YES to the FMGR 060 message, the system purges all the files, reinstalls BOOTEX on track 0, and re-establishes the directory on a single track.

The second IN command increases the number of directory tracks to 2, the original number.

Packing a File Cartridge

When files are purged from a cartridge, they leave gaps in the disk space. The PK command lets you “pack” a file cartridge, which eliminates such gaps and allows more efficient use of the disk space.

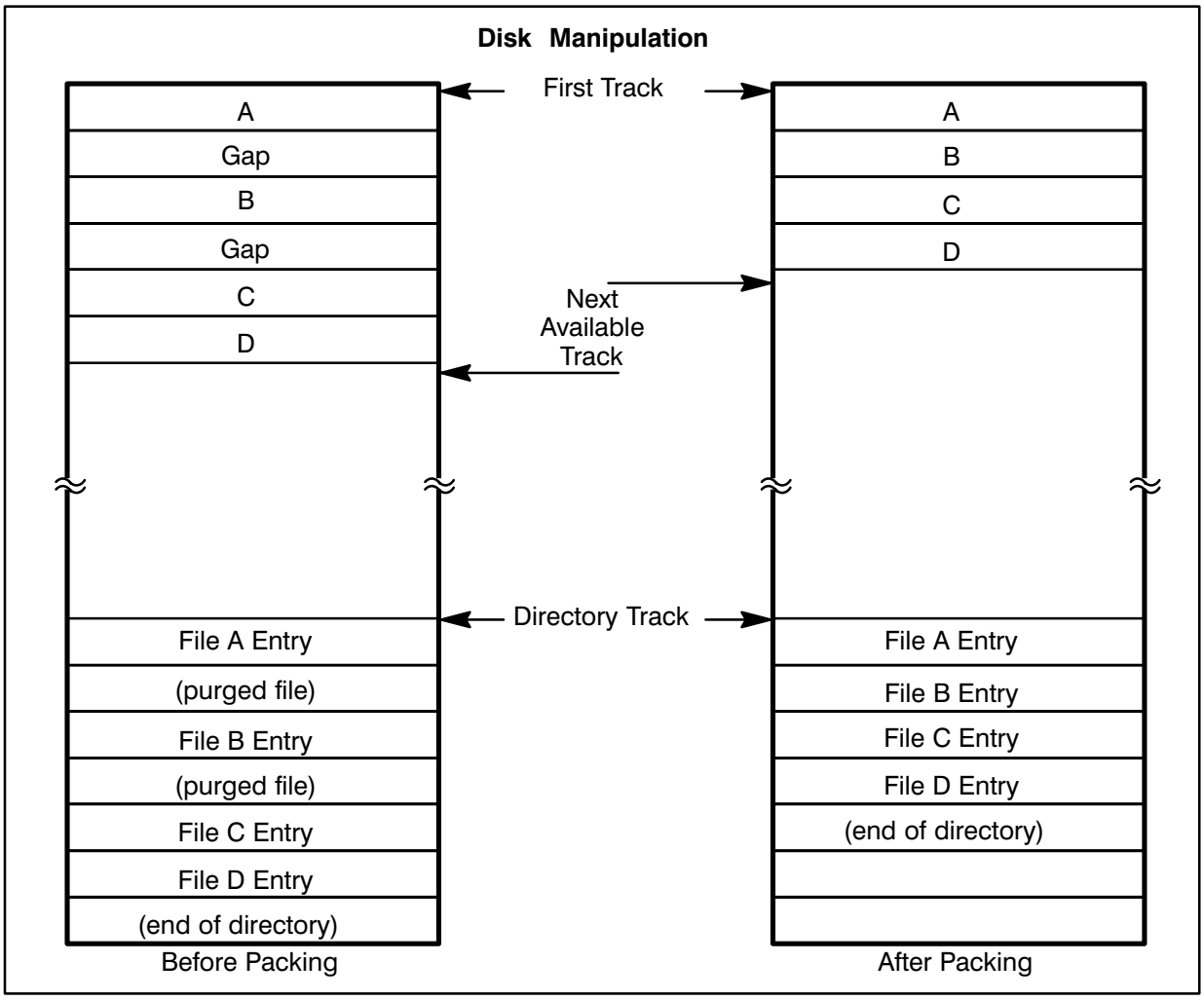
When PK executes, it moves files into the empty spaces left by purged files wherever possible. After a file is packed, its directory entry is updated. After all the files are packed, the directory is packed and the entries for purged files are removed.

Note that the PK command purges all scratch files (scratch files are discussed later in this appendix). If you want to save the information in a scratch file, either rename the file or store the information in another file.

The PK command also purges files that contain tracks reported as bad by the IN command. If you want to save such a file, use the ST command to store it on another cartridge.

For more details, refer to the discussion of the PK command later in this appendix.

NO TAG shows the structure of a disk before and after the PK command is executed.



. Disk Structure Before and After Packing

Transferring Files Between Disk Cartridges

All or selected files residing on a mounted disk cartridge can be copied or moved to another mounted disk cartridge with the CO command. Using the CO command, you can perform the following operations:

- Copy or move the contents of an entire disk cartridge to another disk cartridge.
- Copy or move selected files from one disk cartridge to another disk cartridge.
- Back up one disk cartridge to another (or several) disk cartridge(s).
- Merge many cartridges onto one cartridge (restoring from a backup).

The CO command and its options are described in subsequent sections.

Disk Manipulation Commands

The sections that follow describe the commands you can use for manipulating disks, including listing the cartridge directory, listing the cartridge file directory, mounting and dismounting cartridges, initializing cartridges, packing a file cartridge, and transferring files between mounted cartridges. NO TAG summarizes the commands.

. Disk Manipulation Commands Summary

Commands	Description
Listing Commands	
CL List Cartridge Directory	Lists all mounted cartridges in the cartridge directory
DL List File Directory	Lists some or all the entries in any or all of the file directories on a cartridge
Configuration Commands	
DC Dismount File Cartridge	Dismounts a file cartridge logically, removing the entry for the cartridge from the cartridge directory
IN Initialize File Cartridge	Builds or modifies the cartridge header information for a mounted disk cartridge, and used to assign or change the master security code
MC Mount File Cartridge	Mounts a file cartridge logically and creates an entry for the cartridge in the cartridge directory
PK Pack File Cartridge	Moves files together on a cartridge to eliminate gaps left from purged files
File Transfer Command	
CO Transfer Files Between Cartridges	Copies or moves selected files between mounted cartridges

List Cartridge Directory (CL)

Purpose: Lists all cartridges in the cartridge directory. Only mounted cartridges appear in the directory.

Syntax: `CL`

Description:

When you use the CL command, the cartridge directory list is sent to the designated list device. You may change the list device with the LL command, described earlier in this chapter.

Cartridges are listed in the directory in the same order that is used for all commands that access the dis (such as CL, DL, ST, DU, PU, RN, LI, RU, and XQ commands).

CL Command Example

The following example illustrates the use of the CL command:

```
FMGR : CL
       LU         LAST TRACK     CR       LOCK

       12          0405         00012
       13          0199         00013
       14          0199         00014
       17          0199         00017
       18          0199         19523
```

LU	The logical unit number of the cartridge.
LAST TRACK	The highest track available on the cartridge.
CR	The cartridge reference number.
LOCK	The name of the program locking the cartridge; blank if not locked.

Copy Files (CO)

Purpose: To copy files from one disk cartridge to another.

Syntax: `CO,cartridge1,cartridge2[,options[,name1[,name2[,msc]]]]`

or

`CO,namr1,cartridge2[,options[,name1[,name2[,msc]]]]`

cartridge1	The source cartridge identifier (negative LU number or positive CRN).
namr1	The file name and optionally, security code, cartridge identifier (negative LU number or positive CRN), file type, and file size. The namr may contain wildcards. See the discussion of the namr parameter in the section “File Manipulation”.
cartridge2	The destination cartridge identifier (negative LU number or positive CRN).
options	The CO command options are described below.

name1	The starting file name (first file to be copied to the destination cartridge). If specified, it must reside on the source cartridge or files are not copied.
name2	The ending file name (last file to be copied to the destination cartridge). If specified, it should reside beyond name1 in the source cartridge file directory. See the comments in the examples below.
msc	The master security code; must be two ASCII characters. The msc is used in conjunction with the Purge and/or Clear options, described below.

CO Command Options

The five CO command options provide additional control over the transfer of files.

You may enter single or multiple options, in any order, in the third parameter of the CO command. Options are entered without delimiters.

Note that neither the C option nor the P option removes the file BOOTEX from the source or destination cartridge. The D and P options do not purge any type 6 or active file.

NO TAG summarizes the options, which are discussed below.

. CO Command Options Summary

Option	Description
C	Clear destination cartridge before copying files
D	Dump mode (store file even if it exists on destination cartridge)
E	Eliminate extents on copied files
P	Purge source files after copy
V	Verify that files are copied correctly

Clear (C)

This option clears the destination cartridge before any files are copied to it (this is similar to using the IN command, discussed later, except that any existing BOOTEX file is not removed).

If you specify this option from an interactive terminal, an FMGR 060 message requiring a YES or NO response is displayed. (If you enter anything else, the CO command displays the message again.) If you reply NO, the CO command aborts. If you answer YES, the destination cartridge is cleared of all files (except BOOTEX). You must correctly specify the msc parameter to use this option; a missing or incorrect msc causes an FMGR −51 error.

Dump (D)

When you specify the Dump option, the normal copy process still takes place. When the program encounters a duplicate file name on the destination cartridge, it tries to purge that file and replace it with the file on the source cartridge. Purging and replacing the file only occurs if the security code on both files match; otherwise, an FMGR−007 error occurs, leaving the destination file unchanged and the source file uncopied.

Eliminate Extents (E)

The E option eliminates extents from the resulting destination files. A destination file's size is calculated from the source file's size and the number of extents used.

Purge (P)

This option purges selected files from the source cartridge as they are copied to the destination cartridge. This results in "moving" files from the source to the destination cartridge. To purge a file, the security code of the file must match the security code in the subparameter of namr1. Alternatively, you may specify the msc parameter to have files purged regardless of their security codes. The purge option does not affect whether a file is copied or not.

Verify (V)

The Verify option compares the source and destination files after the copy to make sure the transfer was done correctly. If the files differ, CO terminates with an FMGR−049 error.

CO Command Option Examples

Example 1: Copy files in dump mode, and verify their transfer.

```
CO,DV
```

Example 2: Clear the destination cartridge first, and purge the source files after copying.

```
CO,CP
```

Example 3: Clear the destination cartridge first, copy files in dump mode, eliminate any extents, verify their transfer, and purge the files after copying.

```
CO,CDEVP
```

CO Command Examples

The following four examples of CO command formats show how to select files to copy to the destination cartridge.

Example 1: **Specify cartridge1. With this form of the command, all the files on the source cartridge are selected (no mask is specified). The starting and ending files are selected by the name1 and name2 parameters (see Example 4).**

```
CO,10,20,,&FIRST,&LAST
```

Example 2: **Specify namr1 without wildcard characters. With this form of the command, only the file specified on the source cartridge is selected. The name1 and name2 parameters are not used with this command format.**

```
CO,&PROG::10,20
```

Example 3: **Specify namr1 with wildcard characters (mask specified). With this form of the command, those files matching namr1 (see the discussion of the namr parameter in the "File Manipulation" section) on the source cartridge are selected. The starting and ending files are selected by the name1 and name2 parameters (see Example 4).**

```
CO,&-----::10,20
```

Example 4: **Specify name1 and/or name2 in conjunction with one of the above forms of the CO command. These parameters restrict the search of the source cartridge. Only those files that reside between name1 and name2 (inclusive) on the source cartridge can be copied. The beginning of the cartridge is assumed if name1 is not specified. The end of the cartridge is assumed if name2 is not specified, or if it doesn't reside on the source cartridge.**

```
CO,&-----::10,20,,&FIRST,&LAST
```

If you specify two ASCII characters for the source CRN, the system interprets them as a namr (see Example 2 above). To specify an entire cartridge, you must use the LU number of the cartridge, or use the −−−−−−::CRN format (shown in Example 3 above) of namr1.

In the following three examples, files &A, %A, &B, %B, &C, %C, &D, %D reside on cartridge 10; BOOTEX, %C, &C reside on cartridge 20; and cartridge 30 is empty.

Example 5: Copy all files on cartridge 10 to cartridge 20.

```
FMGR  : CO,10,20
&A                          (Cartridge1 is 10; cartridge2 is 20)
%A                          (The file names are printed out as they are copied)
&B
%B
&C
FMGR-002                    (Message indicates that these files were not copied)
%C
FMGR-002
&D
%D
FMGR  :                     (Ready for next command)
```

Example 6: Copy all files on cartridge 10 starting with the file &B and stopping with file &D to cartridge 30. Verify the copies.

```
FMGR  : CO,10,30,V,&B,&D
&B                          (Cartridge1 is 10; cartridge2 is 30; option is verify; first file is
%B                          &B; last file is %D.)
&C
%C
&D
%D
FMGR  :                     (Ready for next command)
```

Example 7: Move all program source files (&) to cartridge 20 and all relocatable files (%) to cartridge 30. Replace files with the same name on the destination cartridge (must supply the master security code if selected files have non-zero security codes).

```
FMGR  : CO,&----::10,20,PDV,,,SC
&A                          (Namr1 selects all files beginning with "&" on cartridge 10;
&B                          cartridge2 is 20; options are purge, dump and verify (note
&C                          that &C was replaced); master security code is SC.)
&D
FMGR  :

FMGR  : CO,%----::10,20,PDV,,,SC
%A                          (Namr1 selects all files beginning with "%" on cartridge 10;
%B                          cartridge2 is 20; options are purge, dump and verify.  Ready
%C                          for next command.)
%D
FMGR  :
```

Example 8: Back up cartridge 40 onto a set of floppy disks (cartridges 10, 20, 30).

```
FMGR : CO,40.10.CV
FILEA                       (Cartridge1 is 40; cartridge2 is 10; options are clear
FILEB                       destination and verify file integrity.)
.
.
.
FILEK
FMGR-033                    (Disk is full)
FMGR : DP,10G               (Display next file name)
FILEK
FMGR : CO,40.20.CV,10G      (Repeat above for next floppy)
FILEK
.
.
.
FILEQ
FMGR-033                    (Disk is full)
FMGR : CO,40.30.CV,10G      (Repeat above for last floppy)
FILEQ
.
.
.
FILEZ
FMGR :                      (Ready for next command)
```

CO Command Termination

The CO command terminates in the following circumstances:

- The selected files were correctly copied to the destination cartridge. This is a normal termination. The 1P global parameter is set to 0 to indicate a successful termination.
- A file copy was not verified correctly. If you specified the Verify option and a file did not copy correctly, the CO command terminates. An FMGR−049 error is generated, and the file name is placed in the 10G parameter.
- The destination cartridge runs out of file space or directory space. If a selected file cannot be copied to the destination cartridge for lack of space, the CO command terminates with an FMGR−014 or FMGR−033 error. The file name is placed in the 10G global parameter.

If an FMGR−049 error occurs, you may restart the CO command to the same destination cartridge by specifying the 10G global parameter for the starting file name. To display the file name, enter DP,10G from FMGR (shown in Example 8, above).

Dismount File Cartridge (DC)

Purpose: Dismounts a file cartridge logically, removing the entry for the cartridge from the cartridge directory.

Syntax: `DC,cartridge`

cartridge — The cartridge identifier (negative LU number or positive CRN; a CRN may be an integer or two ASCII characters).

Description:

Dismounting a cartridge makes the cartridge unavailable to the system. It should be done for all file cartridges (logical units) on a physical disk—"hard" disk or flexible ("floppy") disk—before the disk is physically removed from the disk drive.

When you dismount a cartridge, you can reference it by using its LU number, a CRN that is an integer, or a CRN that uses the two-character ASCII equivalent. For example:

```
FMGR : DC,-10      (Dismounts a cartridge using its LU number)

DC,21587           (Dismounts a cartridge using an integer CRN)

DC,TS              (Dismounts the same cartridge using the two-character ASCII
                    equivalent)
```

When you dismount a cartridge, it is locked and its entry is removed from the system. If any program files (type 6) on the cartridge are currently occupying ID segments, the system cannot lock the cartridge and issues an FMGR 038 error (see "DC Error Handling," below).

Dismounting a cartridge can be used to change the order of a cartridge directory. That technique and its usefulness are discussed in the earlier section, "Cartridge Directory."

If FMGR discovers that a cartridge directory is corrupt (the directory is not internally consistent), the cartridge is dismounted as long as it contains no program (type 6) files or swap files. If there are program or swap files on the cartridge, the cartridge remains mounted, it is locked to the copy of FMGR that discovers the corruption, and an FMGR −103 error ("disk directory corrupt") is issued.

DC Command Error Handling

FMGR 038 ATTEMPT TO REMOVE ACTIVE TYPE 6 OR SWAP FILE

There are probably some program files (type 6) on your file cartridge that currently occupy ID segments. FMGR lists the offending programs for you. You can release the ID segments by entering the OF, program, DR command for each program, then DC again.

FMGR −103 DISK DIRECTORY CORRUPT

The file management system found that the information in the directory was not internally consistent. An FMGR −103 error was issued and the cartridge was locked to the copy of FMGR that found the error (that is, your copy). Even though the cartridge is locked, you can still resolve the problem by copying the files to a new cartridge with the CO command.

FMGR −008 LOCK REJECTED

The cartridge is in use by another program. Reissue the command when it is free.

List File Directory (DL)

Purpose: Lists some or all the entries in any or all of the file directories on a cartridge.

Syntax:

```
DL,,[master security]
```

or

```
DL,namr[,master security]
```

or

```
DL,cartridge[,master security]
```

namr	The file name and, optionally, security code and cartridge identifier (negative LU or positive CRN.) Any subparameter may also be specified. You may include wildcards in the namr parameter.
cartridge	The cartridge identifier (a negative LU number or positive CRN. If you specify a CRN, it must be an integer, not ASCII characters. If you omit the parameter or enter zero, the directories of all the mounted cartridges are listed.
master security code	The code assigned to the system at system generation. If this is specified correctly, the listing is in the long form; otherwise, the short form is used.

Description:

Each cartridge has on its last tracks a directory that describes the cartridge and all the files in the cartridge. The DL command can be used to list some or all of the entries in any or all of the file directories. Output goes to the list device.

You may enter

```
FMGR : DL
```

to list the directories for all mounted file cartridges; or

```
FMGR : DL,namr
```

to list cartridge and file information on file “namr” for any directory that contains that file; or

```
FMGR : DL,cartridge
```

to list the directory for the specified cartridge. An optional master security code parameter can be specified for all three command forms.

If two ASCII characters are specified for the CRN, the system interprets them as a namr. To list a specific cartridge, use a negative LU number or a positive CRN in integer form.

DL Command Examples

Example 1: Short form listing

```
FMGR : DL,21587

CR=21587
ILAB=TRAV NXTR=0005 NXSEC=036 #SEC/TR=060 LAST TR= 0133 #DR
  TR=01

NAME   TYPE #BLKS/LU OPEN TO

BOOTEX 00001 00001
&STH1  00003 00002
&STH2  00003 00002
%STH1  00005 00002
%STH2  00005 00002
TEST1  00006 00030
TEST2  00006 00030
JLIST  00003 00001 FMGR –
JLIST  00003 00001 +001
JLIST  00003 00001 +002

CR       =  cartridge reference number
ILAB     =  cartridge label
NXTR     =  next available track
NXSEC    =  next available sector
#SEC/TR  =  number of 64-word sectors per track
LAST TR  =  last track on this cartridge
#DR TR   =  number of directory tracks on this cartridge
NAME     =  file name
TYPE     =  file type
#BLKS/LU =  number of blocks in file or LU number of type 0 file
OPEN TO  =  a file extent number, if preceded by a plus (+) sign; or the
            name of the program to which the file is open (a minus (–)
            sign following the program name indicates that the file is
            open to that program exclusively)
```

Example 2: Long form listing

```
FMGR : DL,-5,DX
CR=21587
ILAB=TRAV NXTR=0005 NXSEC=040 #SEC/TR=060 LAST TR= 0133 #DR TR=01

NAME   TYPE #BLKS/LU SCODE TRACK SEC  OPEN TO
BOOTEX 00001 00001  -32767  0000 000
&STH1  00003 00002   00000  0000 002
&STH2  00003 00002   00000  0000 006
%STH1  00005 00002   00000  0000 018
%STH2  00005 00002   00000  0000 022
TEST1  00006 00030   00000  0001 034
TEST2  00006 00030   00000  0002 034
JLIST  00003 00001   00000  0005 034
JLIST  00003 00001   00000  0005 036  +001
JLIST  00003 00001   00000  0005 038  +002

SCODE     =  security code of file
TRACK SEC =  track and sector address of start of file
```

Example 3: The general forms of the DL command

```
DL,222            (List all files on cartridge 222)
DL,-33            (List all files on disk LU 33)
DL,A              (List all files whose name is A)
DL,A-----         (List all files whose name starts with A)
DL,--A--A:-5:2    (List all files whose 3rd and 6th name characters are A, whose security
                    code is −5 or 5, and which are in cartridge 2)
DL,------::::1    (List all files with length of 1 block)
DL,------:::::16  (List all files with record length of 16 words)
```

Initialize File Cartridge (IN)

Purpose: Builds or modifies the cartridge header information for a mounted disk cartridge. IN may also be used to assign or change the system's master security code.

Syntax:
```
IN,[master security],cart,crn,label[,first track[,#dir tracks]]
```

or

```
IN,master security--new security
```

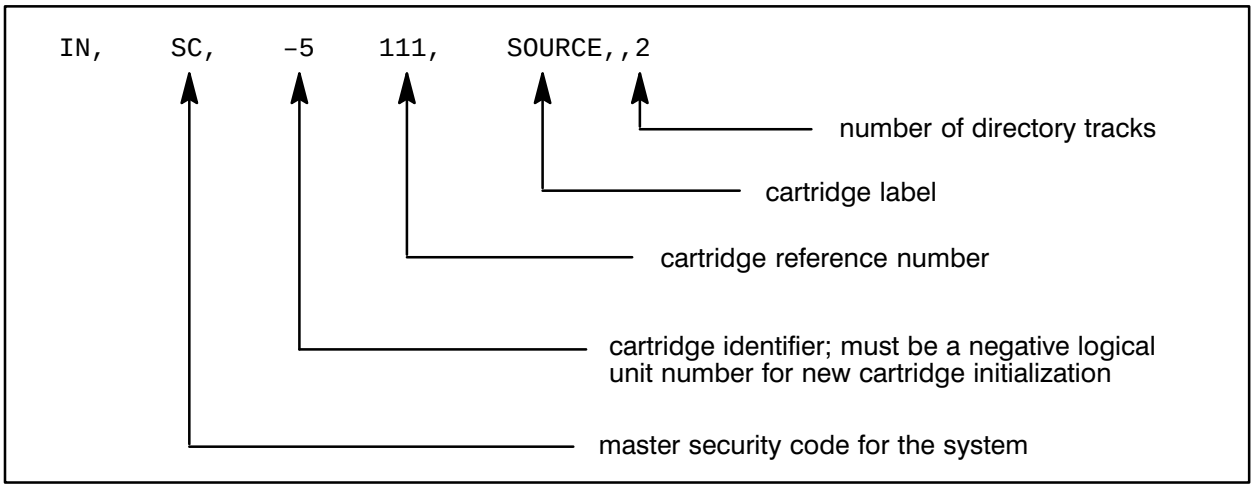
(The first format initializes a cartridge or changes the description of an initialized cartridge. The second format changes the master security code of the file system. This reverts to the original code when the system is re-booted.)

master security code (msc)	The security code that governs access to certain FMGR commands that can adversely affect the file system. This must be two ASCII characters. If they are omitted, directory and file security codes can be accessed with any security code or with none.
cart	The cartridge identifier that if positive, specifies the cartridge reference number; if negative, it specifies the logical unit number. It must be negative the first time the cartridge is initialized.
crn	The cartridge reference number that identifies the cartridge. It must be a positive integer from 1 through 32767 or two ASCII characters.
label	The cartridge label is a string of up to six ASCII characters. The same restrictions that apply to file names apply to the cartridge label (see the discussion of the namr parameter in the "File Manipulation" section in this chapter). The cartridge label is used to identify the contents of the cartridge; the system does not use it in any way. This label shows up in the heading for each cartridge listed by the DL command.
first track	The first FMP track on a cartridge, a positive integer. If this is omitted, track 0 is assumed.
#dir tracks	The number of directory tracks used by the file directory on the cartridge, a positive integer from 1 through 48 for a 96 sector/track LU, or 1 through 36 for a 128 sector/track LU. If it is omitted, one track is assumed. Note that using a value that is divisible by 7 will result in directory corruption if the disk is mounted as an FMGR disk. (Refer to the *RTE-A System Generation and Installation Manual* for more information.)

Description:

The information that you supply when you invoke the IN command is used to build or modify the cartridge entry in the directory for that cartridge. You must use IN to initialize a cartridge before it is recognized by the file management system. Subsequently, you may re-initialize a cartridge, changing the parameters specified as often as necessary.

The command for initializing a new cartridge is illustrated below.



In the example above, the first track parameter is omitted, causing it to be set to zero. This means that the first file in the cartridge starts at logical track zero, sector zero. Except when using the IN command to purge an entire cartridge, there is no reason to specify any first track other than zero; using a non-zero first track wastes disk space.

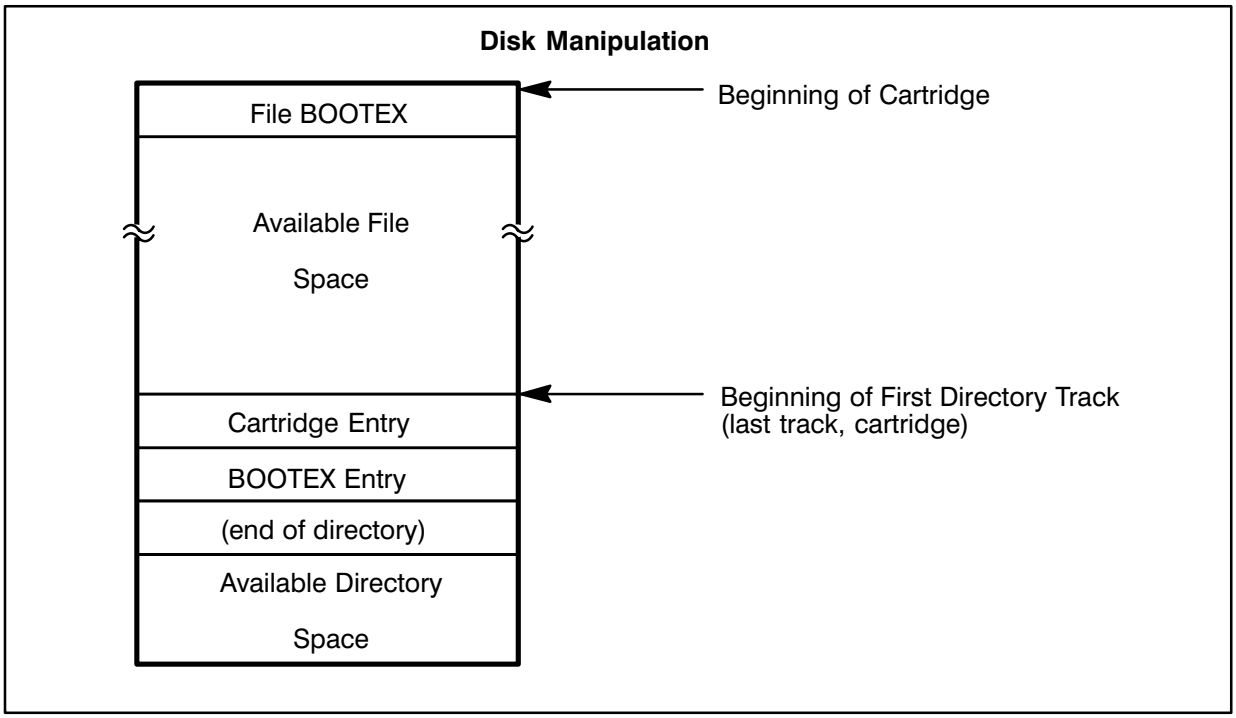
The number of directory tracks specified depends upon the number of files on the cartridge. Floppy (flexible) disks, as used by the HP 7902 disk drive, hold 240 directory entries per track. Among hard disks, HP 7906 and 7920 hold 384 directory entries per track, and HP 7925 holds 512 directory entries per track.

Each directory has a cartridge entry, an end-of-directory entry, plus one entry per file. The first directory track is the last track on the cartridge, the second directory track is the next-to-last track, and so on.

The parameters for the number of sectors per track are also omitted in the example above. You can omit the number of sectors per track and let the system supply the value. Specify 60 for a flexible disk (HP 7902), 96 for HP 7906 and 7920 disks, and 128 for an HP 7925 disk.

As part of the initialization, the boot extension file BOOTEX is created as the first file of the cartridge. BOOTEX is required if the cartridge is to be used for booting up the system. (See the *RTE-A System Generation and Installation Manual* for more details on BOOTEX and its use.) If you do not want BOOTEX on your cartridge, you can use the PU (Purge) command to purge the file.

NO TAG shows the structure of the initialized cartridge.



. Initialized Cartridge Structure

Note that if FMGR finds that the directory of a cartridge is not internally consistent, the program locks the cartridge to the copy of FMGR that discovered the discrepancy and issues a FMGR −103 error (“disk directory corrupt”).

IN Command Error Handling

FMGR 060 DO YOU REALLY WANT TO PURGE THIS DISK? (YES OR NO)

This is not an error, just a question. You entered an IN command that the system interprets as a request to purge all files on the cartridge. If you answer YES, the files are purged; if you answer NO, the command is aborted. For more information about purging files, see the section called "Purging All Files," later in this chapter.

FMGR −103 DIRECTORY IS CORRUPT

The file management system found that the information in the directory was not internally consistent. An FMGR −103 error was issued and the cartridge was locked to the copy of FMGR that found the error (that is, your copy). Even though the cartridge is locked, you can still resolve the problem by copying the files to a new cartridge with the CO command.

FMGR −008 FILE IS ALREADY OPEN

The cartridge is in use by another program. Re-issue the command when it is free.

Mount File Cartridge (MC)

Purpose: Mounts a file cartridge logically, creating an entry for it in the cartridge directory and making it available to the file management system.

Syntax: `MC,lu[,last track]`

lu	The logical unit number of a disk. It may be a positive or negative number, but not a cartridge reference number.
last track	The last track on the cartridge available to the file management system. The default is the last track defined for the disk LU at system generation time.

Description:

The lu parameter may be positive or negative (see example below), but it must refer to a disk logical unit. It cannot be a cartridge reference number. (The use of a cartridge reference number assumes an existing association between a logical unit and a cartridge reference number; it is the MC command that establishes that association.) For example, either of the following commands makes an entry in the cartridge directory for disk logical unit 10:

```
FMGR : MC,10
FMGR : MC,-10
```

You must both mount and initialize a cartridge (using the IN command, discussed in subsequent sections) before you can use it. Typically, a cartridge may be mounted and dismounted frequently, but initialized only once.

When MC is executed, an entry is established for the cartridge at the bottom of the cartridge directory (see the section "Cartridge Directory," below). When you mount a cartridge, if FMGR discovers that the directory of a cartridge is not internally consistent, the cartridge is locked to the

copy of FMGR that discovers the discrepancy and the FMGR −103 error (“disk directory corrupt”) is issued.

MC Command Error Handling

FMGR 012 DUPLICATE DISK LABEL OR LU

You probably tried to mount an already mounted cartridge. Either the LU number or the CRN is already entered in the cartridge directory. You can check the contents of the directory with the CL (List Cartridge Directory) command.

FMGR −018 ILLEGAL LU. LU NOT ASSIGNED TO SYSTEM.

You tried to mount a disk LU that is not recognized by the system. Make sure you are not trying to mount a cartridge by its CRN; you must use the LU number. If this is not the problem, check the I/O tables with the IO command to make sure that the LU you want to mount is actually a disk LU.

FMGR −103 DISK DIRECTORY CORRUPT

The file management system found that the information in the directory was not internally consistent. An FMGR −103 error was issued and the cartridge was locked to the copy of FMGR that found the error (that is, your copy). Even though the cartridge is locked, you can still resolve the problem by copying the files to a new cartridge with the CO command.

FMGR −008 LOCK REJECTED

The cartridge is in use by another program. Reissue the command when the cartridge is free.

Pack File Cartridge (PK)

Purpose: To move files together on a cartridge to eliminate gaps left from purged files.

Syntax: `PK[,cartridge]`

cartridge	The cartridge identifier (negative LU or positive CRN) of the cartridge to be packed; for example, PK,−10. If you omit the cartridge identifier, all mounted cartridges are packed.

Description:

The PK command locks a file cartridge before packing it. If the cartridge cannot be locked because files on it are open, the cartridge is not packed. If there are active program (programs that have been restored with the RP command) files (type 6) on the cartridge, the cartridge is packed beyond the last active program file.

Be aware that when you issue a PK command within a transfer file, if the transfer file crosses a sector boundary during PK execution, succeeding commands may be read from the wrong area of the disk. To prevent this problem, create a small (under one sector) transfer file that contains a PK command only. Then invoke the small transfer file either by itself or from within another.

If FMGR discovers that the cartridge is corrupt (the directory is not internally consistent), the cartridge is locked, it is not packed, and an FMGR −103 error displays.

PK Command Error Handling

FMGR −103 DISK DIRECTORY CORRUPT

The information in the directory is not internally consistent. An FMGR −103 error is issued and the cartridge is locked to the copy of FMGR that found the error (that is, your copy). Even though the cartridge is locked, you can resolve the problem by copying the files to a new cartridge with the CO command.

FMGR −008 LOCK REJECTED

The cartridge is in use by another program. Reissue the command when the cartridge is free.

File Manipulation

File manipulation refers to the operations performed on FMGR files such as copying, purging, listing, and so on. The sections that follow provide information about disk files and the various commands that you can use to perform file manipulation.

Records and File Types

Disk files are composed of records of fixed or variable length, written in binary or ASCII code. The nature of the records determines the file type, as shown in NO TAG below.

. Record and File Type Equivalences

Category	Type	Description
Device	0	Non-disk (device) file
Fixed length, random access, not extendable	1	Fixed-length (128 words) records
	2	Fixed-length (user-defined) records
Variable length, sequential access, automatic extents	3	Variable-length records, any data type
	4	Program source file, ASCII
	5	Program object file,relocatable binary
	6	Executable file, memory-image code
	7	Absolute binary
	8 to 99	User-defined data format
	32767	Reserved

For example, the following frequently used programs produce the listed output file types:

Program	Output File type
RTAGN	1
EDIT	4
FTN7X	5
MACRO	5
LI	6

Scratch Files

Scratch files are produced by a call to the CRETS routine. They provide temporary disk space for any program that needs a scratch area during its execution. When a program finishes using scratch files, a call to the PURGE routine eliminates them. Scratch files are also purged by the PK command. (The CRETS and PURGE routines are described in the *RTE-A Programmer's Reference Manual*.)

Accessing a Disk File

Disk files are created and accessed through the file descriptor (FD) namr parameter. When it refers to a disk file, the namr parameter is equivalent to:

```
FD = filename[:sec[:cart[:file type[:file size[:record size]]]]]
```

Refer to the file using the form:

```
filename
```

or

```
filename::cartridge
```

If you use the first form, the cartridges are searched in the order listed in the cartridge directory. The second form restricts the file search to the specified cartridge.

For example,

```
FMGR : LI,SOURCE::AA
```

searches cartridge AA only for the SOURCE file and lists it when it is found.

The command form

```
FMGR : LI,SOURCE
```

starts searching the first cartridge in the cartridge directory and continues through the mounted cartridges in order of the cartridge directory until a file named SOURCE is found. This may be the file you want, or it may be another file named SOURCE on a different cartridge.

Specifying the cartridge along with the file name ensures the correct file is found, and provides the fastest search, since the system goes directly to the file directory of the named cartridge.

Creating a File

When a disk file is created, an entry is made in the file directory on the cartridge to which the file is allocated. If a file of the same length as the new file was purged, the new directory entry replaces that of the purged file.

The file is created on the cartridge you specify, or, if you do not name a cartridge, on the first mounted cartridge with enough space to hold the file. If a file with the same name already exists on the cartridge, the new file is not created.

If the file is type 3 or greater, an EOF mark is written at the beginning of the file. As data are added to the file (with the DU command, described later), the file mark is moved to the end of the data.

If you create a file using a negative file size subparameter, the file occupies all the remaining available space on the cartridge (up to 16383 blocks).

A non-disk file is created as a type 0 file. In general, a type 0 file can be specified with just the required parameters (file name, LU number, I/O mode). The optional parameters usually follow from this information.

Non-disk (type 0) files are useful because they let you address a device as though it were a file. Programs can address devices using the standard FMP calls.

Use the CR (Create) command, described below, to create a new disk or non-disk (device) file. Note that this command does not store information in the file; it simply creates it.

Purging Files

You can free up disk space by purging files that you no longer need. Once a file is purged, it cannot be accessed, and its name no longer appears on a file directory listing. A new file with the same name can then be created on the same cartridge.

Disk space allocated to purged files is automatically released to the system if the purged files are at the end of the cartridge. To reclaim disk space from other purged files, pack the cartridge with the PK command (described earlier in this appendix).

For example, assume files A, B, and C are the last three files in the directory. They are purged in order, A, B, and C. When C is purged, its disk space is released by the system. The system then checks file B, which is now the last file. Since B was already purged, its disk space is also released. Then file A is checked, and its disk space is released. This process continues until the system finds a file that was not purged.

One or more files can be purged each time you issue the PU command. Wildcard characters in the namr parameter cause all the files matching the namr to be purged from the disk cartridge.

You may purge a scratch file if you append an “A” to the file number. This is because the PU command does not purge a numbered file, but the addition of a character changes the scratch file name from a numeric to an ASCII character string.

Storing Data on a Device or New File

You can use the ST (Store Data on a Device or New File) and DU (Dump Data to a Device or Existing File) commands to transfer data from one disk file or logical unit to another. ST is used when the destination is a new disk file (ST creates the file then stores the data), and DU is used for transfers to an existing file.

CS/80 cartridge tape drives (CTDs) should have files stored to them using either the TF or FC utilities, not the ST or DU command. Refer to the *RTE-A Backup and Disk Formatting Utilities Reference Manual*, part number 92077-90249, for information on the TF and FC utilities.

The ST and DU commands are very similar in the way they operate. Both commands are described in subsequent sections.

Listing the Contents of a File

You may list the contents of a file, file directory information, or data stored on a logical unit to a list device using the LI command. The maximum size of a record being listed is 128 words.

When you use the LI command, the output is directed to the list device. You can change the list device with the LL command.

Renaming Files

At times, you may want to rename an existing disk file. The RN (Rename) command, described below, lets you do this. When you rename a file, the file name is the only characteristic that changes.

A new name must be unique to the cartridge on which the existing file resides.

File Manipulation Commands

All the commands available for creating, purging, and listing FMGR files are discussed in the sections that follow. NO TAG summarizes the commands.

. File Manipulation Commands Summary

Commands		Description
Listing Command		
LI	List Contents of a File	Lists a file, file directory information, or data on an LU to the list device
Configuration Commands		
CR	Create A File	Creates a disk or non-disk (device) file
DU	Dump Data to a Device or Existing File	Transfers data from one disk file to another
PU	Purge a File	Removes selected files and their extents from a cartridge
RN	Rename a File	Changes the name of an existing disk file
ST	Store Data on a Device or New File	Transfers data from one disk file logical unit to another

Create a File (CR)

Purpose: Creates a disk or non-disk (device) file.

Syntax: **For disk files:**

```
CR,fileDes
```

fileDes — The file descriptor parameter, equivalent to:

```
fileName[:security[:cartridge[:fileType[:fileSize
[:recordSize]]]]]
```

This parameter must not be an LU number. Omitted subparameters default to 0. File type must be a positive integer and file size must not be zero. Record size need be specified only for type 2 files. (See NO TAG for a brief description of file types.)

For non-disk files:

```
CR,namr,lu,i/o  mode[,spacing[,fileMark[,dataType]]]
```

namr — The file name and, optionally, security code and cartridge reference; file type defaults to 0 and all remaining subparameters do not apply.

lu — The LU number of the non-disk device; a positive integer.

i/o mode — The required specification of one of the following:

- RE — Read only, accepts input only, forward spacing.
- WR — Write only, accepts output only, no forward spacing.
- BO — Both read and write, accepts input and output, backspacing and forward spacing allowed.

spacing — The type of spacing for the device; if omitted, forward spacing is assumed for read only devices and no spacing is assumed for others. May be specified as:

- BS — Backspacing is supported.
- FS — Forward spacing is supported.
- BO — Both backspacing and forward spacing supported.

fileMark — The type of EOF mark to be written on the device; if omitted, the default depends on the driver type. May be specified as:

- EO — End of file mark for magnetic tape (default if device has driver type greater than 16 octal, cartridge tape or mass storage device).
- PA — Page eject for line printer or two line feeds on teleprinter, device type 00 (default if not a punch or if driver type less than 17 octal).
- cntrl word — Control subfunction (equivalent to function code in FCONT call), supplied if further EOF definition is needed. Specify as octal integer of which only the lowest 5 bits are used.

dataType — The specified type of data on the device; default is ASCII.

- BI — Binary data
- AS — ASCII data
- cntrl word — Subfunction (equivalent to bits 6−10 of IOPTN parameter in OPEN call; supplied if further data definition is needed. Specify as decimal or octal integer of which only the lowest five bits are used.

CR Command Examples

Example 1: Create a type 4 file with a security code of −25 (read and write are restricted to users knowing the code); put the file on cartridge 100, using 10 blocks (20 sectors) of disk space.

```
CR,MYFILE:-25:100:4:10
```

Example 2: Create a type 2 file, using 20 blocks, with each record 72 words.

```
CR,URFILE:::2:20:72
```

Example 3: Create a type 3 file with a security code of EJ (only write restricted), put on cartridge 100; allocate the remaining unused portion of the cartridge, up to 16K blocks, to the file.

```
CR,MYFILE:EJ:100:3:-1
```

Example 4: Create LP as output file on LU 6; defaults are no spacing and ASCII data.

```
CR,LP,6,WR,,PA
```

Example 5: Create MT as input/output file on LU 8, both forward and backspacing supported; security code is 32107.

```
CR,MT:32107,8,BO,BO
```

Example 6: Create a read only cartridge tape file on cartridge 100.

```
CR,MAG:JT:100,8,RE
```

Example 7: Create REALR as input only file on LU 5.

```
CR,REALR,5,RE
```

CR Command Error Handling

FMGR −002 File already exists

A file already exists on the specified cartridge with the requested name. Either purge the file, pick another cartridge, or use a different file name.

Dump Data to a Device or Existing File (DU)

Purpose: Transfers data from one disk file or LU to another.

Syntax: `DU,namr1,namr2[,fmt/cntl[,file#[,#files]]]`

namr1	The source of data. This may be a disk file or an LU number (positive).
namr2	The destination of the data. This may be a disk file or an LU number (positive). If namr2 is a disk file, it must already exist. Handling of the file type and file size parameters is the same as for the ST command; see the ST command discussion for details.
fmt/cntl	This may be a record format parameter, an EOF control parameter, or one of each (separated by a comma). The parameters and their defaults are the same as for the ST command, provided in the previous section.
file#	This parameter is a positive integer that specifies the starting point on namr2. If file# is n, transfer begins at the nth file, (if namr2 is a non-disk device) or subfile (if namr2 is a disk file). If you omit file#, transfer starts at the beginning of namr2. Transfer is always from the beginning of namr1. Note that file# here applies to the destination file, whereas in the ST command this parameter applies to the source file.
#files	This parameter is a positive integer that specifies the number of files (if namr1 is a non-disk device) or subfiles (if namr1 is a disk file) to be transferred from namr1. The default is 1, except that if namr1 is a disk file and you omit file#, the default is 9999.

Description:

Note that files are transferred record by record, and records longer than 128 words are truncated.

Default values and subfile considerations are the same as those for the ST command, provided in the sections above.

DU Command Examples

Example 1: Dump the contents of file SORC3A to a line printer connected as logical unit 6.

```
DU,SORC3A,6
```

Example 2: Dump the contents of binary relocatable file RELBN6 to a magnetic tape mini-cartridge mounted in logical unit 23.

```
DU,RELBN6,23
```

Example 3: Copy a file from cartridge 333 to cartridge FD.

```
DU,FILEX::333,FILEX::FD
```

Example 4: Take the third cartridge tape file on logical unit 8 and the first cartridge tape file on logical unit 23 and concatenate them into one disk file, TAPE12.

```
ST,8,TAPE12,IH,3
DU,23,TAPE12
```

Example 5: Add the contents of file B to file A when both are existing disk files.

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

List Contents of a File (LI)

Purpose: Lists a file, file directory information, or data on an LU to the list device.

Syntax: `LI,namr[,format[,line1[,line2]]]`

namr — The file name or LU number. If the file is protected by a negative security code, you must specify it. If you include the CRN, only that cartridge is searched; otherwise, the first file with the same file name is listed.

format — The list format specification:

- null ASCII source format
- A ASCII source format
- S ASCII source format
- B binary format
- D directory information only

If you omit this parameter, the file type determines the format as follows:

- S if file is type 0, 3, or 4
- B for all other file types

line1 — The first record to list; default is 1.

line2 — The last record to list. If you do not specify line1 or line2, all records of the file are listed. If you specify line1 but not line2, only one record (line1) is listed.

Description:

Binary records longer than 128 words are truncated. Source records are truncated to 72 characters.

On a teleprinter, the list starts in column one; on other list devices two blanks precede each list line. If namr is a file, the listing is headed by:

```
fileName T=fileType IS ON CR cartridge USING size BLKS R=recSize
```

In this case, the lowercase words are replaced by the actual values in the file directory for the file (see the LI Command Examples, below). The record size has a value for type 1 and type 2 files only; all other file types have a record size of 0.

If namr is a logical unit, a brief heading is printed with asterisks replacing the file name. Here nn is the logical unit number.

```
****** T=00000 IS ON LU nn
```

When you specify D, one of the headings shown above is all that is listed. When you specify S, each line number followed by a line of text is printed. When you specify B, the record number is printed, followed by each word of the record. Words are printed in octal, followed by an ASCII equivalent if a legal ASCII character corresponds to the octal. The ASCII is separated from the octal by an asterisk. Lines are truncated after the last non-blank character (asterisks are treated as blanks in this case). Binary format prints eight words per line, using as many lines as are needed to print the record up to the maximum of 128 words.

For zero-length records, only the record number is printed.

LI Command Example

```
FMGR : LI,&STH3
&STH3  T=00003 IS ON CR21587 USING 00002 BLKS R=0000

0001        PROGRAM TEST3
0002        INTEGER P(5)
0003        CALL RMPAR (P)
0004        LU = P(1)
0005        DO 120 J = 401,600
0006        WRITE(LU,116) J
0007    116 FORMAT(1X,"J = ",I3)
0008        IF(IFBRK(IDMY))200,120
0009    120 CONTINUE
0010        WRITE(LU,136)
0011    136 FORMAT(1X,"END STOP 3")
0012        GO TO 300
0013    200 WRITE(LU,216)
0014    216 FORMAT(1X,"BREAK STOP 3")
0015    300 END

FMGR : LI,&STH3,D
&STH3  T=00003 IS ON CR21587 USING 00002 BLKS R=0000

FMGR : LI,&STH3
%STH3  T=00005 IS ON CR21587 USING 00002 BLKS R=0000

REC# 00001

010400 020000 175566 052105 051524 031440 000127 000000*      TEST3  W
000000 000003 000143 000000 000000 000000 000000 000000*
000000

REC# 00002

003400 040001 015277 052105 051524 031440 000005       *  @  ?TEST3

REC# 00003

014000 100007 153051 027104 044517 027003 027111 044517*    V).DIO. .IIO
027004 027104 052101 027005 042530 042503 020007 041514*. .DTA. EXEC  CL
051111 047401 051115 050101 051002 044506 041122 045406*RIO RMPAR IFBRK
.
.
.
...and so on.
```

Purge a File (PU)

Purpose: Removes selected files and their extents from a cartridge.

Syntax: `PU,fileDescriptor[,msc]`

fileDescriptor	The file name (see comments) and, optionally, security code, cartridge identifier (negative LU number or positive CRN, or 2 ASCII characters), file type, and file size.
msc	The master security code; used to override file security code on wildcard purges.

Description:

If a file is protected by a security code, you cannot purge it unless you enter the correct code or the master security code is supplied.

When you specify a cartridge identifier, only that cartridge is searched for the file to be purged; otherwise, all mounted cartridges are searched and the first file found with the specified namr is purged. The search starts with the first cartridge in the cartridge directory. The CL command shows you the order in which cartridges are searched when a cartridge identifier is not specified.

PU Command Examples

Example 1: Purge the first file found with a name of DATA7.

```
FMGR: PU,DATA7    (Namr1 is DATA7 with no security code and no disk id supplied.)
FMGR:             (Ready for next command.)
```

Example 2: Purge a file named A4 with a security code of XX on cartridge number 43.

```
FMGR: PU,A4:XX:43        (Namr1 is A4 with security code XX on cartridge 43.)
FMGR:                    (Ready for next command.)
```

Example 3: Purge all files on cartridge 30 starting with FILE. You must enter the master security code in order to ensure that all files regardless of security code are purged.

```
FMGR: PU,FILE--::30,YY   (Namr is FILE-- on cartridge 30.  The msc is YY.)
FILEA
FILEB
FMGR:                    (Ready for next command.)
```

Rename a File (RN)

Purpose: Changes the name of an existing disk file.

Syntax: `RN,fDesc,newName`

fDesc	The file name and, optionally, security code and cartridge identifier (negative LU number or positive CRN).
newName	The new file name to replace existing file name.

Description:

When you include a cartridge identifier in namr, only that cartridge is searched. If you omit the cartridge identifier, the program searches through all the mounted cartridges and renames the first file it finds. The search starts with the first cartridge in the cartridge directory. You can use the CL command to see the order in which the cartridges are searched.

Note that if the file you are renaming was created with a non-zero security code, you must include the same security code in the namr.

RN Command Examples

Example 1: Search the cartridge for the first occurrence of file JD17 and change its name to DATA1. The file is not protected by a security code.

```
RN,JD17,DATA1
```

Example 2: Rename file LOG3 on cartridge 100 to Table 2. The file has a security code of XX.

```
RN,LOG3:XX:100,TABLE2
```

Store Data on a Device or New File (ST)

Purpose: Transfers data from one disk file or logical unit to another.

Syntax: `ST,namr1,namr2[,fmt/cntl[,file#[,#files]]]`

namr1	The source of data. This may be a disk file or a positive LU number.
namr2	The destination of data. This may be a disk file or a positive LU number. If namr2 is a disk file, it must not already exist.
fmt/cntl	This may be a record format parameter, an EOF control parameter, or one of each (separated by a comma).
	The record format (fmt) parameter specifies the format of data in namr1, and may be:

AS ASCII records are transferred.

BR Binary relocatable records are transferred; checksum is performed.

BN Binary relocatable records are transferred; no checksum is performed.

BA Binary absolute records are transferred; checksum is performed.

The EOF control (cntl) parameter specifies the treatment of file marks. The term "file mark" refers to a subfile mark on a disk file or an EOF mark on a non-disk file. As embedded file marks are transferred, they are changed to the form that is appropriate for namr2: EOF marks for non-disk file, subfile marks for disk files. (See the explanation of subfiles, below.)

The EOF control parameter may be:

IH No file marks, embedded or terminating, are passed to namr2.

SA All file marks, embedded and terminating, are passed to namr2.

If the EOF control parameter is omitted, embedded file marks are not passed to namr2, but the terminating file mark is passed.

file# This parameter is a positive integer that specifies the starting point on namr1. If file# is n, transfer begins with the nth file. If you omit file#, transfer starts at the beginning of namr2.

#files This parameter is a positive integer that specifies the number of files (if namr1 is a non-disk device) or subfiles (if namr1 is a disk) to be transferred from namr1. Default is 1, except that if namr1 is a disk file and file# is omitted, the default is 9999.

Description:

Note that files are transferred record by record; records longer than 128 words are truncated.

Default Values

If namr2 is a disk file and the file type subparameter is not specified, then:

If namr1 is a disk file, the file type of namr1 is assigned to namr2.

If namr1 is a device, the file type of namr2 is:

type 3 if the record format of namr1 is omitted
type 3 if the record format of namr1 is AS
type 5 if the record format of namr1 is BR
type 3 if the record format of namr1 is BN
type 7 if the record format of namr1 is BA

If namr2 is a disk file and the file size subparameter is not specified, then:

If namr1 is a disk file, the file size of namr1, without extents, is assigned to namr2.

If namr1 is a device, the file size of namr2 is set to 24 blocks.

If the record format is not specified and namr1 is a disk file, the record format defaults to:

AS if namr1 is a type 3 or type 4 file
BR Binary relocatable records are transferred; checksum is performed
BN Binary relocatable records are transferred; no checksum is performed
BA Binary absolute records are transferred; checksum is performed

Subfiles

Subfile marks are zero-length records that are used to subdivide a disk file. You can use subfile marks to store several non-disk files (say, cartridge tape files) on one disk file, but still maintain them as separate entities. The EOF mark used by a non-disk device depends on the type of device:

- A 264x cartridge tape unit uses a special magnetic tape EOF mark.
- A line printer uses a top-of-form (page feed) control sequence.
- A terminal uses a control D on input and a zero-length record (two consecutive carriage returns) on output.

When files containing these marks are transferred to a disk file, the file marks may be written as EOF or subfile marks, or may be deleted, according to the EOF control parameter. When a disk file is transferred, its subfile marks (if any) may be transferred as subfile marks (to another disk file), as EOF marks, or may be deleted.

ST Command Examples

Although the ST command can become rather complex in some situations, most of the time it is quite simple to use. The following examples show some typical uses of the ST command.

Example 1: Transfer a disk file of FORTRAN source code, &FORT2, to a magnetic tape on logical unit 8.

```
ST,&FORT2,8
```

Example 2: Transport the source file, &FORT4, of a program stored on a model 7908 hard disk (for example, LU 20), to another system on a flexible disk that has a CRN equivalent to FD.

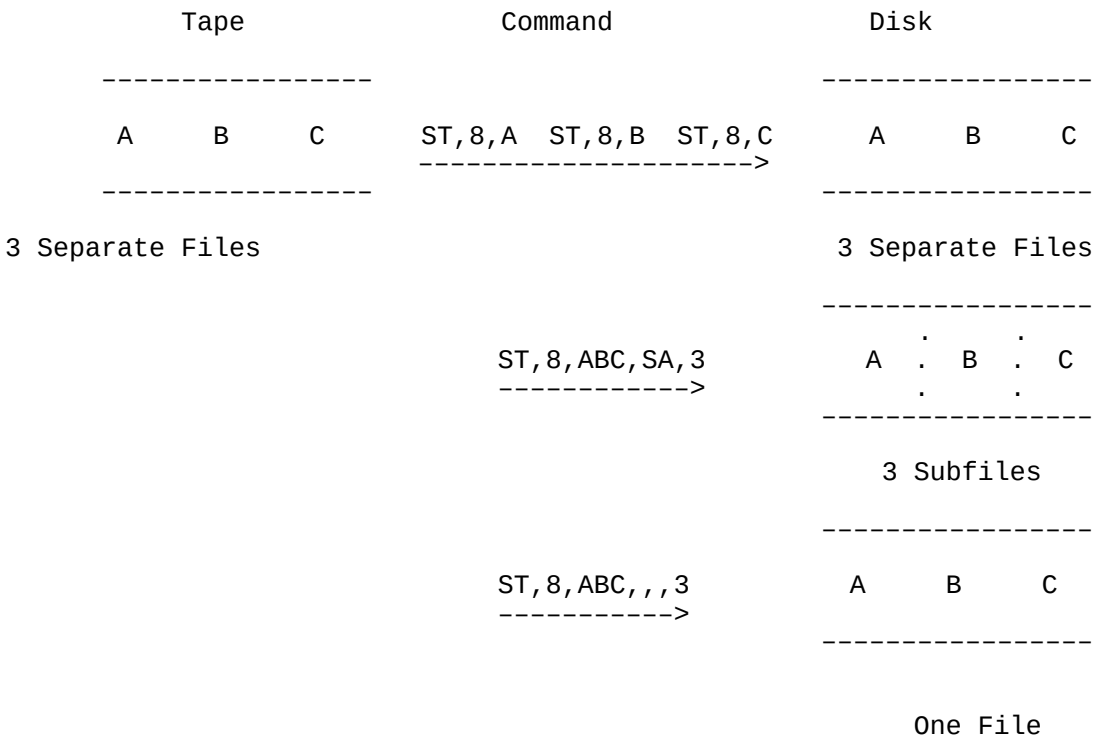
```
ST,&FORT4::-20,&FORT4::FD
```

When you do this, you can then carry the flexible disk to the other system and operate from the flexible disk or transfer the program to a different disk once again with the ST command.

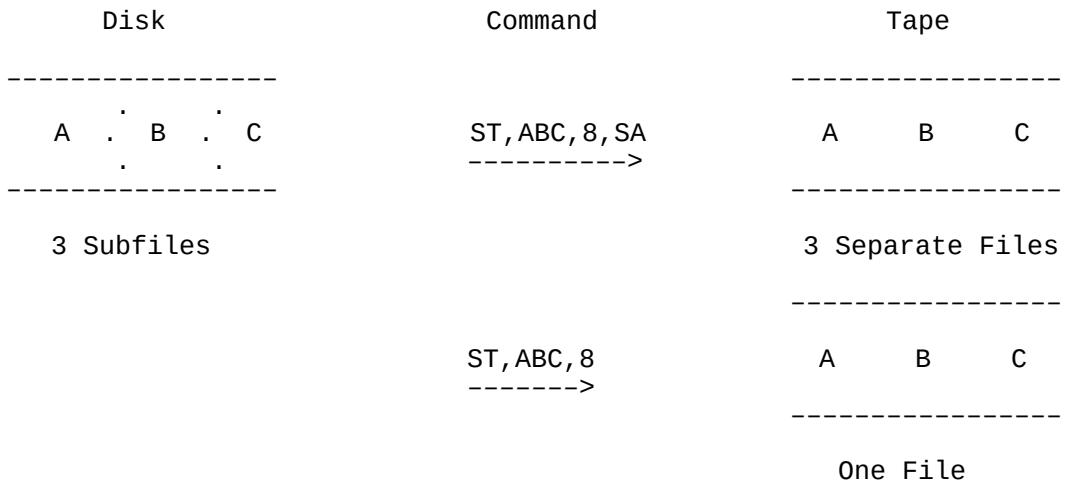
Example 3: Transport a file of binary relocatable code, %RELO5, from LU 20 to CRN FD.

```
ST,%RELO5::-20,%RELO5::FD
```

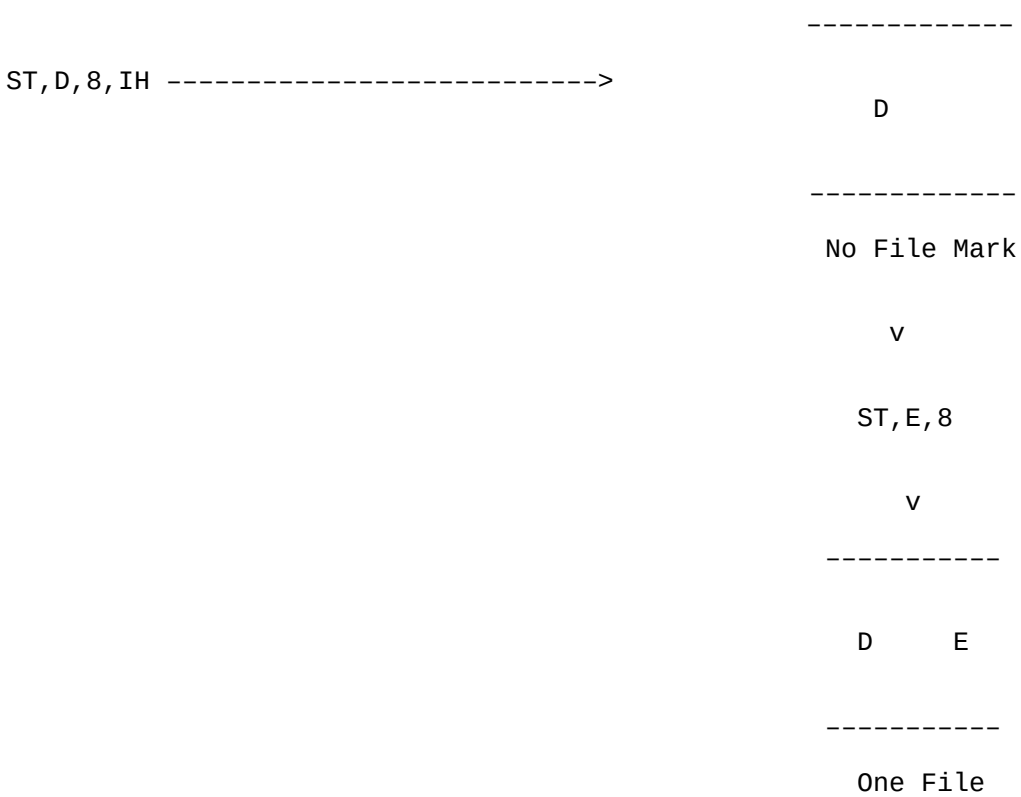
Example 4: Transfer the first three files on a cartridge tape (A, B, and C), which contain program source code (ASCII), to disk as separate files, subfiles, or one file.



Example 5: Transfer a disk file composed of three ASCII subfiles (A, B, and C) to cartridge tape as separate files or as one file.



Example 6: Concatenate two disk files, D and E, on one magnetic tape file.



Example 7: Take the fourth, fifth and sixth files of binary relocatable code on a magnetic tape mounted on LU 8 and copy them to three subfiles on disk file !BINR3 on cartridge CC.

```
ST,8,!BINR3::CC,BR,SA,4,3
```

Note that in this case, you must specify the format parameter (BR). In Example 3 above, the format parameter was not necessary because it could be derived from the file type. In the case of a cartridge tape file, however, the system has no way of knowing the file type or the format type, so you must specify the format type.

Transfer Files

A transfer file (sometimes called a procedure file) is, in effect, a program written in the language of the operator commands. It lets you execute a series of commands by transferring control to a file that contains those commands. This is very useful when you need to execute a particular sequence of commands repeatedly.

Creating a Transfer File

Suppose you are using a terminal assigned to LU 13 and are debugging a particularly troublesome FORTRAN program. You make repeated runs of the compiler, the loader, and the loaded program by typing in the following command sequence:

```
FMGR: PU,TEST4                          (Purge the old type 6 file)
FMGR: RU,FTN7X,&SORCE,13,%SORCE         (Compile the source code)
FMGR: RU,LINK,,%SORCE                   (Load the relocatable code)
FMGR: RU,TEST4                          (Run the new program)
```

You can make your job easier by creating a transfer file that looks like this:

```
:PU,TEST4
:RU,FTN7X,&SORCE,13,%SORCE
:RU,LINK,,%SORCE
:RU,TEST4
:TR
```

You can create this file with the text editor program. The only difference between this file and the original sequence of commands is that a colon (:) precedes each command, and there is a TR statement at the end of the file. If you name this file TFILE4, you can compile, load and execute your program by typing the following command in response to an FMGR prompt:

```
TR.TFILE4
```

This causes FMGR to transfer control to file TFILE4 and execute the commands contained in the file. FMGR terminates the old program and releases its ID segment, compiles the new program, loads it, executes it, and returns control to your terminal.

This is much easier than retyping the commands each time, and it lets you spend your time debugging the program rather than typing at your terminal.

G-Type Global Parameters

G-type global parameters are variables that increase the generality of a transfer file. Commonly called G globals, they hold an integer value or an ASCII string up to six characters long. There are nine general and two specialized G globals.

You can directly assign values to globals 1G through 9G. Global 0G is set to the namr value of the input device when FMGR was scheduled; thus, 0G usually contains the LU number of the terminal you are using.

Global 10G is set by a call to library routine PRTN (refer to the *RTE-A Programmer's Reference Manual*, and see also the explanation of P-type globals, below). The most visible use of global 10G is that the on-line loader sets 10G equal to the name of the most recently loaded program or to a loader error code if the load failed.

You can pass values to globals 1G through 9G as part of the TR command or you can set them with the SE (Set) or CA (Calculate) commands in a transfer file. You can also make conditional jumps with the IF (Conditional Skip) command. (The TR, SE, CA, and IF commands are discussed in the following sections.)

The following example shows how you can set up a more general transfer file to compile, load, and execute a FORTRAN program.

```
:PU,3G
:RU,FTN7X,1G,0G,2G
:RU,LINK,,2G
:IF,10G,EQ,3G,2
:DP,LOAD FAILED,RETURNING CONTROL TO TERMINAL
:TR
:RU,10G
:TR
```

You can name this file TESTG and execute it with the following command:

```
FMGR : TR,TESTG,&Sorce,%SORCE,TEST4
```

When values are substituted for globals 0G, 1G, 2G, 3G and 10G, the file becomes:

```
:PU,TEST4
:RU,FTN7X,&SORCE,13,%SORCE,TEST4
:RU,LINK,,%SORCE
:IF,TEST4,EQ,TEST4,2
:DP,LOAD FAILED,RETURNING CONTROL TO TERMINAL
:TR
:RU,TEST4
:TR
```

The IF statement tests whether the load is successful. When it is successful, control skips two statements and schedules the program that was just loaded. If the load is unsuccessful, the value of 10G is set to a loader error code and the transfer file prints the message (DP statement) and terminates.

Each G global is composed of four words. The first word, word 0, defines the type of value held by the global; the remaining words, words 1 through 3, hold the actual value. The arrangement is shown below:

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    word 0 (type)       0 (null)     1 (numeric)     3 (ASCII)
-------------------------------------------------------------------------
    word 1 (value)         0           integer       characters 1 & 2
-------------------------------------------------------------------------
    word 2 (value)         0              0          characters 3 & 4
-------------------------------------------------------------------------
    word 3 (value)         0              0          characters 5 & 6
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

P-Type Global Parameters

P-type global parameters, commonly known as P globals, are one-word global parameters numbered from −36P to 7P. They are related to G globals as shown in NO TAG.

0G	word 0 (type)	
	word 1	
	word 2	
	word 3	
1G	word 0 (type)	−36P
	word 1	−35P
	word 2	−34P
	word 3	−33P
2G	word 0 (type)	−32P
	word 1	−31P
	word 2	−30P
	word 3	−29P
3G	word 0 (type)	−28P
	word 1	−27P
	word 2	−26P
	word 3	−25P
4G	word 0 (type)	−24P
	word 1	−23P
	word 2	−22P
	word 3	−21P
5G	word 0 (type)	−20P
	word 1	−19P
	word 2	−18P
	word 3	−17P
6G	word 0 (type)	−16P
	word 1	−15P
	word 2	−14P
	word 3	−13P
7G	word 0 (type)	−12P
	word 1	−11P
	word 2	−10P
	word 3	−9P
8G	word 0 (type)	−8P
	word 1	−7P
	word 2	−6P
	word 3	−5P
9G	word 0 (type)	−4P
	word 1	−3P
	word 2	−2P
	word 3	−1P
10G	word 0 (type)	0P
	word 1	1P
	word 2	2P
	word 3	3P
		4P
		5P
		6P
		7P

. Global Parameters

You can use P globals to examine any of the words in G globals 1G through 10G. In addition, globals 1P through 5P contain the values passed to and from programs using the RMPAR and PRTN subroutines. (RMPAR and PRTN are discussed in the *RTE-A Programmer's Reference Manual*.)

For example, after executing a program, the value returned in global 10G (actually 1P to 3P) is either zero or the return parameters of the program. Global 6P contains the latest error code, and global 7P contains a copy of the current severity code.

If you are using the contents of global 6P to check for errors, note that the program clears the error code (6P is reset to 0) after the error message is expanded with the ?? command. The error code is also cleared after automatic error expansion (severity code set at 1000 to 1004).

Thus, a check of global 6P always returns a value of zero if automatic error expansion is in effect (because the error code is cleared immediately after it is reported and expanded). Therefore, if your transfer file uses the value in 6P, you should make sure that the severity code is not set for automatic error expansion.

The FORTRAN compiler passes error information via a PRTN call, and the information shows up in globals 1P through 5P. Of particular interest is the value in 1P, which shows the total number of errors, warnings, and disasters encountered during compilation.

The following transfer file expands on the one in the previous example by setting 1P to zero (CA statement) before compilation, then checking to make sure that there were no errors during compilation. The value of the current severity code is saved, and the severity code is reset to 1 (no command echo) while the transfer file executes, and then it is reset to the original value at the end of the file. Two IF statements are used for unconditional skips and the :* statements are comment lines.

```
:*  TRANSFER FILE TESTG        (Comment, not executed)
:*
:CA,4G,7P                      (Save severity code in 4G)
:SV,1                          (Set severity code to 1)
:CA,1:P,0                      (# set 1P to 0)
:PU,3G                         (# purge the old type 6 file)
:RU,FTN7X,1G,0G,2G             (Compile source code)
:IF,1P,EQ,0,2                  (Test for compilation errors)
:DP,** COMPILATION ERROR **
:IF,1,EQ,1,5                   (Unconditional skip)
:RU,LINK,,2G                   (Load the relocatable code)
:IF,10G,EQ,3G,2                (Test for load errors)
:DP,** LOAD ERROR **
:IF,1,EQ,1,1                   (Unconditional skip)
:RU,10G                        (Run the program)
:SV,4G                         (Reset severity code)
:DP,** TESTG TERMINATING **
:TR                            (Return)
```

You can execute this transfer file with a command such as:

```
Loops
```

You can also perform loops in a transfer file using the CA command and the IF test with a negative skip count, as follows:

```
.
.
.
FMGR : CA,7G,0              (Initialize a counter)
.
.
a
few                         (These commands are executed 10 times)
other
commands
.
.
FMGR : CA,7,7G,+,1          (Increment the counter test; go to top of loop)
FMGR : IF,7G,LT,10,-6

.
.
.
```

TR and Related Commands

NO TAG summarizes the commands for creating and using transfer files.

. Transfer Commands Summary

Commands		Description
TR	Transfer Control to a File or Device	Transfers control to a file or a logical unit
SE	Set Global Parameter	Assigns values to the global parameters 1G through 9G
CA	Calculate Global Parameter	Assigns individual G-type and P-type global parameter values or nulls them
IF	Conditional Skip	Compares two values (usually global parameters) number of commands, depending upon the result of the comparison
PA	Pause and Send Message	Suspends execution of a transfer file and transfers control to the log device or other specified device
DP	Display Parameters	Displays the current values of up to 14 parameters on the log device, which must be an interactive terminal

Calculate Global Parameter (CA)

Purpose: Assigns individual G-type and P-type global parameters values or nulls them.

Syntax: `CA,global#[,p1[,op1,p2[op2 ... ,opn,pn+1]]]`

global# — The integer (1−9) that identifies the global 1G through 9G to be set to the result of the calculation.

Globals 36P through 1P and 1P through 6P may also be set to the result of the calculation using the following entry form for global#:

`n:P`

n is the P type global to be set in the range −36 through +6 (excluding 0). For example, to set global 6P to zero, enter:

`:CA,6:P,0`

or

`:CA,6:P` (if value or operation is omitted, zero is assumed)

You can also specify the global type in the first subparameter. "P" identifies P-globals, and "G" identifies G-globals. If you omit the subparameter, G-globals are assumed.

p1−pn — Values used in calculations. If you omit entering them, the global is nulled.

op1− opn — Operations performed on operands. These may be:

+	add two operands
−	subtract one operand from another
/	divide one operand by another
*	multiply one operand by another
O[R]	inclusive OR two operands
X[OR]	exclusive OR two operands
A[ND]	AND two operands

Description:

In its simplest form, CA is used to null an individual global parameter or to set an individual global to the value of p1. The values assigned can be the result of arithmetic or logical calculations.

The type of result depends upon the type of operands you enter. If operand types differ in any one CA statement, the highest type value is used (type 0 = null, type 1 = numeric, and type 3 = ASCII). For example, if one operand is numeric and the other is ASCII, the result is ASCII.

Calculations are performed separately on each word of three-word ASCII globals, except for division and multiplication, in which all three words of the first operand are divided or multiplied by word 1 of the second operand.

When you invoke the command, evaluation proceeds from left to right until a null operation code is detected. Multiple CA statements may change the precedence.

You cannot set all of the globals; some are used to provide system information. The G-globals that cannot be set are 0G and 10G, and the P-globals are 0P and 7P.

CA Command Examples

```
FMGR : CA,6,FTN7X          (Set global 6G to ASCII value "FTN7X")
FMGR : DP,-16P             (Display type of 6G is ASCII (3))
       3
FMGR : CA,6                (Clear global 6G to null value)
FMGR : DP,-16P             (Display type of 6G null (0))
       0

FMGR : CA,2,15             (Set global 2G to integer value 15)
FMGR : DP,-32P             (Display type of 2G numeric (1))
       1
FMGR : CA,7,2G             (Set global 7G to current value of 2G)
FMGR : DP,-12P             (Display type of 7G assumes type of 2G)
       1
FMGR : DP,2G,7G
       15,15               (Display values of 2G, 7G)
FMGR : CA,1,2G,*,14,+,1    (Set 1G to product of 2G and 14 plus 1)
FMGR : DP,1G
       211                 (Display 1G)
FMGR : CA,7,7G,-,1         (Decrement 7G by 1)
FMGR : DP,7G
       14                  (Display 7G)

FMGR : CA,1,7,OR,15        (Inclusive OR 7 and 15 (octal 17),
                            assign to 1G)
FMGR : CA,2,7,XOR,15       (Exclusive OR same values, assign to 2G)
FMGR : CA,3,7,AND,15       (AND these values, assign to 3G)
FMGR : DP,1G,2G,3G
       15,8,7

FMGR : DP,-23P             (Manipulation of P globals)
       0
FMGR : CA,-23:P,-23P,+,1
FMGR : DP,-23P
       1
```

Display Parameters (DP)

Purpose: Displays the current values of up to 14 parameters on the log device, which must be an interactive terminal.

Syntax: `DP[,p1[,p2...[,p14]...]]`

p1 to p14	The parameter values or names of global parameters to be displayed. These may also be a string of text. If you do not enter any values, nothing is displayed.

Description:

Although you may use the DP command to display any parameter value, it is typically used to display the values of global parameters (G-globals or P-globals). Null global parameters are displayed as a pair of commas (,,). See also the discussion of global parameters in previous sections in this appendix.

You can use DP to send messages to the log device without causing a transfer of control, as shown in Example 1 below.

Note that the DP command displays parameter values even if the severity code is greater than zero.

DP Command Examples

The following excerpts from a transfer file illustrate some of the uses of the DP command:

```
:
:
:
:DP,First section (initialization) completed.
:
:
:
:DP,Output file is,1G
:DP,Values of -36P through -33P are, -36P, -35P, -34P, -33P
:
:
:
:DP,Job done!
:
:
:
```

When the transfer file is executed, the log device displays:

```
First section (initialization) completed.

Output file is,FILE23
Values of -36P through -33P are,3,FI,LE,23

Job done!
```

Conditional Skip (IF)

Purpose: Compares two values (usually global parameters) and skips a specified number of commands, depending upon the result of the comparison.

Syntax: `IF,p1,operator,p2[,skip]`

p1,p2 — The values to be compared; one or both may be global parameters.

operator — The relative operator used to compare values of p1 and p2, entered as one of the following two-character keywords:

Keyword	Operation
EQ	p1 = p2
NE	p1 /= p2
LT	p1 < p2
GT	p1 > p2
GE	p1 @>@ p2
LE	p1 @<@ p2

skip — The skip count, a positive or negative integer that specifies the number of commands to skip when the relation between p1 and p2 is true; forward skip if positive, backward if negative. If you omit this parameter, the program skips one command. If the relation is not true, the next sequential command is executed.

Description:

Note that you cannot execute IF from an interactive device, only from within a transfer file.

The specified relation between p1 and p2 is examined and if it is true, commands are skipped. If you do not specify a skip count, one command is skipped; otherwise, as many commands as the specified skip number are skipped. A skip of −1 causes the IF command to repeat. To skip back to the preceding command, specify −2.

IF does not skip past the beginning or the end of the transfer file; an attempt to do so causes a skip to the beginning or EOF mark, but does not cause an error.

To use a negative skip, the file must be on a device that recognizes a backspace.

The following relations hold for mixed types:

null < numeric < ASCII

This corresponds to the type codes: null=0, numeric=1, ASCII=3.

If p1 and p2 are both ASCII, the comparison is based on the ASCII collating sequence.

You may also use the IF command to do an unconditional skip by making a comparison that is always true (see the examples below).

IF Command Examples

```
:*  PERFORM TWO SECTIONS OF COMMANDS
:*  DEPENDING ON THE STATE OF 1G
:*
:IF,1G,EQ,1,4
:IF,1G,EQ,2,9
:DP,VALUE OF 1G (,1G,) IS OUT OF RANGE
:SE
::
:*  EXECUTE THIS CODE IF 1G = 1
:   first command
:   second command
:   third command
:*  NOW UNCONDITIONALLY SKIP THE '1G = 2' CODE
:IF,1,EQ,1,4
:*  EXECUTE THIS CODE IF 1G = 2
:   first command
:   second command
:   third command
:*  EXECUTION CONTINUES HERE
:
```

To do an unconditional skip, just make sure that the comparison is always true:

```
:** IF 7G IS NEGATIVE, ZERO OUT 4G
:** ELSE (POSITIVE OR ZERO) PUT 7G INTO 4G
:**
:IF,7G,GE,0,2
:CA,4,0
:IF,0,EQ,0,1
:CA,4,7G
:CA,5,4G,*,6G,+,1
:
:
:
```

Pause and Send Message (PA)

Purpose: Suspends execution of a transfer file and transfers control to the log device or other specified device.

Syntax: `PA,lu`

lu	The LU number of the device to which control is transferred and on which messages are displayed. If you do not specify an LU, the log device is assumed.

PA Command Example

You can use PA in a transfer file to request and wait for operator action/response. For example, assume a transfer file contains the following command:

```
:DP,,Insert cartridge tape on LU 8, then type TR.
:PA
```

When the command executes, the log device displays:

```
:Insert cartridge tape on LU 8, then type TR.
```

The PA command gives you a chance to respond. After you mount the tape and type TR on the log device, control is transferred back to the transfer file and execution continues with the next command after PA.

Set Global Parameter (SE)

Purpose: Assigns values to the global parameters 1G through 9G.

Syntax: `SE[,p1[,p2[...[,p9]]]]`

p1 through p9	Values assigned to global parameters 1G through 9G. If you omit all parameters, are nulled. If you omit any one parameter, the corresponding global parameter is unchanged.

Description:

The value in p1 is assigned to 1G, that in p2 to 2G, and so on. You can use SE alone (no parameters) to have all the global parameters 1G through 9G cleared (nulled). Individual global parameters can be nulled or set with the CA command.

You may assign any integer value between −32768 and 32767 or an ASCII value up to six characters to p1 through p9. If, however, you enter a global name (0G − 10G, 1P − 5P) as a parameter, the value of the specified global is assigned to the global parameter that corresponds to the parameter position.

SE Command Examples

Example 1: Assign global 5G. Assign values to globals 1G through 4G. Keep 5G through 9G unchanged. Display the globals (DP).

```
SE,,,,,FIVE
SE,256,NEWFIL,AA,0
DP,1G,2G,3G,4G,5G
256,NEWFIL,AA,0,FIVE
```

Example 2: Set global 1G to the value of 4G, in this case, 0.

```
SE,4G
DP,1G
          0
```

Transfer Control to a File or Device (TR)

Purpose: Transfers control to a file or a logical unit.

Syntax: `TR[,,parameters]`

or

`TR,namr[,parameters]`

or

`TR,-integer[,parameters]`

namr	The namr identifies a file or logical unit to which control is transferred. Pointers to the namrs are maintained in a transfer stack and may be nested up to 8 levels deep.
−integer	A negative integer. Control is transferred back the specified number of levels in the transfer stack. If the integer is greater than the current level, the stack is reset to the top.
parameters	The values to be set for the global parameters 1G through 9G. Position determines the global parameter to which the value is passed; omitted global parameters are unchanged.

Note that a colon (:) or comma (,) may replace TR as the command code.

Description:

The form TR,namr is like a subroutine call. It transfers control to the specified namr (either a file or a device) and starts executing whatever commands it finds. When all the commands are executed, TR causes transfer of control back to the calling namr (like the RETURN statement of a subroutine), and execution continues with the first statement after the calling TR statement.

When you issue a PK command from within a transfer file, it is important to ensure that the transfer file does not cross a sector boundary, else, commands that follow PK may be read from the wrong area of the disk. It is best to make a file that contains the PK command only and invoke this transfer file from within other transfer files when necessary.

A TR command (with no parameters) is the same as a TR,−1 command. Both commands transfer control back one level in the transfer stack. A TR,−3 transfers control back three levels in the transfer stack, and so on.

When you transfer back to a file on disk, it is possible to backspace and re-execute one or more commands within that file. To do this, specify a negative integer as the security code of a null namr. For example, the command TR,:−4 transfers back to the previous namr, backspaces four commands, and then begins to execute. (There is no way to skip levels and backspace at the same time.)

If an error that requires operator intervention occurs, the system transfers control to the log device. You can use the TR command (either TR or a colon) to transfer control back to the file or device where the error occurred.

TR Command Examples

Example 1: Transfer control to file TFILE1.

```
TR,TFILE1
```

Example 2: Transfer control to file TFILE2 and set the value of global parameter 1G to FILEA.

```
TR,TFILE2,FILEA
```

Example 3: Transfer control to file TFILE3 and set the value of global parameter 4G to 4. Values of all other global parameters are unchanged.

```
TR,TFILE3,,,,4
```

Device Manipulation

Device manipulation includes taking down a device and displaying and modifying buffer limits. The following sections discuss how to call FMGR and COMND, and describe the DN (Make Device Unavailable) and the BL (Display/Modify Buffer Limits) commands. (See end of appendix for more information on the COMND program.)

Calling FMGR and COMND

To call FMGR and COMND, enter the commands in the runstring as shown below, from a device or from a file.

```
FMGR: RU,FMGR[,input[,log[,list[,severity]]]]  (Commands entered from device)

FMGR: RU,FMGR,fi,le,nm[,severity[,list]]       (Commands entered from file)

FMGR: RU,COMND[,INPUT[,LOG]]
```

input	The LU number of the input device for FMGR or COMND commands. If you do not enter an LU number, the console log device is assumed.
log	The LU number of the log device used to log and to correct any diagnosed errors. It must be an interactive device.
list	The LU number of the device used to list results of FMGR commands. If you do not enter another, LU 6 is assumed.
fi,le,nm	The name of the file that contains command input to FMGR commands. Specify 3 words, each consisting of 2 ASCII characters.
severity	The severity code that defines the action in case of error messages.

- 0 Display error codes and echo commands on log device (the default).
- 1 Display error codes on log device, inhibit command echo.
- 2 Error code displays only if error requires transfer of control to log device for correction; in this case, the active job terminates. No command echo.
- 3 Same as 2, except that the active job does not terminate when an error causes transfer to the log device.
- 4 If an FMGR command error is encountered, the job continues automatically. There is no command echo, no transfer to log device, and no job abort occurs.

Once FMGR is running, you may change the list device using the LL command, and the severity code using the SV command.

Alternatively, you can call FMGR from a program, as follows:

```
CALL  EXEC(ICODE,FMGR,INP,LOG,LIST,ISV,IDUM,IBUFR,IBUFL)
                                                    (Entered from device)

CALL  EXEC(ICODE,FMGR,2HFI,2HLE,2HNM,ISV,LIST,IBUFR,IBUFL)
                                                    (Entered from file)
```

ICODE — This is 23 to schedule FMGR with wait, or 24 to schedule FMGR without wait.

FMGR — A 3-word array that contains ASCII file name FMGR.

INP,LOG,LIST,ISV
FI,LE,NM,ISV,LIST

These correspond to the parameters in the RU,FMGR command.

IDUM — The placeholder for parameter 5 in a command entered from a device.

IBUFR,IBUFL

The buffer address and length containing the command string to be passed to FMGR.

You must begin the command string that you want to pass to FMGR via IBUFR with a colon, or the program ignores the commands. The buffer length that you specify in IBUFL is a positive value for words, or a negative value for characters.

FMGR assumes that the string you enter is a command and executes it before the first command on the specified input device is executed.

Device Manipulation Commands

The following sections describe the BL and DN commands.

Display/Modify Buffer Limits (BL)

Purpose: Lets you examine and reset the buffer limits for an I/O device.

Syntax: `BL,lu[,buf[,low[,high]]]`

lu	The logical unit number.
buf	BU (buffered) or UN (unbuffered).
low	The low buffer limit, > 0.
high	The high buffer limit, low < high < available SAM.

Description:

This form of the command always displays the buffer limits for the specified LU:

```
FMGR : BL,lu
```

This form of the command always modifies the buffer limits:

```
FMGR : BL,lu,buf,low,high
```

The BL command always provides one of the following responses:

```
LU#lu  buf  BL=low,high  AC=accum
```

```
LU # lu UNASSIGNED.
```

lu	The logical unit number.
buf	BU or UN.
low,high	The actual limits applied.
accum	The current buffer accumulation.

If buf=BU, all output to the named LU is directed to an I/O buffer in System Available Memory (SAM). The LU (peripheral device) retrieves the output from SAM and completes the output processing at its own rate. This allows a program to continue without waiting for the output processing to complete.

To prevent having all of SAM monopolized by slow devices, the system limits the accumulation (AC) of buffer requests. When the accumulation for a buffered device exceeds the high limit, any program that attempts I/O to the device is suspended, and not resumed until the accumulation drops below the low limit.

Low and high buffer limits are truncated to multiples of 16 words. The low limit may never exceed the high limit, and the high limit may never exceed 6112 words. If you specify a high limit that is larger than the maximum, the system reduces the specification to the maximum allowed. The low limit is increased, if necessary, so that it is within 2032 words of the high limit. If you specify low=high=0, buffer limit constraints are not applied to the named LU.

The buffer limits that you set for a device depend upon several factors:

- The output speed of the device.
- The rate of output of your programs.
- The amount of memory available for buffering.
- The extent to which you can afford to have your programs go into I/O suspension.

Whenever you reboot, the buffer limits revert to those set at system generation.

BL Command Examples

Example 1: Display buffer limit

```
BL,6

LU# 6 UN BL=768, 1056 AC= 0
```

Example 2: Modify buffer limits.

```
BL,6,BU,200,800

LU# 6 BU BL=192, 800 AC= 0
```
(Limits are truncated to multiples of 16)

Make Device Unavailable (DN)

Purpose: Declares an LU or device down and unavailable for use by the system.

Syntax: `DN,lu`

lu The device LU number.

Description:

When you down an I/O device (LU), only that specified device becomes unavailable. Others that use the device's I/O channel are unaffected. The I/O device remains unavailable until you bring it back up with the UP command. Equipment problems or normal maintenance (for example, changing paper or ribbon on a printer) may necessitate taking down a device.

DN Command Example

For example, to take down LU 8, simply enter the command as follows:

```
FMGR : DN,8
```

Display Device Status

The DS command can be used to display device status.

DS Command

Purpose: Displays device status.

Syntax: `DS,lu`

lu The logical unit number of the device.

Device status is indicated by the following codes:

			AV	
	UP		ND	
Response:	DN	DS(*xxx*)	BZ[*proga*]	LK[*progb*]
			AB[*proga*]	
			SY	
			CL	

or: `UNASSIGNED`

where the codes listed above have the following meanings:

`UP` Device considered operable (up) by the system.

`DN` Device has been set down as a result of an error reported by the driver or by the DN command.

`DS` Eight bits of status posted by the driver at the completion of the last I/O operation. (DVT word 6.) Status displayed in octal format.

`AV` Available; no requests pending on the device.

`ND` Some other device on the I/O node is busy. No requests can be issued to this device until the other device completes.

`BZ` Device is busy processing some read, write, or control request. For a buffered or unbuffered user request, [*proga*] identifies the issuing program by name unless it has gone dormant.

`AB` Device is busy processing an abort request.

`SY` Device is busy with a system request.

`CL` Device is busy with a Class I/O request.

`LK` Device has been locked by an LURQ request from [*progb*]. The lock may be shared by cooperating programs.

Other FMGR Commands

NO TAG shows other FMGR commands and their CI equivalents.

. FMGR/CI Commands

FMGR *	CI
AS	AS
BR	BR
CN	CN
GP	GO
IO	IO
IT	AT
OF	OF
ON	AT
PL	WH
PR	PR
PS	PS
RP	RP
SS	SS
SZ	SZ
TM	TM
TO	TO
UL	UL
UP	UP
VS	VS
WS	WS
XQ	XQ

* The FMGR commands listed in this table are documented under FMGR in the Commands chapter of the *RTE-A Quick Reference Guide*, part number 92077-90020.

COMND

The COMND program is a compact version of FMGR that only supports a subset of FMGR commands as listed in NO TAG below. Reference NO TAG for CI equivalents.

. COMND Commands

Command:
BL
CN
IO
PL
TM
TO
IT
ON
EX

C

CS/80 Exerciser Utility (EXER)

Introduction

EXER is a CS/80 Exerciser Utility program used to diagnose and troubleshoot CS/80 disk drives on HP 1000 systems. It is available as an offline program and an online program. This appendix covers using the EXER utility in an online environment. The offline version of EXER is documented in the *CS/80 External Exerciser Reference Manual*, part number 5955-3462.

The online version of EXER is essentially the same type 6 program file as in the offline version. The offline version for RTE-A runs in a memory-based RTE-A system environment.

Getting Started

If you have installed a Primary system, EXER is already loaded as a program under /PROGRAMS. In actuality there are two programs required: EXER, the parent, and EXER1, the child, which is scheduled by EXER as needed. Before executing EXER, the child program EXER1 must be RP'ed, otherwise SC05 errors result.

```
CI> rp,exerl
CI> exer
```

At this point you are in the exerciser. At any point in the program, the command "HELP" will list valid commands. (A command file can be used to RP the EXER1 program and run EXER, simplifying the process. Refer to Chapter 2 for command file usage.)

Loading the Program

With RTE-A software, the relocatables EXER.REL and EXER1.REL are supplied, along with link command files #EXER and #EXER1.

The command file RTEA2.CMD, which loads RTE-A programs, loads EXER and EXER1 automatically.

Using the Exerciser

When you run EXER, the first thing the program wants to know is what disk LU to test. The program scans through the device reference table looking for all entries with a driver type 33B or 26B (disk or tape). It then reports all the LUs that match and asks you to select one. Realize that EXER itself does *not* concern itself with individual LUs, it merely needs to differentiate between multiple disks connected to the system. Once it is given *any* LU on a given disk, EXER looks at the disk as *one physical* volume. All subsequent EXER commands will address the *entire* disk.

The following is a sample of the LU table reported by EXER for a 5.2 Primary system:

```
CS/80 EXTERNAL EXERCISER -- Rev. 5020  02-07-90

*********  CS80 LU s  ************
Sel Code   LU #   HPIB Addr   Unit
========   ====   =========   ====
   26       12         1         0
            41         7         0
            42         7         0
            43         7         0
            61         7         1
   27        9         1         0
            16         0         0
            17         0         0
            18         0         0
            19         0         0
            20         0         0
            21         0         0
            22         0         0
            23         0         0
            24         0         1
            25         0         0
            26         0         0
            29         0         1
            30         0         1
            31         0         1
            36         5         0
            37         5         1
            50         2         0
            51         2         0
            52         2         0
            53         2         1
            54         2         1

Input DRIVE LU?
```

Note EXER only looks for LUs up to 63. For DataPair systems using LU numbers greater than 63 (which is recommended for DataPair systems) EXER will not find them. You will not be able to run EXER successfully in this case.

At this point, you must check the list for the appropriate LU that corresponds to the disk you wish to examine. For example, if you wanted to examine the boot disk for this Primary, you could enter "16" as the LU. Since EXER does not care about LUs, you could use *any* LU 16−26, or 29−31 since they all correspond to the same physical disk drive (HPIB address 0, on select code 27B).

The next thing EXER does is attempt to identify the device type attached, with the result appearing as follows:

```
Input DRIVE LU? 16

LU 16 is a 7914

Current unit = 0
```

If the identify function fails, for example, if the LU specified is turned off or disconnected, then the following message is displayed:

```
Error on initial describe, please check drive.

Input DRIVE LU?
```

Note that a "broken" disk could also cause this error. EXER then asks for the LU again. Enter a valid (existent LU) .

Sometimes, entering a non-existent LU will cause a "hang" waiting for the CS/80 timeout. Be patient. EXER will come back. OF'ing EXER may not clear the timeout.

After EXER has ID'ed the disk successfully, it will display the prompt:

```
EXER>
```

Typing "HELP" at this point will display all available commands. Note that commands do not need to be entered in their entirety, for instance "H" will suffice for "HELP".

The following are valid EXER commands:

```
EXER> help

CHANGE LU       – change the lu that you are working on
CANCEL          – cancel transaction
CICLEAR         – channel independent clear
DESCRIBE        – describe selected unit
ERT LOG         – output error rate test log
EXIT            – exit program or command
FAULT LOG       – output fault log
HELP            – output help information
INPUT           – change input file or lu
OUTPUT          – change output file or lu
PRESET          – update device logs
REQSTAT         – request status
REV             – output firmware revision
RF SECTOR       – read full sector
RO ERT          – perform read-only error rate test
RUN LOG         – output run log data
TABLES          – output device tables
TERM            – input/output at terminal
UNIT            – set unit number
```

Selected Command Descriptions

A full description of all the EXER commands is available in the *CS/80 External Exerciser Reference Manual*, part number 5955-3462. The commands most useful to a System Manager or programmer who desires to check the condition of a disk drive are briefly described below.

CHANGE LU This command allows testing another disk drive without exiting EXER.

DESCRIBE As the name implies, this gives a description of the disk drive. For example:

```
DESCRIBE UTILITY
LU 16 is a 7914

Current unit = 0

MODEL: 7914
UNIT: 0
TYPE: DISC
Maximum cylinder address =          1151
Maximum head address =                6
Maximum sector address =             63

Maximum block address =           516095
Current interleave factor = 1
```

ERT LOG This command displays the Error Rate Test log information for the drive. This log is used *only* by the exerciser and does not reflect usage from the system (RTE). See RUN LOG.

```
READ ERT LOG UTILITY
LU 16 is a 7914

Current unit =  0

Input the head (0 – 6) or ALL? 0

Head # = 0
# sectors read = 0
Correctable errors = 0
Uncorrectable errors = 0
 No errors logged
```

FAULT LOG This displays *all* faults logged by the disk during system or exerciser operation.

```
FAULT LOG UTILITY
LU 16 is a 7914

Current unit =  0

No drive faults
```

RUN LOG This displays the error log information from online system usage. This is useful information for the system manager to ascertain disk usage and error rates.

```
READ RUN LOG UTILITY
LU 18 is a 7914

Current unit =  0

Input the head (0 – 6) or ALL? 0

Head # = 0
# sectors read = 429483
Correctable errors = 0
 No errors logged
```

TABLES This will display information on the number of sparing operations that have been performed on the drive with the FORMC utility from the system. By definition, a "secondary" spare is any user-invoked spare operation. (Primary spares are factory done and are not ascertainable.) Note that all sparing is handled by the disk controller, not RTE. For more information on sparing tracks, see the description of FORMC in the *RTE-A Backup and Disk Formatting Utilities Reference Manual*, part number 92077-90249.

```
READ DRIVE TABLES UTILITY
LU 19 is a 7914

Current unit =  0

SPARE TRACK TABLE

Head number =  0
# of secondary spares =  4
# of tracks used =  2
# of logical tracks spared =  1
```

```
 CYL       TYPE       SCALAR
=====   =========    =======
 491    SECONDARY       50

Head number =  1
# of secondary spares =  0
# of tracks used =  0
# of logical tracks spared =  0
```

REV This reads the revision level of the firmware in the disk microproessor. This may be useful information for field service personnel for troubleshooting purposes.

```
READ REVISION NUMBER UTILITY
LU 16 is a 7914

Current unit =  0

Part   Revision
number  number          *NOTE   These part numbers refer
------ --------                 to the individual firmware
   1    5 - 0                   ROMs on the processor card.
   2    5 - 1
   3    5 - 0
   4    5 - 0
   5    5 - 0
   6    5 - 0
```

RO ERT This command performs a Read Only error rate test on the disk volume. The test area used is determined by user input, regardless of the LU entered when EXER was started.

Caution Executing an RO ERT will lock the disk to EXER for the duration of the test. *Do not* use a loop count of INFinite for the RO ERT command as this will hang the disk and require a reboot of the system.

INPUT Allows use of an "answer" file to supply the responses to EXER, allowing the program to be scheduled automatically from CI. See OUTPUT command.

OUTPUT Allows the output from EXER to be directed to a disk file for later examination by the user. When used with the INPUT command, EXER could be time-scheduled, get its answers from a command file (INPUT) and store the results in a file (OUTPUT) for viewing later.

Error Handling

Many errors occurring during the execution of EXER can be attributed to either user input error (wrong LU number) or certain normal conditions, such as loading a tape in the drive (for example, HP 7914 with built-in CTD). For the most part these can be ignored.

As a rule, the errors seen by looking at the fault and run logs require a much greater knowledge of the hardware than the average user possesses. These types of errors are best left to an HP service representative to analyze.

If you are merely checking the error logs on your disk as a preventative measure, without experiencing any disk related system problems, the majority of any errors logged will be inconsequential. Also, different CS/80 disks have different allowable error rates, and certain limits for certain fault conditions. These are best left to trained service personnel to interpret.

The absolute best preventative measure you can take is a regular schedule of system backups. Since the CS/80 drives are relatively easy and fast to repair, your primary concern should be having backups, not downtime.

EXER and CS/80 Tape Drives

The online EXER program is not intended to be used for either built-in tape drives in the HP 791x disk family or HP 9144 standalone drives. (The offline version of EXER contains the additional program TAPE, which is not available online.)

EXER and Cache Disks (HP 793xXP)

RTE-A supports the Cache version of the HP 793x family of CS/80 disks. If you have a Cache version drive, EXER determines this when it ID's the drive during the initial DESCRIBE command, and will allow access to several additional commands.

These additional commands allow the user to check certain statistics regarding the disk controller cache. They also allow the user to turn caching on and off. This can be useful in determining whether a particular application is really benefitting from the presence of the cache disk. Typical HP 1000 systems *do not* benefit from the cache, they may even slow down overall throughput because of the additional overhead required for the caching feature. Consult your local HP service representative for more information regarding disk caching.

EXER and SubSet 80 Disks

SubSet 80 refers to a group of disk drives that are a "subset" of the CS/80 family. These include the HP 9133D/H/L and the HP 9153B/C (including the internal disk in the HP 12122A). As far as RTE-A is concerned, these drives are treated just like CS/80 disks. The EXER utility *does not* support these drives. Most commands will return an "Illegal Opcode error".

Index

Symbols

? (help) command, CI, 2-3, 6-3
"." and ".." directory specifiers, 3-9
#n directory specifier, 3-10
$AUTO_LOGOFF variable, 2-16
$CMNDO variable, 2-17
$COLUMNS variable, 2-17, 7-8
$DATC variable, 2-17
$EVB_SIZE variable, 2-17
$FRAME_SIZE variable, 2-17
$HOME variable, 2-17
$IFDVR variable, 2-17
$KILLCHAR variable, 2-17
$LINES variable, 2-18
$LOG variable, 2-18
$LOGON variable, 2-18
$MY_NAME variable, 2-18
$OLDPWD variable, 2-18
$OPSY variable, 2-18
$POLL variable, 2-18
$POLLINT variable, 2-18
$PROMPT variable, 2-18
$QUIET_CMD variable, 2-19
$REPROMPT variable, 2-19
$RETURN_S variable, 2-19
$RETURN1 – $RETURN5 variables, 2-19
$RU_FIRST variable, 2-19
$SAVE_STACK variable, 2-20
$SESSION variable, 2-20
$VISUAL mode command line editing, 7-8
$VISUAL variable, 2-20
$WD variable, 2-20
* (comment), transfer files, 6-3
/ command stack editor, 7-1
/USERS directory, 5-4
~ substitution, 2-21

A

A990 Computer, clock, 6-23
AB2MI, 6-4
  error messages, 6-5
alias
  defining, 6-6
  deleting, 6-6, 6-187
  displaying, 6-6
ALIAS command, 6-6
AS command, 4-10, 6-8
ASK command, 6-9
assigning
  partitions, 4-10
  programs to partitions, 6-8
AT command, 4-5, 6-11

B

base page, links, in source BOOTEX, 6-89
base set command, 6-1
  RU command, 6-155
  TM command, 6-180
  XQ command, 6-207
bit map, 3-36
  displaying, MPACK, 6-117
  free space table, 6-73
bootable system installation utility (FPUT), 6-62
  FPUT operation, 6-63
BOOTEX, 6-87
  and FMGR, B-10, B-28
  initializing, 6-87
  installing, 6-62
  VCP/loader ROM, 6-87
BR command, 4-7, 6-13
breaking program execution, 4-7, 6-13
bringing up a device, 2-11, 6-190
buffer limit, displaying and modifying, FMGR, B-64

C

CALLM utility, .include directive, 6-15
CALLS utility
  catalog file, 6-17
  directives, 6-17
  index file, 6-19
  online help facility, 6-16
  relating topics to other topics, 6-18
capability levels, 5-5
  for CI commands, 6-1
  within commands, 5-6
cartridge, initializing in FMGR, B-11
CD command, 2-24, 4-12, 6-20
CDS
  displaying code partition size, 6-44
  displaying data partition size, 6-50
  modifying code partition size, 6-44
  modifying data partition size, 6-50
  program, 4-11
    memory requirement, 4-12
changing
  associated group, 3-31
  CDS program memory requirement, 4-12
  code partition size, 6-44
  data partition size, 6-50
  device timeout, 6-181
  directory owner, 3-31
  directory protection, 3-33, 6-139
  file protection, 3-27, 6-139
  I/O device attributes, 2-11
  memory requirements, 4-9
  program priorities, 4-9, 6-138

program size, 6-179
subdirectory protection, 3-33
UDSP, 6-134
VMA size, 4-11, 6-191
VMA working set size, 6-206
working directory, 6-20, 6-194
character count, 6-192
checking consistency on a CI file system disk LU, 6-73
CI commands
? (help), 6-3
* (comments), 6-3
ALIAS, 6-6
AS, 6-8
ASK, 6-9
AT, 6-11
BR, 6-13
CD, 6-20
CL, 6-22
CLOCK, 6-23
CN, 6-24
CO, 6-26
CP, 6-29
CR, 6-32
CRDIR, 6-35
CRON, 6-37
CRONTAB, 6-38
CZ, 6-44
DC, 6-45
DL, 6-46
DT, 6-50
ECHO, 6-51
EX, 6-52
FGREP, 6-80
FUNCTION, 6-70
FUNCTIONS, 6-72
GO, 6-79
GREP, 6-80
IN, 6-85
IO, 6-90
IS, 6-95
KTEST, 6-96
LI, 6-98
LNS, 6-103
MC, 6-108
MO, 6-116
MV, 6-124
OF, 6-128
OWNER, 6-133
PATH, 6-134
POLL, 6-137
PR, 6-138
PROT, 6-139
PS, 6-141
PU, 6-143
PWD, 6-145
RETURN, 6-147
RM, 6-148
RN, 6-149
RP, 6-150
RS, 6-152
RU, 6-153
SET, 6-171
SS, 6-176
SYSTZ, 6-177
SZ, 6-179
TM, 6-180
TO, 6-181
TOUCH, 6-182
TR, 6-184
UL, 6-186
UNALIAS, 6-187
UNPU, 6-188
UNSET, 6-70, 6-189
UP, 6-190
VS, 6-191
WC, 6-192
WD, 6-194
WH, 6-195
WHOSD, 6-204
WS, 6-206
XQ, 6-207
CIX program, 6-27
CL command, 6-22
class, table, displaying information, 6-198
clock
A990 Computer, 6-23
system, 6-180
CLOCK command, 6-23
clock daemon (CRON), 6-37
CLOSE utility, 3-43
CM program, copy of CI, 1-5
CMNDO monitor, 7-17
CN command, 6-24
CO command, 3-20, 3-24, 3-39, 6-26
code partition, 4-12
displaying/modifying size, 6-44
code segment, 4-12
command, capability levels, 5-6
command editing, 7-1
$VISUAL mode, 7-8
CMNDO monitor, 7-17
CSH visual editing mode, 7-17
EMACS/GMACS visual editing mode, 7-9
VI visual editing mode, 7-13
Command Interpreter (CI), 1-1
CM, copy of CI, 1-5
features, 1-2
termination, 1-5
unavailability, 1-5
command stack
editor, 7-1
posting contents, 3-31
using, 2-4
commands and command files
command descriptions, 6-1
editing commands, 7-3
entering multiple commands in one line, 2-24
executing a command file, 2-13
execution control, 2-25
file manipulating commands, 3-2, 3-3
nesting, 2-21

quoting, 2-23
return status, 2-24
returning from, 6-147
sample command file, 2-13
transferring to command file, 6-184
comment, transfer files, 6-3
COMND program, B-64
commands, B-71
compacting files, 6-117
comparing
strings or numbers, IS command, 6-95
two files, SCOM, 6-159
concatenate
many files into one, MERGE, 6-109
programs and subroutines, 6-109
continuous streaming mode, 6-41
control, file, FMGR, B-51
control commands, program, 4-1
control structures
execution, 2-25
IF-THEN-ELSE-FI, 2-24, 2-25, 6-83
WHILE-DO-DONE, 2-24, 2-25, 6-203
controlling
devices, 2-10, 6-24
programs, 4-1
converting, FMGR directory structure, 6-67
copy of CI (CM program), 1-5
Copy System utility (CSYS), 6-40
error messages, 6-43
examples, 6-41
loading CSYS, 6-42
operation, 6-41
copying
CO command, 6-26
CP command, 6-29
files, 3-24, 6-26
files and directories, 6-29
memory image files to tape, 6-40
recursive, 6-29
to/from devices, 6-27, 6-30
type 1 files, 6-27, 6-30
type 2 files, 6-27, 6-30
variable-length file, 6-27, 6-30
count, line, word, char, 6-192
CP command, 6-29
CR command, 3-26, 6-32
CRDIR command, 3-29, 6-35
creating
a directory, 3-29, 6-35
a disk file, 6-32
a subdirectory, 3-30, 6-35
empty files, 3-26
symbolic links, 3-28
LNS command, 6-103
user accounts, 5-5
CRETS routine, B-34
CRON command (clock daemon), 6-37
CRONTAB command, 6-38
CS/80, cartridge tape drive, copying to, 6-40
CS/80 Exerciser utility. *See* EXER utility
CSH visual editing mode, 7-17
CZ command, 6-44

D

data
storage requirements, 4-11
transfer to and from devices, 3-39
data partition, 4-10
displaying/modifying size, 6-50
daylight savings time, 6-177
DC command, 3-36, 6-45
default, search sequence, 3-34
defining
UDSPs, 3-34
variables, 6-171
deleting
aliases, 6-187
functions, 6-70
user-defined variables, 6-189
destination file masks, 3-20
device
bringing up, 2-11
changing attributes, 2-11
changing timeout values, 2-12
control functions, 2-10, 6-24
controlling, 2-10
copying to/from, 6-27, 6-30
manipulation, in FMGR, B-64
referenced as a file, 3-2
status, displaying from FMGR, B-69
timeout, 2-12
displaying/modifying, 6-181
transferring data to/from, 3-39
differences between FMGR and CI files, 3-40
directory, 3-6
copying, 6-29
creating, 3-29, 6-35
default (WD), 3-6
listing, 3-22, 6-46
manipulating, 3-29
moving, 3-31
on FMGR, B-1
ownership, 3-31
protection, 3-33, 6-139
purging, 3-33
report user of, 6-204
specifiers, 3-9, 3-10
working (WD), 3-6, 3-30
disk
caching, 6-41
dismounting a volume, DC command, 6-45
initializing, IN command, 6-85
manipulation, from FMGR, B-7
mounting, MC command, 6-108
packing, FPACK, 6-57
problems
volume dismounted, 8-5
volume full, 8-3
space, reporting amount used, 6-54
disk packs, B-8
dismounting volumes, 3-36, 6-45

displaying
A990 Computer clock, 6-23
aliases, 6-6
CPU usage, METER, 6-111
device status, from FMGR, B-69
device timeout, 6-181
directory owner, 3-31
directory protection, 3-33, 6-139
disk space used, FOWN, 6-54
free disk space, FREES, 6-64
I/O configuration, 2-8, 6-90
memory usage, 2-8
parameters at terminal, 6-51
program size, 6-179
program status, 2-6, 4-8, 6-141
prompt and read response, 6-9
resource number information, 6-200
system
clock, 6-180
time-zone offset, 6-177
status, 2-6
time, 2-13
UDSP, 6-134
variables, 6-171
VMA size, 6-191
VMA working set size, 6-206
working directory, 3-30, 6-145, 6-194
Distributed System (DS) Network, 3-41
See also DS
DL command, 3-22, 6-46
down device, bringing up, 2-11
DS
file access, 3-41
file access considerations, 3-43
software errors, A-23, A-24
DT command, 2-24, 4-12, 6-50
dynamic, memory
allocation, 4-9
partitions, 4-10

E

ECHO command, 2-24, 6-51
echoing commands, B-5, B-6
EDIT/1000, 1-1
EMA
maximum size in pages, 4-13
models, 4-13
partition, 6-186
shareable, 6-201
EMACS/GMAC visual editing mode, 7-9
empty file, creating, 3-26
enlarge free space areas on disks, 6-117
Environment Variable Block (EVB), 6-6, 6-70, 6-171
environment variables, 2-16
error
formats, A-1
logging (VC+), 2-26
messages, A-1
messages and codes, A-1, A-4, A-13
EX command, 1-5, 6-52
exception condition handling, 8-1
executing a program, 4-3
EXER utility, C-1
error handling, C-8
EXER and cache disks (793xXP), C-8
EXER and CS/80 tape drives, C-8
EXER and subset 80 disks, C-8
loading the program, C-2
selected command descriptions, C-5
CHANGE LU, C-5
DESCRIBE, C-5
ERT LOG, C-5
FAULT LOG, C-6
INPUT, C-7
OUTPUT, C-7
REV, C-7
RO ERT, C-7
RUN LOG, C-6
TABLES, C-6
using the Exerciser, C-2
exiting, CI, 6-52
exported function, 6-70
exported variable, 6-6, 6-171
extended record converter utility (OLDRE), 6-129
error messages, 6-132
extended records, 6-129
FORTRAN code, 6-132
FORTRAN code processing, 6-130
Macro code, 6-131
Macro code processing, 6-130
operation, 6-129
Pascal code, 6-131
program restrictions, 6-131
translation results, 6-130
extended record, definition of, 6-129
extended system available memory (XSAM), 6-156
extent, file, 3-12, 6-57

F

FC, 1-1
FGREP command, 6-80
file
comparison, SCOM, 6-159
ownership, reporting using FOWN, 6-54
packing, MPACK, 6-117
renaming during conversion, FMGR, 6-68
file manipulation, in FMGR, B-33
command summary, B-37
file manipulation utilities
compare files (SCOM), 6-159
concatenate files (MERGE), 6-109
extended record converter (OLDRE), 6-129
summary of, 3-38
file system
conversion, FSCON, 6-67
verification, FVERI, 6-73
file system utilities
file compacting and disk pack (MPACK), 6-117

File System Conversion (FSCON), 6-67
file system pack (FPACK), 6-57
file system verification (FVERI), 6-73
report disk Free Space (FREES), 6-64
report file space by owner (FOWN), 6-54
summary of, 3-38
files
access errors, 8-1
cannot create directory, 8-2
cannot open file, 8-2
file not found, 8-2
non-standard file names, 8-1
OWNER, PROT, or WD command failures, 8-3
same directory and cartridge name, 8-2
clearing open ones, 8-5
command, 2-13, 2-21
copying, 3-24, 6-26, 6-29
creating a disk file, 6-32
creating empty, 3-26
descriptors, 1-4, 3-5
destination masks, 3-20
directories, 1-4, 3-1, 3-6
directory specifiers, 3-9, 3-10
extents, 3-12
FMGR, 3-20, 3-40
I/O devices referenced as, 3-2
identification, 3-1
introduction to, 1-4
length, 3-1
listing, 3-23, 6-98
manipulating, 3-1
manipulating commands, 3-2, 3-3
mask characters, 3-16
masks, 3-14, 3-20
moving, 3-25, 6-116, 6-124
names, 1-4, 3-1, 3-4, 8-1
open, clearing, 8-5
operations, 3-22
ownership and associated group, 3-13
properties, 1-4, 3-1
protection, 3-1, 3-13, 3-27, 6-139
purging, 3-25, 6-143
record length, 3-1, 3-12
remote, 3-41, 3-43
removing, 6-148
renaming, 3-24, 6-124, 6-149
restoring program, 6-150
searching for, 3-34, 3-35
searching for a pattern, 6-80
size, 3-1, 3-12
subdirectories, 1-4, 3-8
temporary, 3-2
time stamps, 3-1, 3-14
type extensions, 1-4, 3-4
types, 3-1, 3-11
unpurging, 3-26, 6-188
unusual access errors, 8-1
updating times, 6-182
FMGR, B-1
accessing a disk file, B-34
cartridge directory, B-10
cartridge file directory, B-10
cartridge initialization, B-11
cartridges and cartridge reference numbers, B-8
changing the master security code, B-12
configuration of logical units/cartridges, B-8
control commands, B-2
??, B-3
LL, B-4
LO, B-5
SV, B-5
converting directory structure, 6-67
creating a file, B-35
device manipulation, B-64
calling FMGR and COMND, B-64
DN, B-68
downing a device, B-64
device manipulation commands, BL, B-66
disk logical units and LU numbers, B-7
disk manipulation, B-7
disk manipulation commands, B-15
CL, B-16
CO, B-17
DC, B-23
DL, B-24
IN, B-27
MC, B-30
PK, B-31
errors, B-2
file manipulation commands
CR, B-37
DU, B-40
LI, B-42
PU, B-44
RN, B-45
ST, B-45
files, 3-20, 3-40, B-1
handling, 3-40
list device, B-2
listing the contents of a file, B-36
log device, B-2
masking, 3-20
mounting and dismounting cartridges, B-9
new cartridge initialization, B-28
packing a file cartridge, B-14
program, 1-1, 3-40
purging files, B-35
purging files on a cartridge during re-initialization, B-13
re-initializing a cartridge, B-12
records and file types, B-33
renaming files, B-36
scratch files, B-34
severity code, B-2
storing data on a device or new file, B-36
transferring files between disk cartridges, B-15
FMP, error code, A-13
FOWN, 6-54
examples, 6-55
FPACK, 6-57
moving directories, 6-59

moving files, 6-61
moving subdirectories, 6-60
packing process, 6-57
free space table
See also bit map
on an FMGR cartridge, 6-67
FSCON, 6-67
conversion process, 6-67
error messages, 6-69
renaming during conversion, 6-68
requirements for conversion, 6-67
functions
defining, 6-70
deleting, 6-70
exporting, 6-70
FUNCTION command, 6-70
FUNCTIONS command, 6-72
importing, 6-70
FVERI, 6-73
error messages, 6-76
error recovery, 6-75

G

G-type global parameters, B-5, B-51
setting, B-61
getting help, 2-3
GO command, 4-8, 6-79
GREP command, 6-80
group, accounts, 5-4

H

halt errors, 8-6
help command, 2-3
CI, 6-3

I

I/O
command, 2-8
configuration, displaying, 2-8, 6-90
devices
changing attributes, 2-11
controlling, 2-10
referenced as files, 3-2
errors, A-2
ID, segment, 4-2, 4-3, 4-5, 4-11
displaying prototypes, 6-199
identifying programs, 4-2, 5-8
IF-THEN-ELSE-FI, 1-5, 2-24, 6-83
imported function, 6-70
imported variable, 6-6, 6-171
IN command, 3-38, 6-85
increasing, free space on disk, 6-57
initializing
a cartridge in FMGR, B-11
BOOTEX, using INSTL, 6-86
disk volume, IN command, 6-85
disk volumes, 3-38
installing, bootable systems and diagnostics, 6-62
INSTL, 6-86
error messages, 6-89
IO command, 1-5, 6-90
IS command, 6-95

K

KTEST command, 6-96

L

LI command, 3-23, 6-98
library, of relocatables, forming, 6-109
LIF, 1-1
line count, 6-192
LINK, 1-1, 4-3, 4-9, 4-11, 4-12
listing
data from I/O LU, 3-39
directory, 3-22, 6-46
files, 3-23, 6-98
memory partition, 6-199
mounted disks, 6-22
volumes, 3-37
LNS command, 3-28, 6-103
locking, WH in memory, 6-202
logging on/off, 5-1
logoff/timeout function, 2-25
logon, errors, unusual, 8-3

M

M1KSS, 6-127
manipulating
directories, 3-29
files, 3-1
volumes, 3-36
mask qualifier, 3-15
masking
and FMGR files, 3-20
characters, 3-16
destination file, 3-20
file masks, 3-14
qualifiers, 3-16, 3-21
master security code, B-12
FMGR, B-44
MC command, 3-36, 6-108
memory
partitions, 4-10, 6-199
requirements
changing, 4-9
for CDS, 4-12
usage, displaying, 2-8
memory image format, 6-4
converting to, AB2MI, 6-4
MERGE, 1-1
METER
displaying process information, 6-112
loading, 6-113
output example, 6-112
MI2AB, 6-114
error messages, 6-115

mini-cartridge bootstrap loader, 6-114
MO command, 3-20, 3-25, 6-116
models, EMA/VMA, 4-13
modifying, user accounts, 5-5
mounting disk volumes, 3-36, 6-108
moving
  directories, 3-31
  files, 3-25, 6-116, 6-124
MPACK
  examples, 6-123
  logging option, 6-122
  options, 6-117
  visual mode, 6-121
multiple commands per line, entering, 2-24
multiuser remote file access, 3-42
multiuser session operation (VC+), 5-1
  capability levels, 5-5
  creating and modifying accounts, 5-5
  group accounts, 5-4
  identifying programs, 5-8
  log on, log off, 5-1
  running out of SAM, 5-8
  session handling, 5-7
  superusers, 5-5
  user accounts, 5-4
MV command, 6-124

N

native language support utilities errors, A-24
nesting, command files, 2-21
NOTIFY utility
  and M1KSS monitor program, 6-127
  defining aliases for notification, 6-126
    alias definition format, 6-127
    alternate logons for a user, 6-126
    group distribution lists, 6-126
    hosts in a DS network, 6-127
  installed by Mail/1000, 6-125
  sending a message, 6-125
  turning notification on or off, 6-125
  using the NOTIFY utility, 6-125

O

OF command, 1-5, 2-24, 4-2, 4-7, 6-128
online, help summary, 2-3
open files, clearing, 8-5
overview of RTE-A features, 1-1
OWNER command, 3-37, 6-133
ownership
  changing, 3-31, 6-133
  directory, displaying, 3-31, 6-133
  file, 3-1, 3-13
  volume, 3-37

P

P-type global parameters, B-53
packing, disk
  FPACK, 6-57
  MPACK, 6-117
parity, error, 8-6
  message, A-4
partition
  assigning, 4-10, 6-8
  displaying/modifying
    code size, 6-44
    data size, 6-50
  dynamic, 4-10
  reserved, 4-10
  shareable EMA, 6-186
password, 5-1
PATH command, 6-134
POLL command, 6-137
polling function, 6-137
positional variables, 2-14
power fail, 8-9
PR (set priority) command, from CI, 2-24, 4-9, 6-138
precedence within CI, 6-2, 6-6, 6-70
predefined variables, 2-16
PRINT utility, 1-1
priority, program, 4-2, 4-9
program(s)
  abort errors, A-3
  breaking execution, 4-7, 6-13
  changing priorities, 4-9, 6-138
  control command summary, 4-1
  controlling, 4-1
  displaying status, 2-6, 4-8, 6-141
  executing, 4-3
  ID
    duplicating, 4-6
    segment description, 4-2
    segments, 4-5
  identification, 4-2, 5-8
  memory requirements, 4-9
  priorities, 4-2, 4-9
  protection, 5-6
  protection through, capability levels (VC+), 5-6
  removing, 4-7, 6-128
  restarting, 4-8, 6-152
  restoring, 1-4, 4-5, 6-150
  resuming execution, 4-8, 6-79
  running, 4-2, 6-153
    with wait, 4-3
    without wait, 4-4, 6-207
  scheduling, 4-2, 4-5
  set run time, 6-11
  shared, 4-11
  size, display/modify, 6-179
  stopping, 6-128
  suspending, 4-8, 6-176
  system, missing, 8-7
  time scheduling, 4-5
  typical RTE-A interactive, 1-1
  unrestored, 1-4
prompt
  logon, 5-1
  RTE, 1-6, 8-8
  system, 8-8

properties, file, 3-1
PROT command, 3-27, 3-33, 6-139
protection
  directory, 3-33, 6-139
  file, 3-27, 6-139
  volume, 3-37, 6-139
prototype ID segments, 4-6, 6-199
PS command, 2-24, 4-8, 6-141
PU command, 3-25, 3-33, 6-143
PURGE routine, B-34
purging
  directories, 3-33
  files, 3-25, 6-143
PWD command, 6-145

Q

quoting, 2-23

R

reassign file-protection status, 3-27
record length, 3-12
recursive copy, 6-29
redirection, FGREP/GREP, 6-81
remote files
  access limitations, 3-44
  closing, 3-43
  considerations, 3-43
  multiuser, 3-42
  specifying, 3-41
removing
  files, 6-148
  programs, 4-7, 6-128
renaming files, 3-24, 6-124, 6-149
reserved, partition, changing memory requirements, 4-9
resource number, display information on, 6-200
restart a program, 4-8, 6-152
restoring, programs, 4-5, 6-150
resume program execution, 4-8
resume suspended program, 6-79
RETURN command, 2-24, 6-147
return status, 2-24
RM command, 6-148
RN command, 3-20, 3-24, 6-149
RP command, 4-2, 4-6, 6-150
RS command, 1-5, 4-8, 6-152
RTE prompt, 1-6, 8-8
RU command, 3-34, 4-2, 6-153
running
  a program, 4-2
    RU command, 6-153
  a program with wait, 4-3
  a program without wait, 4-4, 6-207
running out of SAM (VC+), 5-8

S

SAM
  displaying status of, SAM utility, 6-156
  running out of, 5-8
SAM utility, 6-156
  loading, 6-158
  returned values, 6-157
  running SAM with AL, 6-157
  running SAM without AL, 6-156
scheduling, a program, 4-2, 4-5
SCOM
  compare operation, 6-162
  error messages, 6-170
  examples, 6-163
  returned values, 6-162
  status interrogation, 6-163
search sequence, 3-34
  specifying a directory, 3-34
searching, for files, 3-34
Security/1000, 5-1
serial port, display status of, SPORT, 6-173
session, handling, 5-7
SET command, 2-15, 2-16, 2-24, 6-171
set up a working directory, 3-30
setting
  A990 Computer clock, 6-23
  daylight savings time flag, 6-177
  program run time, 6-11
  system clock, 6-180
  time zone offset, 6-177
shareable EMA, 6-186, 6-201
shared, programs, 4-11
SHEMA, maximum number of, 4-13
SP command, 2-24
specifying
  directories, 3-6, 3-34
  remote files, 3-41
  subdirectories, 3-8
  UDSPs in file descriptors, 3-35
SPORT, 6-173
  examples, 6-173
  including in a user program, 6-174
  loading, 6-175
SS command, 4-8, 6-176
stopping, a program
  using BR command, 4-7
  using OF command, 6-128
  using SS command, 6-176
streaming mode, 6-41
subdirectories, 3-8
  creating, 3-30
subfile marks (FMGR ST command), B-47
superuser
  capabilities, 1-3
  with Security/1000, 5-1, 5-5
suspend program execution command SS, 4-8
suspending a program, 4-8, 6-176
symbolic debug, 1-1
symbolic links
  copying, 6-30
  creating, 3-28, 6-103
  LNS command, 3-28, 6-103
system
  clock, 6-180

commands, 2-1
information, 1-5
programs, missing, 8-7
prompt, 8-8
status obtaining, 2-6, 6-195, 6-202
time, displaying, 2-13
system setup utilities, 2-27
Absolute Binary to Memory Image (AB2MI), 6-4
bootable system installation (FPUT), 6-62
Copy System (CSYS), 6-40
initialize BOOTEX (INSTL), 6-86
Memory Image to Absolute Binary (MI2AB), 6-114
system status utilities
display CPU usage (METER), 6-111
Serial Port Analyzer (SPORT), 6-173
System Available Memory utility (SAM), 6-156
SYSTZ command, 6-177
SZ command, 2-24, 4-10, 4-12, 6-179

T

temporary files, 3-2
terminating, CI, 1-5, 6-52
TF utility, 1-1
time, system, 2-13, 6-180
time scheduling programs, AT command, 4-5
time stamps, 3-14
time zone offset, 6-177
timeout, 2-12
displaying/modifying, 6-181
during logoff, 2-25
timeout/logoff function, $AUTO_LOGOFF, 2-25
timeslicing, 4-9
TM command, 1-5, 6-180
TO command, 6-181
TOUCH command, 6-182
TR command, 1-5, 2-14, 3-34, 6-184
transfer files
* (comment), 6-3
G-type global parameters, B-51
in FMGR, B-51
calculating global parameters, B-56
displaying parameters, B-58
IF conditional skip, B-59
PA, B-61
setting global parameters, B-61
transfer control, B-62
P-type global parameters, B-53
transfer to command file, 6-184
transferring, data to/from devices, 3-39
translating extended relocatable record formats, 6-129, 6-130
type 1 files, copying, 6-27, 6-30
type 2 files, copying, 6-27, 6-30

U

UDSP
default, 3-10
defining, 3-34
display/modify, 6-134
numbers, 3-10
searching for files, 3-34
specifying in file descriptors, 3-35
used in restoring a program file, 4-3
UL command, 6-186
UNALIAS command, 6-187
unlock, shareable EMA partition, 6-186
UNPU command, 3-26, 6-188
unpurging files, 3-26, 6-188
UNSET command, 2-15, 2-16, 6-70, 6-189
up a device, 6-190
UP command, 1-5, 6-190
user, account, 5-4
creating/modifying, 5-5
user.group identification, 5-1
user-defined variables, 2-15, 6-189
using
RTE-A, 1-1
the system, 2-1

V

variable-length file, copying to, 6-27, 6-30
variables
$AUTO_LOGOFF, 2-16
$CMNDO, 2-17
$COLUMNS, 2-17
$DATC, 2-17
$EVB_SIZE, 2-17
$FRAME_SIZE, 2-17
$HOME, 2-17
$IFDVR, 2-17
$KILLCHAR, 2-17
$LINES, 2-18
$LOG, 2-18
$LOGON, 2-18
$MY_NAME, 2-18
$OLDPWD, 2-18
$OPSY, 2-18
$POLL, 2-18
$POLLINT, 2-18
$PROMPT, 2-18
$QUIET_CMD, 2-19
$RETURN_S, 2-19
$RETURN1 – $RETURN5, 2-19
$RU_FIRST, 2-19
$SAVE_STACK, 2-20
$SESSION, 2-20
$VISUAL, 2-20
$WD, 2-20
environment, 2-16
exported, 6-6, 6-171
imported, 6-6, 6-171
positional, 2-14
predefined, 2-16
user-defined, 2-15, 6-189
VC+ System Extension Package, 1-1, 1-2, 2-1, 3-6, 5-1
VCP, 6-40

and BOOTEX, 6-87
bootstring for SCSI or CS/80 disk, 6-62
halt, 8-6
VI visual editing mode, 7-13
virtual control panel (VCP). *See* VCP
virtual memory area. *See* VMA
visual mode, MPACK, 6-121
VMA
change VMA size, 4-11
changing size, 6-191
changing working set size, 6-206
displaying size, 6-191
displaying working set size, 6-206
maximum size in pages, 4-13
models, 4-13
volumes
initializing, 3-38, 6-85
listing, 3-37
manipulating, 3-36
mounting and dismounting, 3-36, 6-108
ownership and protection, 3-37, 6-139
user of, 6-204
VS command, 2-24, 4-11, 6-191

W

WC command, 6-192
WD command, 3-30, 6-194
WH
command, 1-5, 2-6, 2-8, 6-195
CL, 6-198
D, 6-199
PA, 6-199
RN, 6-200
SC, 6-199
SH, 6-201
ST, 6-202
locking in memory, 6-202
WHILE-DO-DONE, 1-5, 2-24, 6-203
WHOSD command, 6-204
word count, 6-192
working directory, 3-6, 3-30, 3-34
changing, CD command, 6-20
display/change, 6-194
working set
changing size, 4-11
changing size of, 6-206
displaying size of, 6-206
WS command, 2-24, 4-11, 6-206

X

XQ command, 4-3, 4-4, 6-207
XSAM, 6-156