RDOS/DOS

Command Line Interpreter

User's Manual

093-000109-01

For the latest enhancements, cautions, documentation changes, and other information on this product, please see the Release Notice (085-series) supplied with the software.

Ordering No. 093-000109

Printed in the United States of America
Revision 01, May 1978

DataGeneral
SOFTWARE DOCUMENTATION

NOTICE

RDOS/DOS
Command Line Interpreter
User's Manual
093-000109

Revision History:

Original Release - February 1975
First Revision - May 1978

This document has been extensively revised from revision 00; therefore, change indicators have not been used.

Preface

This manual describes the Command Line Interpreter, which is your primary interface with Data General's Real-Time Disk Operating System (RDOS), or Diskette Operating System (DOS). RDOS and DOS use the same CLI, except for a few features and commands; these are noted as "RDOS" or "DOS" in the text.

To use this manual properly; you'll need some experience with the CLI. If the CLI is new to you, read *Learning to Use Your RDOS/DOS System. Learning to Use* offers practical exercises with the CLI and other system utilities, and provides a background for both RDOS and DOS programming.

We have organized this manual as follows:

- Chapter 1 introduces the command mechanism and features of the CLI.
- Chapter 2 explains operating the CLI; it describes command line format, syntax, filename templates, console breaks, and error handling. An understanding of this chapter will allow you to code command lines and manage your files.
- Chapter 3 contains more advanced CLI features; it describes writing compact command lines, indirect commands, macros, and foreground/background programming.
- Chapter 4 presents the CLI commands by category and alphabetically. The categorical command summary is printed on yellow stock.
- Appendix A, which begins with a yellow page, contains an error summary. Other appendixes include an ASCII character set, a description of dump files, and an explanation of CLI command interpretation.

Certain CLI commands invoke *System Utilities* which can help you write and develop your own programs. Each utility has its own protocol and error messages, which are described briefly under its command entry in Chapter 3, and more extensively in one of the manuals listed below.

If you want a detailed explanation of RDOS or DOS, read the *RDOS Reference Manual* (ordering number 093-000075), or *DOS Reference Manual* (093-000201). Different aspects of RDOS and DOS are covered in four other books:

Introduction to RDOS (ordering number 093-000083).

RDOS/DOS User's Handbook (093-000105). This is a pocket-sized summary of CLI commands and system utility features.

How to Load and Generate Your RDOS System (093-000188). Data General supplies all the files you need for SYSGEN. DOS SYSGEN is covered in the *DOS Reference Manual.*

Learning to Use Your RDOS/DOS System (093-000223). This describes building programs in several different languages.

For more on a system utility, or other useful topics, consult the appropriate manual from the following list:

Extended ALGOL User's Manual (093-000052)
Extended Assembler User's Manual (093-000040)
Extended BASIC User's Manual (093-000065)
Extended BASIC System Manager's Guide (093-000119)
Batch User's Manual (093-000087)
Software Catalog (093-000106)
Symbolic Debugger User's Manual (093-000044)
FORTRAN IV User's Manual (093-000053)
FORTRAN IV Runtime Library User's Manual (093-000068)

FORTRAN 5 User's Manual (093-000085)
FORTRAN 5 Supplement (093-000185)
FORTRAN 5 Runtime Library User's Manual (093-000096)
FORTRAN Commercial Subroutine Package (093-000107)
Library File Editor User's Manual (093-000074)
Macroassembler User's Manual (093-000081)
Octal Editor User's Manual (093-000084)
Publications Catalog (012-000330)
Extended Relocatable Loaders (RLDR) User's Manual (093-000080)
RDOS Sort/Merge User's Manual (093-000108)
Real Time Input/Output System User's Manual (093-000095) (This explains the DGC analog-digital interface.)
Software Summary and Bibliography (093-000110)
Superedit User's Manual (093-000111)
Symbolic Editor (093-00160)
Text Editor User's Manual (093-000018)

Reader, Please Note:

We use these conventions for command formats in this manual:

COMMAND required *[optional]* ...

Where	Means
COMMAND	You must enter the command as shown.
required	You must enter some argument (such as a filename). Sometimes, we use: $\left\{\begin{matrix}\text{required}_1\\ \text{required}_2\end{matrix}\right\}$ which means you must enter *one* of the arguments. Don't enter the braces; they only set off the choice.
[optional]	You have the option of entering some argument. Don't enter the brackets; they only set off what's optional.
...	You may repeat the preceding entry or entries. The explanation will tell you exactly what you may repeat.

Additionally, we use certain symbols in special ways:

Symbol	Means
)	Press the RETURN key on your terminal's keyboard.
□	Be sure to put a space here. (We use this only when we must; normally, you can see where to put spaces.)

All numbers are decimal unless we indicate otherwise; e.g., 35_8.

Finally, we usually show all examples of entries and system responses in THIS TYPEFACE. But, where we *must* clearly differentiate your entries from system responses in a dialog, we will use

THIS TYPEFACE TO SHOW YOUR ENTRY)
THIS TYPEFACE FOR THE SYSTEM RESPONSE

Within this manual, the word "disk" means either disk or diskette.

We welcome your suggestions for the improvement of this and other Data General publications. To communicate with us, either use the postpaid remarks form at the end of this manual, or write directly to:

Software Documentation
Data General Corporation
Route 9
Westboro, MA 01581

End of Preface

Contents

Chapter 1 - Introduction

Chapter 2 - Operating the CLI

Chapter 3 - Advanced Features

Chapter 4 - CLI Commands

Appendix A - Error Messages

Appendix B - Standard ASCII Character Set

Appendix C - Dump File Format

Appendix D - Extended Uses of the CLI

Chapter 1
Introduction

The Command Line Interpreter is your primary interface with RDOS/DOS and their system utilities. From your viewpoint at the console, the CLI is a number of discrete commands which do helpful things with information - i.e., files. These commands provide the basis for maintaining a simple, productive, and happy relationship with RDOS or DOS and your computer. Figure 1-1 illustrates your relationship with the CLI.

Among other things, CLI commands allow you to:

- Develop your files into executable programs with the system utilities;
- Execute a program;
- Handle a file on disk, magnetic tape, or paper tape, or punched cards, and transfer any file between these media;
- Create random, contiguous, or sequential files (RDOS only) on disk;
- Combine many files into one file;
- Build a file containing filenames, selecting the filenames by name, date, or other criteria.
- Delete a file, or series of files, by name or date;
- Restrict access or permit conditional access to a file;
- Access a file from anywhere on disk via link entries;
- Organize files into logical subdivisions on disk;
- Write sequences of CLI commands into an indirect or macro file;
- Write descriptive messages to the console during the execution of indirect or macro files;
- Monitor console activity on your system via a log file;
- Back up your disk files by dumping them onto mag tape, cassette, paper tape, or other disks.

The CLI also protects data by aborting any command (except DELETE) whose execution would threaten an existing file, and returning an explanatory error message. Whenever it cannot execute any command, it displays an explanatory error message.

CLI Commands

A valid CLI command normally consists of the command name and one or more arguments. In many cases you can append switches to commands and arguments to modify command execution. The name of each command describes its function, for example:

```
CRAND FILE1 FILE2 FILE3)
```

CRAND is the command name; it is a contraction of "Create a RANDom file". This example creates three random files, with the names FILE1, FILE2, and FILE3.

Control Characters

If you make a mistake while typing, and you have not yet pressed the RETURN key, you can delete one character at a time by pressing the RUBOUT (or DEL) key. RUBOUT (or DEL) echoes as ← (or _) on teletypewriters. To delete the entire line, press the SHIFT and L keys simultaneously, or press the backslash key (\) on a DASHER or 6052/6053; the system will then echo a backslash (\). Then, re-enter the command.

To abort an executing CLI command, press the CTRL and A keys simultaneously. CTRL-A halts command execution and returns the CLI prompt. CTRL-A is a system interrupt, and it will return you to the CLI from all utility programs except editors. We describe other interrupts at the end of Chapters 2 and 3.

You can interrupt output to your console at any time by typing CTRL-S. The output won't be lost; you can continue it from the point of CTRL-S by typing CTRL-Q.

System Utilities

Data General system utilities are useful program development tools which you can access easily from CLI.

The utility programs supplied with your system are listed in Table 1-1. Additionally, you received the appropriate version of the system-generation program (SYSGEN). If you have DOS, you also received the diskette copy program (COPY); if you have RDOS, you received the fast-dump and fast-load programs (FDUMP and FLOAD) and the data channel line printer format control program (VFU).

Other RDOS utilities available include BATCH (which can execute jobs serially without intervention), and the sort/merge program (RDOSSORT).

Language compilers you may have on your system are: FORTRAN IV, FORTRAN 5, BASIC, ALGOL, COBOL, or DG/L.

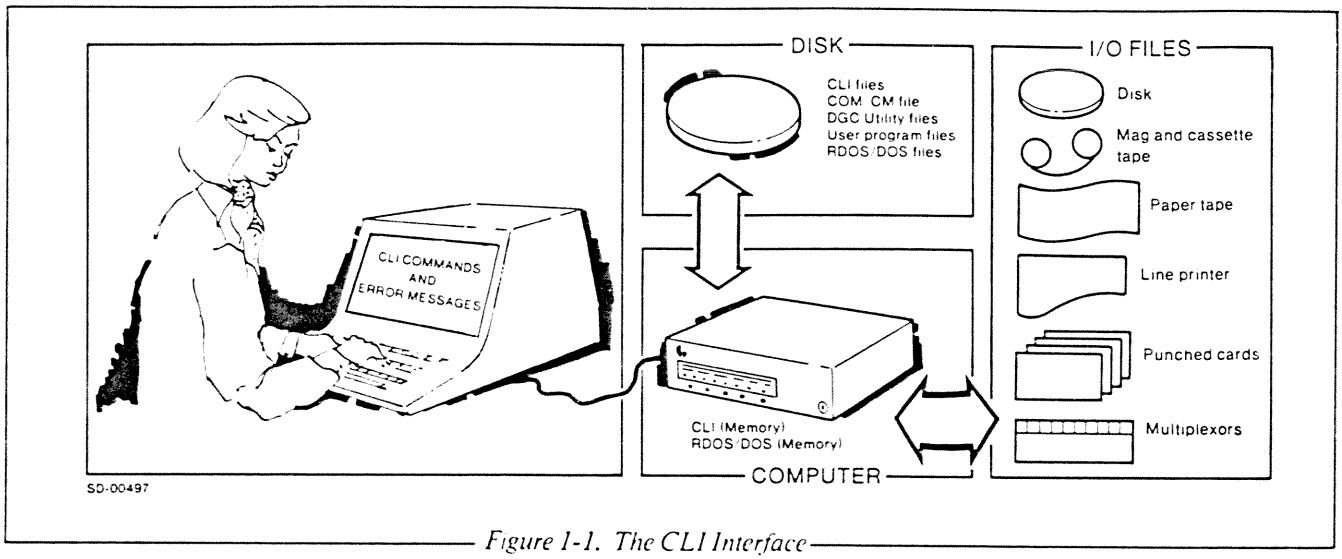


Figure 1-1. The CLI Interface

Table 1-1. System Utilities

Utility	Invoked by This Command	Purpose
Text Editors:		
Text Editor	EDIT	To write and edit a program or text file.
Multiuser Text Editor (RDOS only)	MEDIT	
Superedit	NSPEED (for NOVAs) SPEED (for ECLIPSEs)	
Assemblers:		
Extended Macro	ASM MAC	To assemble a source program (which might have been written with a Text Editor) into a binary file.
Library File Editor	LFE	To manipulate library files.
Loaders:		
Relocatable binary loader	RLDR	To process (load) binaries and create an executable program.
Overlay replacement loader	OVLDR (RDOS)	To replace overlays in an overlay file.
Symbolic Debuggers:		
Debug allowing interrupts Debug with interrupts disabled	DEB	To examine an executing program and correct errors.
Disk File Location Editor	OEDIT	To examine and modify the contents of disk file locations.
Symbolic Editor	SEDIT	To examine, analyze and modify disk files.
PATCH Utilities	ENPAT	To create patch files.
	PATCH	To apply patches in patch files.

End of Chapter

Chapter 2
Operating the CLI

This chapter describes CLI command format, punctuation, file media, disk files and directories, filename templates, console breaks, and error handling. As you read each section, you should use the console to try your own versions of the examples.

If you have not used the CLI before, sit down at the console with the manual *Learning to Use Your RDOS/DOS System*, and run through the console session in Chapter 2 (RDOS) or Chapter 3 (DOS).

Command Line Format

This is the normal format of a CLI command:

```
R
COMMAND/global-switches args/local-switches)
CLI response
R
```

The "R" is the CLI prompt character. The CLI always follows the prompt with a carriage return-line feed, which allows you to start typing commands on the next line. You can append the time of day to this prompt or remove the time, by typing the command ".)" (period, then RETURN).

Command

COMMAND is the name of the command or system utility.

Global Switches

You can often modify a command by appending one or more global switches to a command name. A global switch applies to the command itself, not to any argument in the command line; it immediately follows the command name, and consists of a slash (/) followed by a letter. Generally, the meaning of a global switch depends on the command that it modifies (the CLI ignores switches that are irrelevant to the command). Each individual command entry in Chapter 4 lists global switches. For example:

```
DELETE/V FILEA)
```

DELETE is the command-name, and /V is the switch that modifies the delete function: /V instructs the CLI to verify on the console that FILEA has been deleted.

```
LIST/E/S)
```

This command lists filenames; the global /E switch instructs CLI to list *every* statistic about files, and global /S *sorts* the listing alphabetically.

Comma(,) or Spaces

One or more spaces, or a single comma, separates the command and its arguments. Two or more consecutive spaces or commas are treated as a single delimiter.

Thus, the following two commands are equivalent:

```
DELETE FILEA,FILEB,FILEC
```

and

```
DELETE  FILEA FILEB,FILEC
```

Arguments and Local Switches

Some commands are used alone; others are followed by one or more arguments. A switch which follows an argument is a local switch; it modifies that argument only. A local switch consists of a slash (/) followed by either a number or letter. Depending on the command, you can insert both local and global switches in a command line. The CLI ignores switches which are irrelevant to the argument.

Local Letter Switches

A local letter switch modifies the action of the command on one argument. The meaning of each switch varies with the argument it modifies and the command line that contains the argument. For example:

```
LOAD MT0:0 OBSFILE/N 8-7-77/A)
```

The LOAD MT0:0 command alone would load all nonpermanent files in tape file 0 onto disk. The local /N switch following OBSFILE instructs the CLI not to load OBSFILE; the local /A switch instructs the CLI to load only those files created on or after August 7, 1977.

Local Numeric Switches

A local numeric switch specifies the number of times the preceding argument is to be repeated in the command line. This number must be in the range 0 through 9. For example, both

```
PRINT MYFILE/3 YOURFILE)
```

and

```
PRINT MYFILE MYFILE MYFILE YOURFILE)
```

will print MYFILE three times, and YOURFILE once, on the line printer.

When used alone, the /0 or /1 switch has no effect; the CLI will place the argument once in the command line. When more than one numeric switch follows an argument, they are executed sequentially, hence the effect is additive. If you wanted to print MYFILE 19 times, for example, you'd type:

```
PRINT MYFILE/9/9/1)
```

Carriage Return ())

A carriage return denotes the end of a command string. The command is executed after you press the RETURN key on your console.

CLI Response

Depending on your command, the console may display listing, error, or other information. If the CLI was unable to execute the command, it will try to tell you why in an error message. After it has executed the command, or returned an error message, it will display the R prompt.

Combining Commands

You can include two or more commands on the same command line by placing semicolons (;) between the commands. For example:

```
DELETE FILEA;CRAND FILEX FILEY)
```

CLI executes the two commands sequentially, after the carriage return is pressed.

Long Command Lines

An up arrow (SHIFT-N keys or SHIFT-6 keys on a DASHER) typed immediately before a carriage return allows you to span one or more commands over several input lines. Note that the CLI will not supply a delimiter between lines, so you must insert one manually - either before the uparrow or at the beginning of the new line. For example:

```
DELETE TESTA TEST01 ↑)
TEST02)
```

This executes as DELETE TESTA TEST01 TEST02).

CLI Punctuation Summary

The characters in Table 2-1 serve as delimiters or instructions to CLI. Other special characters, which allow you to manipulate disk files and command lines, are described in Chapter 3.

Table 2-1. CLI Punctuation

Symbol	Function	Example
) or ↓	The carriage return key ()) terminates an input command line and activates the CLI. The CTRL and L keys (↓) have the same effect. The SHIFT and L keys or backslash key (\) delete an entire line. The RUBOUT key or DEL (echoed as ← or _ on teletypewriters) erases the last character entered. On CRT displays the last character will disappear each time you press RUBOUT.	CREATE A B) CREATE A B↓ CCREAGE \ CC←READ←TE
, (space)	The comma or space is used to separate arguments. Extra spaces have no effect.	DELETE A,B) DELETE A B) DELETE A B)
/	The slash key before a character specifies a switch.	LIST/A)
;	The semicolon (;) delimits a command to CLI; you can then type another command on the line. No commands are executed until you enter a carriage return.	CREATE A;LIST)
↑	The SHIFT and N or SHIFT-6 keys (↑) followed immediately by RETURN can extend command lines over multiple lines.	RENAME A ALPHA,↑) B BETA ↑) E EPSILON)
.)	This command adds or removes the time of day to the prompt	.) 14:34:54 R
:	Insert the colon between filenames to access a specific tape file, or a filename in a different directory. The colon is described under Disk Directories, below.	DUMP MT0:2) PRINT SUBDIR:FILEA)
* -	If you enter an asterisk or dash with a filename, CLI will search for a group of related filenames. These are template characters described in Chapter 2.	LIST *A) DELETE A-)
(,,) < >	Parentheses and angle brackets are used when entering compact command lines to CLI. These are described in Chapter 3, Writing Compact Command Lines.	(PRINT,DELETE) LOG.CM) CRAND <A,B,C,D>.SR)
@ .MC	Commercial at signs indicate the contents of a file, rather than the filename itself. The .MC extension designates a macro file. For explanations of both @ and .MC, see Chapter 3, Indirect and Macro Commands.	ASM @FOO @) DAYSEND)
" "	Quotation marks delimit a literal text string. They are most useful with the MESSAGE command (Chapter 4).	MESSAGE "HELLO"
%	Percent signs enclose a CLI-defined variable. See Chapter 3, Variables.	RELEASE %MDIR%)
[,,]	Brackets and commas in the RLDR command line define a user overlay (see RLDR). Within these commands, brackets are used literally; they don't mean optional entries.	RLDR PROGA [A,B]

File Media

To access a file, use its filename. On disk, each filename is an alphanumeric name which the file creator assigns; on mag tape or cassette, the filename is the devicename and a file number. A tape "filename" can hold one or more disk files. The tape devicename is an abbreviation (MT or CT) followed by the unit number (e.g., MT0). The system assigns file numbers sequentially, as you write files to the tape. On a new tape, you write the first file to number 0, the second to number 1, and so on; if the tape is long enough (or your files are small enough) you can write up to number 99. To access a tape file, type the devicename, a colon, and the file number; e.g.,

```
LOAD MT0:3)
```

loads the contents of file 3 of the tape mounted on Magnetic Tape unit 0, onto disk; and

```
DUMP MT0:5)
```

dumps the contents of the current disk directory onto file 5 of the mag tape mounted on magnetic tape unit 0.

Nondisk and nontape filenames are simply devicenames - special device mnemonics which begin with the character S. Common devices are console input (filename STTI), console output (STTO), line printer (SLPT), paper tape reader and punch (SPTR and SPTP), and card reader (SCDR).

Table 2-2 gives the filenames of all RDOS devices.

Table 2-2. RDOS Devicenames

Devicename	Device	Devicename	Device
SCDR	Punched card reader; mark sense card reader.	MCAR	Multiprocessor communications adapter receiver line. You address each line as :n, where n is a number; e.g., MCAR:4.
CTn	First cassette controller drive (n is in the range 0-7). The first file on a cassette tape is 0, the second 1, and so on. To access any file, use a colon and the number; e.g., CT0:8.	MCAT	Multiprocessor communications adapter transmitter line. See MCAR for the line name.
DK0	First model 6001-6008 fixed-head disk, first controller.	MTn	First controller, 7- or 9-track magnetic tape. (Drive n is in the range 0-7.) The first file on a mag tape is 0, the second 1, and so on. To access any file, use a colon followed by the file number; e.g., MT1:6.
DSn	First model 6063/6064 fixed-head disk. For the first controller, n is 0, 1, 2, or 3.	$PLT	Incremental plotter.
DPn	First moving-head disk or diskette controller drive. For the first controller, n is 0, 1, 2, or 3.	$PTP	Paper tape punch.
DPn DPnF	Top-loading disk subsystem, which is two moving-head disks in one unit. For the first controller, n is 0, 1, 2, or 3. The top (removable) disk is DPn; the bottom (fixed) disk is DPnF. This controller can also support diskette units.	$PTR	Paper tape reader.
		QTY	Asynchronous line multiplexor (ALM) or data communications multiplexor (QTY). You address each line as :n, where n is a number from 0 to 63; e.g., QTY:13.
DZn	6060 Series Disk unit, first controller. The 6060 is single-density, the 6061 is double-density. n is 0, 1, 2, or 3.	$TTI	Teletypewriter or CRT display keyboard.
$DPI/$DPO	Dual processor input/output link. (Dual CPU/IPB systems only.)	$TTO	Teletypewriter printer or CRT display screen.
$LPT	Line printer (80- or 132-column).	$TTP	Teletypewriter punch.
		$TTR	Teletypewriter reader.

If you have a second disk controller on your system, address it as follows:

DK1	Second model 6001-6008 fixed-head disk.
DSn	Second model 6063/6064 fixed-head disk controller (n is 4, 5, 6 or 7).
DPn	Second moving-head disk controller (n is 4, 5, 6, or 7).
DPnF	Top-loading unit with two disks. For the second controller, n is 4, 5, 6, or 7. The top disk is DPn; the fixed disk is DPnF.
DZn	6060-6061 series disk. n is 4, 5, 6, or 7.

For a second nondisk device, simply append a 1 to the reserved filename; e.g.,

$LPT1	Second line printer (80- or 132-column).
CT1n	Second cassette controller (n is in the range 0-7).
MT1n	Second mag tape controller (n is in range 0-7) and so on.

Table 2-3 gives all DOS devicenames.

Table 2-3. DOS Devicenames

Devicename	Device
$CDR	Card reader
DPn	Diskette drive n. For the first microNOVA controller, n is 0 or 1; for the second microNOVA controller, n is 2 or 3; for the third and fourth controller, n is 4 or 5 and 6 or 7 respectively. For other NOVAs, on the first controller, n is 0, 1, 2, or 3. On the second controller, n is 4, 5, 6, or 7.
$LPT $LPT1	First and second line printers.
MTn	Magnetic tape unit n, nine-track tape. n is a number from 0 to 7.
$PLT	Incremental plotter.
$PTP	Paper tape punch.
$PTR	Paper tape reader.
QTY	Multiplexor. See QTY in Table 2-2.
$TTI	Console keyboard.
$TTO	Console screen or printer.
$TTI1	Second console keyboard.
$TTO1	Second console screen or printer.

Nondisk Devices

Before you can use magnetic tape or a cassette drive, you must initialize it with the INIT command; you can then access the files on it by devicename and file number. After you have finished with it, be sure to RELEASE it from the system. (Both the INIT and RELEASE commands are covered in Chapter 4.) For example:

```
INIT MT0)
LOAD/V MT0:3)
RELEASE MT0)
```

You must write files to mag tape or cassette in numeric order. If the tape is new, you should fully initialize it before using it:

```
INIT/F MTn
```

or

```
INIT/F CTn
```

This will write a double EOF mark on the tape. The double EOF tells the system where the logical end of the tape is, and defines the number of the next file to be written. INIT/F destroys all existing files on the tape.

The first file on a tape is number 0. When you have finished file 0, the system writes a double end-of-file mark on the tape; this defines the end of file 0. This process continues until you have written 99 files on the tape, or until the tape itself has run out. The system does not check the tape for existing files as it writes, and you should therefore keep careful track of the files and numbers on your tapes to prevent files from being overwritten. Assume, for example, that you have written six files on a tape which is mounted on unit MT0, and you carelessly issue a command like:

```
DUMP MT0:2)
```

The contents of current disk directory will overwrite the old file 2, destroying it and all following tape files.

To access all other devices on your system, use their devicenames. You need not initialize or release them, but you may need to operate them manually during execution of a CLI command. For example,

```
XFER $PTR MYFILE)
```

requests that the CLI copy the tape in the paper tape reader to a new disk file named MYFILE. The CLI replies:

```
LOAD $PTR, STRIKE ANY KEY.
```

You then load the paper tape reader and strike any key on your console. (The key is not echoed.)

Listing File

Many system utility programs offer an /L switch, which directs the utility to send a listing of its output to a file other than the console. Generally, you'll be using this switch to specify either a disk file or the line printer. For example, the command

```
MAC MYFILE OUTPUT/L)
```

assembles MYFILE and sends the assembly listing to disk file OUTPUT. You can then TYPE or PRINT OUTPUT at any time.

The command

```
MAC MYFILE $LPT/L)
```

assembles MYFILE and sends the listing directly to the line printer; the listing is not saved on disk.

The command

```
MAC/L MYFILE)
```

assembles MYFILE, creates a disk file named MYFILE.LS (if it doesn't exist), and appends the assembly listing to MYFILE.LS. As with the file specified by local /L, you can then TYPE or PRINT MYFILE.LS at any time.

Disk Files

A disk is the best place for your working files. Access to a disk is faster and far more versatile than other media; you can give disk files descriptive names; you can organize disk files logically within directories; and you can use any system utility on a disk file. (Although the utility programs will operate on a nondisk file, they work inefficiently, and require special handling. The assembler utilities, for example, make two passes over the input, and this requires two passes through the paper tape reader for a paper tape source file.)

Generally, we recommend using nondisk media only for backup, or to store inactive files which would otherwise consume valuable disk space.

A disk file can be a directory which contains other files; or it can be a data file or program file; or it can be a link entry which contains nothing, but points to another file.

Each disk file has a name (which its creator assigns) and one or more characteristics (which the system assigns). It can also have a two-character extension, and one or more protective attributes. Characteristics and attributes are covered in the LIST and CHATR command descriptions (Chapter 4); names and extensions are explained next.

Disk Filenames

A legal filename has between one and ten uppercase alphanumeric characters, including $. The name must be unique within its directory. The system will not allow you to create an illegal filename. For example, take the CRAND command, which creates a random file, and the LIST command, which lists filenames and file types:

```
R
CRAND FIRSTFILE)
R
CRAND FIRSTFILE)
FILE ALREADY EXISTS:FIRSTFILE
R
```

If you try to create an existing filename, the system will refuse.

```
CRAND #FILE)
ILLEGAL FILENAME: # FILE
R
```

Try an illegal filename, and the system declines.

If you try more than ten characters, the system will create the file:

```
CRAND PROLONGEDFILENAME)
R
LIST)
FIRSTFILE     0  D
PROLONGEDF    0  D
R
```

but will include only the first ten characters.

The system will not create a file with an existing name, an illegal name, or with a name larger than ten characters. Although it appeared to accept PROLONGEDFILENAME, it actually accepted only the first ten characters and ignored the rest.

Filename Extensions

Filename extensions provide you with a fast, versatile way to access and identify the contents of your disk files. (Nondisk filenames are devicenames, and do not have extensions.) You can create an extension when you create or rename a disk file. An extension is delimited by a period and consists of one or two alphanumeric characters (including $) following the filename. You can use any combination of these; for example:

```
TEST.
TEST.A
TEST.A1
```

The system would regard each of these as a separate file, and permit you to create all of them in the same

directory. If you enter more than two characters for an extension, the system will ignore the extra characters. Only the first two are significant. For example:

```
R
CRAND TEST TEST.A TEST.A1)
R
CRAND TEST.A1BB)
FILE ALREADY EXISTS: TEST.A1 BB
R
```

The system created the three files, but refused to create TEST.A1BB, which it saw as another TEST.A1.

Bear in mind when you choose an extension that it will become part of the filename, and in most cases the system cannot access the file without it. You can use the LIST command, in conjunction with filename templates (described later), to refresh your memory and speed up access to different filenames and/or extensions.

Generally, you can be quite free with extensions for files which will hold groups of data (e.g., BILLS.JA, BILLS.FB). For source programs, which will eventually become save files, processing will be easier if you follow system conventions. This means giving your source files the traditional extensions, or none. For assembly-language programs, the traditional extension is .SR, for FORTRAN programs, it's .FR, for ALGOL programs, it's .AL; for other languages, see your language reference manual.

If you follow this practice, you'll never need to type an extension when when processing your source file with an assembler, compiler, or loader, to make it into an executable program. The final program always receives the extension .SV, but you needn't type the .SV extension to execute the program. (However, no program will run if you change or remove its .SV extension.)

RDOS Utilities

The following paragraphs describe the action of some program-development utilities. For more on the compilers, see the individual compiler command name in Chapter 4.

ASM and MAC are assembler utilities; they expect a source file, written in assembly language, with the extension .SR. If the file you want to assemble has any other extension, you must type both name and extension. The assemblers produce binary files from your source files. ASM and MAC usually name the assembled binary *filename.RB*. The original source file (*filename* or *filename.SR)* remains intact after each command.

LFE is a library file editor; it allows you to create and edit library files, which are groups of binary files. A binary file represents an intermediate stage in program development; you produce a binary by assembling a source file with ASM or MAC or compiling it with FORT, etc. LFE always searches for *filename.RB* or *filename.LB*. If the file(s) you want to edit have any other extension, you must type both the filename and extension. The extension your edited output receives depends on the kind of editing you do. See LFE in Chapter 4 for more information.

RLDR produces an executable save file by combining relocatable binary modules. RLDR expects each filename to have the .RB extension or none. If the filename(s) you want to load has any other extension, you must type both name and extension. The save file is always named *filename.SV*, and overlays (if you specified them) are placed in *filename.OL*. The original filenames (.RB, or .LB) remain intact.

filename) instructs CLI to execute the file named *filename*. CLI searches for a macro file *(filename.MC)*; if it does not find one, it searches for the save file *(filename.SV)*.

Disk Directories

Within any system, each user needs disk space for files. Disk directories allow you to organize and assign file space flexibly, by user or category name. You already have some experience with these structures from *Learning to Use Your RDOS/DOS System.*

RDOS Partitions and Subdirectories

When a disk is introduced to RDOS, it has a given number of blocks available for storage (a disk block is 256 16-bit words). These blocks make up an area called the *primary partition*. According to everyone's needs, sections of the primary partition can be logically detached and given different filenames. These discrete sections are called *secondary partitions*. Within the primary partition (and secondary partitions, if any) you can place smaller directories called *subdirectories*. You can create files in or move files to any partition or subdirectory.

The file group in each partition or subdirectory is logically discrete, which means that files with identical names can exist in different directories. If you attempt to create, dump, load, move, or transfer *filename* into a directory where *filename* already exists, however, the system will reject your command and return the message FILE ALREADY EXISTS: *filename*.

Generally, the RDOS system utility files were placed in the master directory (primary partition) during system generation. If you want to use these from another directory, you must either link to them via the LINK command, or use directed access to them.

Figure 2-1 shows disk Dxx before and after partitioning and file creation; it also gives the CLI commands required to do the partitioning. Dxx can be any of the disks in Table 2-2; to determine its name, type MDIR).

Partitions and subdirectories are files which can be shared by several users and by foreground and background programs. All but the primary partition can be deleted. Any command you can execute on a partition will affect its subdirectories and files, and any command made upon a subdirectory will affect its files. All directory commands are summarized at the beginning of Chapter 4.

DOS Directories

In DOS, there are no secondary partitions. Each diskette is a primary partition, on which you can create logical subsets of file space called *directories*. DOS directories are the same as RDOS subdirectories. The files created in (or moved to) each directory are logically separate from those outside it. Figure 2-2 shows a diskette with sample directory structure, and the CLI commands which created the directories.

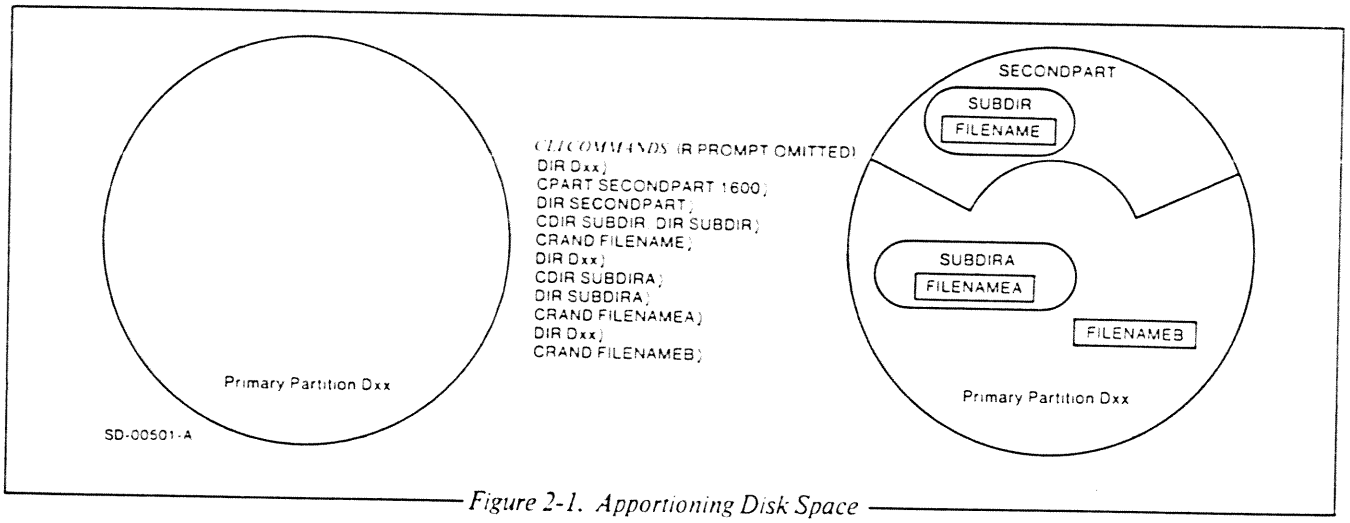


Figure 2-1. Apportioning Disk Space

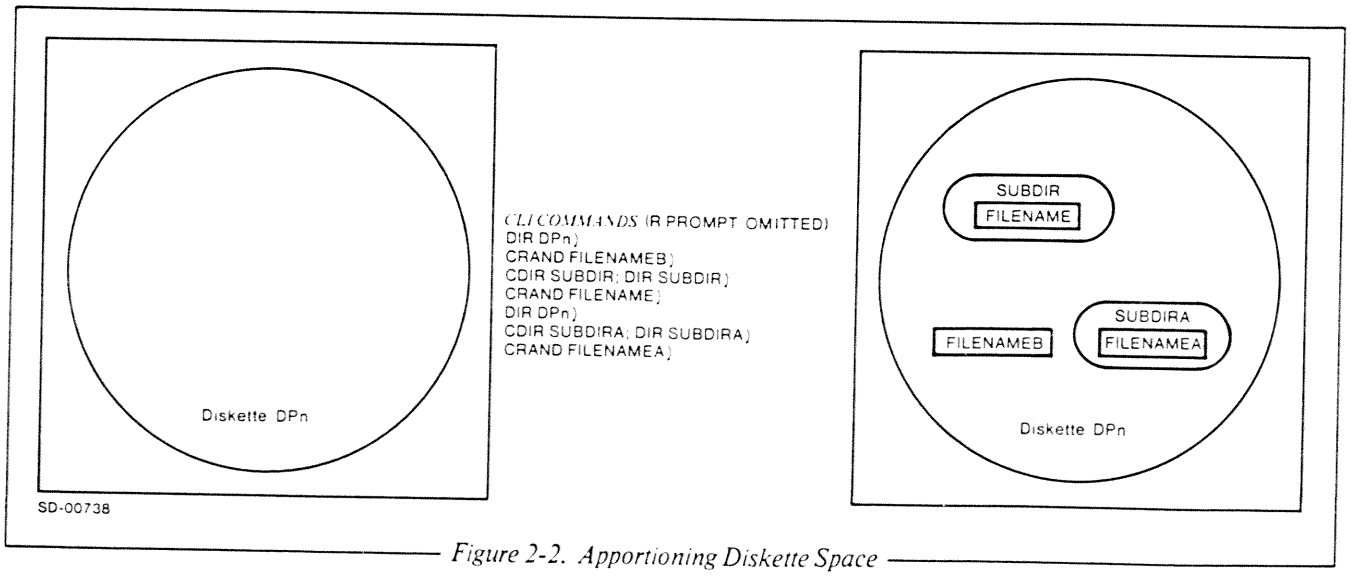


Figure 2-2. Apportioning Diskette Space

Directory Names

Tables 2-2 and 2-3 explain disk and diskette primary partition names. Other directory names must conform to system rules for filenames: up to 10 alphanumeric characters, including S. Whenever you create a secondary partition or subdirectory, the system assigned the extension .DR to it.

To prevent confusion, and logical mixups, the system does not permit two directories of the same name to be initialized at the same time. This is true even if the directories are on different partitions, on different disks, or operating in different grounds.

When a system has more than one directory with the same name, the first one initialized will become *the* version; no matter what directory specifiers you use to select the other version, the CLI will return to the first version - often without an error message. Before you can initialize the second version, you must release the first.

All this may perplex you or other users; anyone coming on the system may inadvertently find himself in the wrong version of a directory, and innocently destroy or move files. To prevent these problems, we recommended that you give every partition and subdirectory (or DOS directory) a unique name.

Directory and File Access

To help understand this section, assume that you are starting from a cold system, and that you want to type the contents of file FILENAME in Figure 2-1 (Figure 2-2 for DOS). FILENAME is a random file, created in a subordinate directory called SUBDIR. You begin by bootstrapping (bringing up) the system, and enter the date and time; you then receive the R prompt.

Before you can access FILENAME (or any file in your system), you must "turn on" the directory which contains it. This step is called initialization; once you have initialized a directory, the system remembers where it is, and can access it directly - even if it is on another disk. Bootstrapping automatically initializes the master directory. FILENAME is not in the master directory; it is in a subordinate directory, and you must initialize this directory before you can access it.

You can use one of two commands to "turn on" a directory: INIT or DIR. INIT simply initializes the directory, while DIR initializes it and makes it the current directory. The CLI is directly aware of files in the current directory, and can access them without directory specifiers. At any moment, many directories can be initialized, but only one can be the current directory. Your choice of INIT or DIR will depend on where you want the current directory to be. (If you forget the name of the current directory, type the command GDIR, and the CLI will return its name.)

To turn on FILENAME's directory, SUBDIR, you could type the following series of commands:

For RDOS:

```
DIR (or INIT) SECONDPART)
DIR (or INIT) SUBDIR)
```

For DOS:

```
DIR (or INIT) SUBDIR)
```

If you chose DIR, the current directory becomes SUBDIR; if you used INIT, the current directory would remain the master directory. In the RDOS example, you could have saved time by typing one command:

```
DIR (or INIT) SECONDPART:SUBDIR)
```

The directory specifier SECONDPART: directs the CLI to execute the command on SUBDIR. Any directory specifier points to the directory or file which follows it; it is an essential file access tool.

While SUBDIR is initialized, FILENAME will be accessible from anywhere in the system via the directory specifier:

```
SUBDIR:FILENAME
```

If the current directory is SUBDIR, you can omit the directory name, and simply specify:

```
FILENAME
```

Thus, if you had "DIRed to" SUBDIR, you could type:

```
TYPE FILENAME)
```

But if you had INITed SUBDIR, the current directory would remain the master, and you would need a directory specifier in the command:

```
TYPE SUBDIR:FILENAME)
```

Note that you can never access or execute a file outside the current directory without preceding its name with a directory specifier, unless you have linked to it.

After you have DIRed to or INITed a directory, it remains in the system until you RELEASE it. Now that SUBDIR has been turned on, you can execute most CLI commands on any file from any directory, by typing:

```
COMMAND SUBDIR:file-in-subdir
```

For example:

```
DIR DP7:OBSCUREDIR)          (Assume that your system has a DP7, with a directory called OBSCUREDIR)
R
TYPE SUBDIR:FILENAME)        (The system remembers the location of any initialized directory.)
(contents of FILENAME)
R
```

During any CLI session, you can DIR to or INIT each directory as you need to access files within it. If you reach the system limit of initialized directories, you'll receive an error message:

```
DIR LIMITPLUS1)
OUT OF DCBS: LIMITPLUS1
```

The *OUT OF DCBS* message means that you have reached the maximum number of directories, as determined during system generation. To initialize the directory which evoked the error message, you must RELEASE a previously-initialized directory.

Releasing a directory turns it off; releasing a partition or diskette also turns off any directories it contains. You can RELEASE a directory from within it. For this example, you can RELEASE all directories, and sign off the system, by typing:

RELEASE {Dxx / DPn})

The system would then verify the message and shut down:

```
MASTER DEVICE RELEASED
```

You cannot RELEASE a directory while any of its files are in use by another program (this can happen in RDOS only). If a foreground program is running, and you try to release a directory which it is using, CLI will display an error message. We describe foreground/background programming at the end of Chapter 3.

During this (or any) CLI session, no harm would have been done if you had made an error in any of these steps. The CLI would simply have returned an error message; for example, if you had tried to TYPE FILENAME from the wrong directory:

```
DIR %MDIR%; TYPE FILENAME)
FILE DOES NOT EXIST: FILENAME
R
```

(No FILENAME exists in the master directory. %MDIR% signifies the master directory name; we describe it in Chapter 3.)

Or if you had tried to initialize a previously initialized directory:

```
DIR SUBDIR)
R
INIT SUBDIR)
DEVICE ALREADY IN SYSTEM: SUBDIR
R
```

Or if you had tried to release a directory which was not initialized:

```
RELEASE SUBDIR)
R
RELEASE SUBDIR)
NO SUCH DIRECTORY: SUBDIR
R
```

To the system, a released directory does not exist.

Filename Templates

As the number of files in your system grows, and you assign extensions to help identify certain file categories, it is handy to have some way to access files by one or two characters in their names.

The CLI offers two template characters to help you access similar filenames. When you use one or both of these characters with a filename in a command, the CLI executes the command on all matching filenames in the current directory. Templates are very useful in conjunction with filename extensions.

A filename template may contain either or both of the following characters:

* *Asterisk* - represents any single character, except a period, in a filename or extension.

\- *Dash* - represents any string of characters, except a period, in a filename or extension.

For example, assume that the following filenames are in the current directory:

```
A
ATOM
A2SM
ADAMS
ADAMS.SR
ADAMS.RB
ADAMS.SV
```

Since the asterisk represents any single character except a period in a filename, the command

```
LIST A**M)
```

would list all four-letter filenames beginning with A and ending with M. These are ATOM and A2SM. A and ADAMS would not be listed because they are not four letters long. Also neither ADAMS.SR, ADAMS.RB, nor ADAMS.SV would be listed because each has a period and extension; the template A**MS.** would access them.

Since the dash represents any character string of *zero* or more characters in a filename or extension (except a period), the command

```
LIST A-)
```

would list all filenames shown without extensions. LIST -.*- would access all filenames with one- or two-character extensions. The template -M would access ATOM and A2SM, and template A-.- would access all names shown. The command LIST -.-) would list all nonpermanent files in the current directory.

Occasionally, as you work with the CLI, you will forget the names of pertinent files or even directories. Templates and the LIST command can help you with any of these situations. For example, consider the disk structures in Figures 2-1 or 2-2. If you were new to this disk, and would like to learn its directories, you'd type:

```
LIST -.DR)
SECONDPART.DR  819200 CTY
SUBDIR.DR      512 DY
R
```

For DOS (Figure 2-2) the figures would be:

```
SUBDIR.DR      512 DY
SUBDIRA.DR     512 DY
R
```

You could then DIR to each directory, and LIST its files. In this example, the system-assigned extension .DR was useful.

Assume that during a CLI session, you created many small files. Knowing that you would eventually want to delete most of them, you gave them the same name (TEST) and different extensions. At the end of the day you'd type:

```
R
LIST/A TEST.-)
TEST.3   40   D
TEST.1   60   D
TEST.1A  480  D
TEST.2   400  D
 and so on.
R
```

If you decided that you would like to save two versions of TEST, you would protect them with the permanent attribute and delete the others:

```
CHATR TEST.2 +P TEST.3A +P)
R
DELETE/V TEST.-)
DELETED TEST
DELETED TEST.1
DELETED TEST.1A
PERMANENT FILE: TEST.2
DELETED TEST.3
PERMANENT FILE: TEST.3A
DELETED TEST.4
.
R
```

The CLI has deleted all files which match the template, except those which you CHATRed permanent, and verified the deletions. You could check this with another LIST template:

```
LIST/A TEST.-)
TEST.2   400 PD
TEST.3A  460 PD
R
```

There are some restrictions on the use of templates:

- The filename you use in the template must be in the current directory. You cannot use templates with a directory specifier.
- You may use templates only in the following commands:

BUILD	DUMP	MOVE
LIST	UNLINK	
DELETE	LOAD	

If you try a template in any other command, the CLI will return an "illegal filename" error message.

Link Entries

Link entries save disk file space by allowing users in different directories to access a single commonly-used disk file; this is their most useful application. A link

entry is a filename in one directory which points to a file (called the resolution file) in another directory, or to another link entry.

With the link entry, you can access any file on any disk as if it were in your own directory (assuming that its attributes permit linking; for more on this, see the CHLAT command). You create a link entry with the LINK command, delete it with the UNLINK command, and change a file's link access attributes with the CHLAT command. See LINK for more detail.

System Console Breaks

You can enter three program breaks from the system console: CTRL-A, CTRL-C, and CTRL-F. CTRL-F terminates the foreground program; we discuss it in Chapter 3. CTRL-F doesn't pertain to DOS.

CTRL-A terminates most executing CLI commands and returns the "R" prompt. It also terminates execution of all utility programs (except editors) and all user programs, unless you have built a different interrupt routine into a program, as described in Chapter 3 of your system reference manual under "Keyboard Interrupts". After you type CTRL-A, the system displays the word *-INT* on the console; then, generally, it invokes the CLI.

For example:

```
A) DIR %MDIR%)
   R
   LIST)
   SYS.SV      SD  39424
   DSKED.SV    SD  12288
   RLDR.OL     C   22464
   CLI.SV      SD  10240
   CLI.OL      C   33008
   CTRL-A
   -INT
   R
```

CTRL-A aborts the listing and saves nothing.

```
B) ASM MYSOURCE)

   PROGRAM IS RELOCATABLE
     .TITL MYSOU

   CTRL-A
   INT
   R
```

CTRL-A aborts the assembly; file MYSOURCE remains intact.

CTRL-C also interrupts an executing CLI command or program, but saves the memory image in a file called BREAK.SV. BREAK.SV is placed in the current directory. (In an RDOS mapped system, a trap will also produce BREAK.SV.) You cannot execute BREAK.SV directly, but you can use it for assembly-language debugging. For more on this file, see .BREAK in Chapter 3 of your system reference manual. After you type CTRL-C, the system displays the word *BREAK* on the console; then, generally, it invokes the CLI.

In the following example, MYPROG is a user program with problems:

```
MYPROG)
.
.
(here, a delay occurs which means that MYPROG isn't
running properly)

CTRL-C

BREAK
R
```

The system places BREAK.SV in the current directory (deleting any existing version first); MYPROG.SV remains intact.

Error Handling

Generally, the CLI will abort any command (except DELETE) that threatens an existing file. If it cannot obey a command for any reason, it will try to explain the reason, and will usually display the argument which caused the problem; e.g.,

```
CRAND MYFILE; CRAND MYFILE)
FILE ALREADY EXISTS: MYFILE
```

Reason — Argument which caused the problem

NOTE: This is not necessarily true when you execute a program or utility via the CLI. See the explanation in the next section, "Using .ERTN."

In most cases, when you enter a command which is legal for some arguments and illegal for others, the CLI will execute all legal arguments before it returns the error message:

```
CRAND A A# B B&)
ILLEGAL FILENAME: A #
ILLEGAL FILENAME: B&
R
LIST *)                    (Were A and B created?)
A       0D
B       0D                 (Yes)
R
```

Most error messages result from forgetfulness - forgetting to initialize a directory, forgetting the location of a file, or forgetting the required format of a command. Although errors may be inconvenient, they are all harmless; the fact that the CLI could interrupt and respond to the error indicates its harmlessness. Generally, the message you receive will allow you to correct the error easily. Appendix A contains a complete list of CLI error messages and their causes.

Here are some instructive error examples. For foreground/background examples, see the end of Chapter 3.

```
RENAME ASM.SV ASSEMBLER.SV)
R
ASSEMBLER MYFILE)
FILE DOES NOT EXIST: ASM.SV
R
```

Some system utilities won't work if you RENAME them.

```
CHATR $LPT 0)
FILE ATTRIBUTE PROTECTED: $LPT
R
```

The attributes of certain files in your system were set and fixed at Data General, and cannot be changed.

```
LINK ASM.SV DP0:ASM.SV)
R
LINK ASM.SV OTHERDIR)
FILE ALREADY EXISTS: ASM.SV
R
```

A link entry is a file, and two files with the same name cannot occupy the same directory (you could create a link entry with a unique name to OTHERDIR).

```
DIR DP0)
R
DUMP DP1:FOO)
FILE SPACE EXHAUSTED: DP1:FOO
R
```

All material on DP0 won't fit into the unused file space on DP1.

```
DIR MYDIR)
R
EDIT MYFILE)
FILE DOES NOT EXIST: EDIT.SV
R
```

The Text Editor program doesn't exist in directory MYDIR. Generally, during system generation, all system utilities were placed on the directory which holds the operating system. You can use link entries to access them from any other directory.

Using .ERTN

In your assembly-language programs, you can instruct the CLI to interpret errors with the system call .ERTN. If a program encounters an error which prevents it from proceeding, it may take the error return through .ERTN. If so, the CLI will interpret the system error code that is always returned in AC2, and display an error message on the console.

The CLI also tries to describe the argument which caused the error, as in the CRAND example above. However, the argument which follows the colon will always be the name of the program which took the error return. (This is true because the CLI is not active when the program encounters the error, thus the CLI does not know what the specific problem was; all the CLI can do is interpret the code in AC2).

For example, assume the program MYPROG.SV needs to read a file named DATA1. If MYPROG can't find DATA1, and MYPROG takes the error return through .ERTN, this message will appear on the console:

FILE DOES NOT EXIST: MYPROG.SV

Interpretation of error code in AC2. — Program which took error return.

NOTE: Most of the Data General utility programs (e.g., MAC, SYSGEN, FORT) also use .ERTN. When they encounter a fatal error, they return to the CLI and the CLI interprets AC2 exactly as it would for a user program. This can be confusing at times. Use the LIST command to check for the existence of the files involved.

Traps and System Errors

If your system halts unexpectedly, it may have encountered an Exceptional Status condition. This is a critical hardware or software error. See the Exceptional Status appendix of your system reference manual for more on these errors.

In a mapped RDOS system, certain errors in a program will halt the program and invoke the CLI, displaying the message:

```
TRAP xxxxxx xxxxxx xxxxxx xxxxxx xxxxxx
BREAK
R
```

on the console. These user errors often involve endless loops or illegal access to system devices. Break file BREAK.SV is placed in the current directory. See the Exceptional Status appendix of the RDOS manual for more on traps.

End of Chapter

Chapter 3
Advanced Features

This chapter describes some of the more complex CLI and operating system features. It begins by showing you how to write compact command lines with parentheses and angle brackets, then describes CLI variables, indirect commands, and macros. (You already have some experience with CLI macros from *Learning to Use Your RDOS/DOS System.*)

It then explains the CLI-system interface, and covers dual programming (RDOS systems only).

None of the features described in this chapter are essential; you can use the CLI without them. They represent shortcuts and ways to exercise the power of the CLI and your system.

You'll be using the symbols in Table 3-1 to write compact command lines, use variables, and create indirect files and macros.

Table 3-1. Advanced CLI Symbols

Symbol	Feature	Example
(,,,,)	Place multiple commands or arguments in parentheses, and separate them by commas.	(PRINT,DELETE) LOG.CM)
< >	Use angle brackets for in-line expansion of multiple arguments.	PRINT DP1:<TEST<01 02 03>>
%	Enclose variables, whose values you want the CLI to supply, in % signs.	MESSAGE %TIME%)
@	Use commercial at signs to indicate an indirect command or file.	XFER/A STTI FOO) . . . ASM @FOO@)
.MC	Use the .MC extension to define a macro file.	XFER/A STTI MYCOMMAND.MC) . . . MYCOMMAND)

Writing Compact Command Lines

Angle brackets < > and parentheses ( ) allow you to enter complex commands to the CLI in an abbreviated form. The CLI will expand these command lines according to certain rules and then try to execute the expanded lines. By combining parentheses and angle brackets properly, you can execute complex command sequences with a minimum of typing at the console.

For example, the CLI expands the command line

```
MAC (<MY,YOUR>FILE) $LPT/L
```

to the commands:

```
MAC MYFILE $LPT/L;MAC YOURFILE $LPT/L
```

General Rules

You must match each opening parenthesis with a closing parenthesis, and each opening angle bracket with a closing bracket. Parentheses can appear within angle brackets - i.e., <()> - and vice versa - i.e., (< >) - but you cannot overlap the sets: <(>) is illegal.

The CLI evaluates the parts of a command string in the following order:

1. Text strings in quotes (MESSAGE command).
2. Indirect files or variables (described below).
3. Angle brackets.
4. Parentheses
5. Numeric switches.
6. Text arguments outside quotes (MESSAGE command).

Parentheses

Parentheses expand a command line into multiple command lines, creating one additional line for arguments within each set of parentheses. Arguments not in parentheses are executed as usual. Parentheses must be paired, and cannot be nested.

Other Rules

1. You must separate each argument within a set of parentheses from the next with a comma; e.g., (A,B,C).

2. The CLI extracts argument strings from parentheses left to right.

3. You can use more than one set of parentheses in a single line. When the CLI encounters more than one set of parentheses in a line, it extracts arguments in turn from each set of parentheses. For example, the line (LIST, PRINT) (FILEA,FILEB)) expands to:

   ```
   LIST FILEA
   PRINT FILEB
   ```

   After each argument in a set of parentheses has been used once, no argument in that string will be used again. For example, the command

   ```
   (LIST,PRINT,TYPE) (FILEA,FILEB))
   ```

   expands to:

   ```
   LIST FILEA
   PRINT FILEB
   TYPE
   ```

   and would evoke an error message, because TYPE requires an argument.

4. Nonparenthesized arguments are executed sequentially, as usual. For example, the line

   ```
   (LIST,PRINT) TEST (FILEA,FILEB))
   ```

   expands to:

   ```
   LIST TEST FILEA
   PRINT TEST FILEB
   ```

5. Processing ends when the CLI has tried to execute all arguments from the parenthesized set which contains the greatest number of arguments.

More Examples

Command	Expands to:
(PRINT,DELETE) LOG.CM)	PRINT LOG.CM DELETE LOG.CM
DUMP/V MT0:(0,1,2) (A,B,C))	DUMP/V MT0:0 A DUMP/V MT0:1 B DUMP/V MT0:2 C
MAC (A,B) (SLPT,OUT.LS)/L)	MAC A SLPT/L MAC B OUT.LS/L
MAC A(1,2) (SLPT,OUT.LS)/L)	MAC A1 SLPT/L MAC A2 OUT.LS/L
MAC (,A,B) OURS SLPT/L)	MAC OURS SLPT/L MAC A OURS SLPT/L MAC B OURS SLPT/L

For a more complex example, take:

```
MAC/U DP4:(TEST,A,B,C) OVLY  LISTDIR:(TEST,A,B,C)/L)
```

This assembles four files from DP4, and file OVLY from the current directory, with user symbols; it places the listings in directory LISTDIR. The expanded command is:

```
MAC/U DP4:TEST OVLY LISTDIR:TEST/L
MAC/U DP4:A OVLY LISTDIR:A/L
MAC/U DP4:B OVLY LISTDIR:B/L
MAC/U DP4:C OVLY LISTDIR:C/L
```

Angle Brackets

The arguments enclosed in angle brackets are expanded into a single command line. Arguments not in brackets (or parentheses) are executed as usual. You can nest angle brackets to any depth, but they must be paired.

Other Rules:

1. You can separate each argument within angle brackets from the next with one or more commas or spaces; e.g., <1,2 3>. Multiple commas or spaces indicate a null argument; e.g., <1,,2 3>. As an example, CRAND FILE<A,B,C>) expands to:

   CRAND FILEA FILEB FILEC

   CRAND FILE<1,2,>) expands to:

   CRAND FILE1 FILE2 FILE

2. You can use multiple sets of angle brackets in command lines. For example,

   CRAND TEST<1 2> TESTA<1 2>)
   creates
   TEST1, TEST2, TESTA1, and TESTA2.

3. When you nest angle brackets, the CLI expands the innermost level first, then proceeds toward the outermost level. Within each level, arguments are expanded left to right. Within nested brackets, a left or right bracket delimits each bracketed string. For example,

   PRINT DP4:<TEST.<01 02 03>>)

   expands to

   PRINT DP4:<TEST01 TEST02 TEST03>

   and then to four PRINT commands.

4. An argument string within a level is concatenated to each argument string at the next outer level. For example, see 3.

5. If the argument string delimiter is a comma or space, it is concatenated to the end of the new string. If the delimiter is a bracket, it is treated as a null. For example, see 3.

More Examples:

Command	Expands to:
PRINT <A B C>.SR)	PRINT A.SR B.SR C.SR
PRINT DP0:<A B C>.SR)	PRINT DP0:A.SR↑) DP0:B.SR DP0:C.SR
PRINT DP1:↑) <<A B C>.SR <D E>.LS>)	first step: PRINT DP1:<A.SR↑) B.SR C.SR D.LS↑) E.LS> second step: PRINT DP1:A.SR↑) DP1:B.SR↑) DP1:C.SR↑) DP1:D.LS↑) DP1:E.LS)

A handy way to learn how to construct compact command lines is to experiment using the MESSAGE command.

```
MESSAGE DP0:<A,B(1,2,3)C,D>.SR)
DP0:A.SR DP0:B1C.SR DP0:D.SR
DP0:A.SR DP0:B2C.SR DP0:D.SR
DP0:A.SR DP0:B3C.SR DP0:D.SR
```

See Chapter 4 for more information on MESSAGE.

Expansion Example

For a useful example, assume that you have created two programs, named RECAST.SV and REEXAMINE.SV. RECAST searches for a file, sorts it, and prints a sorted copy; REEXAMINE searches for certain features in a file, and prints them. Both RECAST and REEXAMINE will operate on an indefinite number of files and will thus accept many filename arguments in a command. Whenever you create a file that you plan to RECAST or REEXAMINE, you assign it to a category, and give it a category name; you use extensions to distinguish files within each category. Assume that after several days you have these files to be RECAST and REEXAMINED:

```
ACCTSDUE.V1,ACCTSDUE.V2
ACCTSDUE.V3,ACCTSDUE.V4
LAWSUITS.V1,LAWSUITS.V2
CREDIT.V1,CREDIT.V2
APOLOGIES.V1,APOLOGIES.V2,APOLOGIES.V3
```

You would then be ready to combine parentheses and angle brackets in a helpful and meaningful way:

```
(RECAST,REEXAMINE) ACCTSDUE.V<1 2 3 4>↑)
  LAWSUITS.V<1 2>↑)
  CREDIT.V<1 2> APOLOGIES.V<1 2>)
```

This command line would print RECAST and REEXAMINEd versions of the ten files - thus executing 20 operations sequentially from one command line, in this order:

```
RECAST ACCTSDUE.V1
RECAST ACCTSDUE.V2
  .
  .
  .
RECAST LAWSUITS.V1
  .
  .
  .
RECAST CREDIT.V1
  .
  .
  .
RECAST APOLOGIES.V2
REEXAMINE ACCTSDUE.V1
  .
  .
  .
REEXAMINE LAWSUITS.V1
  .
  .
  .
REEXAMINE CREDIT.V1
  .
  .
  .
REEXAMINE APOLOGIES.V2
```

You could carry this even further and use nested angle brackets and parentheses to select the categories you wanted RECAST and REEXAMINEd. Assume, for example, that you wanted to RECAST ACCTSDUE and CREDIT, and REEXAMINE every category but CREDIT. You'd type:

```
(RECAST,REEXAMINE) ACCTSDUE.V<1 2 3 4>↑)
  (CREDIT.V<1 2>, <LAWSUITS APOLOGIES>.↑)
  V<1 2>))
```

RECAST is the first command; the CLI would execute it on the four ACCTSDUE files, and on the first argument within following parenthesized sections. The first argument in the following section expands to CREDIT.V1 and CREDIT.V2.

REEXAMINE is the second command; CLI would execute it on the ACCTSDUE files, and on the second argument within the parenthesized expression. The second argument expands to all LAWSUITS and APOLOGIES files.

Thus the command sequence is:

RECAST ACCTSDUE and CREDIT files.

REEXAMINE ACCTSDUE, LAWSUITS and APOLOGIES files.

Variables

The CLI assumes a variable when it encounters a character string enclosed in percent signs (%). When the CLI encounters a legal variable, it replaces the variablename with the current value of the variable. If you had bootstrapped your current system on DP0, for example, the command line RELEASE %MDIR%) would evaluate to RELEASE DP0). Variables are most useful within indirect or macro commands and with the message command. The CLI recognizes the following variable names:

When the CLI encounters this variable name:	It inserts this value:
%DATE%	Today's date, in the form mm-dd-yy.
%GCIN%	The input console name (e.g., STTI)
%GCOUT%	The output console name (e.g., STTO)
%GDIR%	The current directory name (e.g., SUBDIR)
%LDIR%	The name of the previous current directory (e.g., DP4).
%MDIR%	The master directory name (e.g., DP0).
%FGND%	The character "F" if CLI is executing in the foreground; nothing if CLI is executing in the background.
%TIME%	The time of day, in the form hh:mm:ss.

Indirect and Macro Commands

The CLI offers the indirect and macro mechanisms to help you create and use your own compound commands. Both mechanisms involve building a file from several CLI commands, and thereafter

referencing the file by the single filename which you have assigned to the group. To use the macro feature, you must assign the extension .MC to the filename which will contain your commands.

Within your special command file, you can use templates, parentheses, angle brackets, and variables.

Assume that you normally want to do the following at the end of each CLI session:

- delete listing files
- list your nonpermanent files
- determine how much disk space you have left
- close, print, and delete the log file
- release the master device.

The commands required to do all this would be:

```
DELETE/V -.LS; LIST/E; DISK; ENDLOG;↑)
(PRINT,DELETE) LOG.CM; RELEASE %MDIR%)
```

You could save time and effort by writing these commands into a single file, which you might call DAYSEND. To build this file, you would transfer your own console input into file DAYSEND with the XFER command (you could also use a text editor utility):

```
XFER/A $TTI DAYSEND)              (/A specifies ASCII
                                   transfer.)
DELETE/V -.LS; LIST/E/A; DISK;↑)  (These are the
  ENDLOG; (PRINT,DELETE)↑)         commands which
  LOG.CM;                          comprise file
RELEASE %MDIR%)                    DAYSEND.)
CTRL-Z                            (CTRL and Z
                                   indicate a $TTI
                                   end-of-file, and
                                   return control to
                                   CLI.)
R
```

You could then execute the entire command group by typing:

```
@DAYSEND@)
```

The commercial at signs instruct the CLI to access the contents of the filename, instead of the filename itself. Note how this mechanism applies in this example:

```
BUILD DUALFILE FILEA FILEB)
```

This command builds an file named DUALFILE, consisting of filenames FILEA and FILEB. You could verify the build with the command:

```
TYPE DUALFILE)
FILEA,FILEB↑
R
```

But if you enclosed DUALFILE in @ signs, the contents of FILEA and FILEB would be typed:

```
TYPE @DUALFILE@)
.
(contents of FILEA)
.
(contents of FILEB)
R
```

A different way of creating your own command file is to make the command filename recognizable as a macro to the CLI. You can do this by appending the extension .MC to the filename when you build it. For the indirect file DAYSEND, above, the file-creating sequence would be:

```
XFER/A $TTI DAYSEND.MC)
DELETE -.LS; LIST; DISK; etc.

CTRL-Z
R
.
```

The command DAYSEND) would then execute the group.

When you enter a character string within @ signs, the CLI scans the current directory for the string. When you enter a string without @ signs, and the CLI cannot recognize the string as any of its commands, it searches for *string.MC*. If it can't find *string.MC*, it searches for *string.SV*. (If you have both a macro and save version of string in the current directory, you can execute the save version by typing string.SV) .)

A macro differs from an indirect command in two ways: 1) it requires the .MC extension, and 2) it must be self-contained. You cannot execute any CLI command on a macro because a macro is by definition a command, and must be the first word in a command line. You can, however, use an *indirect* filename as an argument to a CLI command. For a useful example of this distinction, assume that you have three source files PART1, PART2, and PART3. You use XFER to build the three files into a file called TEST:

```
XFER/A $TTI TEST)
(PART1,PART2,PART3))
CTRL-Z
R
```

The command ASM @TEST@ will assemble the three files, as if you had issued these commands:

```
ASM PART1;ASM PART2;ASM PART3)
```

The macro version of TEST would not work in this example; the command line ASM TEST.MC) would instruct the system to assemble TEST.MC.

The contents of either an indirect or macro file may in turn point to other files. For example, assume that

file A (or A.MC) contains L @B @
file B contains I @C @
file C contains ST

Then the command @A @) (or A) for the macro) would be equivalent to LIST).

To build file A (or A.MC), you would need to use indirects:

```
XFER/A $TTI A) (A.MC for the macro version)
L @B @)(You could also enter the .MC extension for
        indirect file B - e.g. L @B.MC @ )

CTRL-Z
R
XFER/A $TTI B)
I @C @)
CTRL-Z
R
XFER/A $TTI C)
ST)
CTRL-Z
R
```

When you typed @A @) (or A) for the macro version), the CLI would reference the contents of A - which is L - and the contents of B. B contains I and the contents of C, which is ST; hence the LIST command. When you use an indirect to call another indirect in this way, remember that each carriage return in an indirect file will act as a command delimiter when the file is executed. In the first example below, a RETURN in file A produces a meaningless command; in the second example, file A is built correctly.

1. XFER/A $TTI A)
   LIS) (This RETURN delimits the command string.)
   CTRL-Z
   R
   XFER/A $TTI B)
   T) (The RETURN doesn't matter here, because this is the last character in the last file.)
   CTRL-Z
   R
   (Try to execute the indirect files.)

   @A @ @B @)
   FILE DOES NOT EXIST:LIS.SV
   FILE DOES NOT EXIST: T.SV
   R

   (The command string was LIS -delimiter- T.)

2. XFER/A $TTI A)

   LISR)
   ▲ (CTRL-Z was pressed here.)
   XFER/A $TTI B)
   T)
   CTRL-Z
   R

   (Try again to execute the indirects.)

   @A @ @B @)
   MYFILE 128 D
   ALPHA.SV 400 D
   .
   .
   R

Without a delimiter after "LIS" the indirects produce "LIST". The CLI will accept literally every character you build into an indirect or macro file.

CLI - System Interface

The CLI is a system utility of the operating system, and communicates with it through standard system calls and reserved system calls. Whenever you type in a command that will execute a program (your own or a system utility) the CLI creates a file called COM.CM to communicate with the program. The CLI then takes your command line, formats it, and places it in COM.CM. The command remains in COM.CM until you type another program-executing command; then the new command replaces the old in COM.CM. For example, if you type the command:

```
MAC/N MYFILE)
```

The CLI formats the command, including the switch, and places it in COM.CM. Then, through system calls, the CLI brings MAC (the macroassembler program) into execution; MAC searches COM.CM for a filename, finds MYFILE, and assembles it. This activity is invisible to you at the console, but you must understand it if you want to pass arguments to your own programs from the CLI, as described in Appendix D.

Dual Programming (RDOS Only)

RDOS permits you to divide memory into two portions - called a foreground and a background. Within each ground, you can execute a separate program which runs autonomously in its own ground of memory.

The way you handle dual programming will depend on whether or not your computer has a hardware map. If it is mapped, then it runs under mapped RDOS. The mapped system provides, among other things, hardware separation between the foreground and background sections of memory, and the programs that run in them. Mapped systems also provide the GMEM command to check the memory allotment of each ground, and the SMEM command to change this allotment.

In an unmapped system, you must place memory address information in programs which you want to execute in the foreground.

Executing Background and Foreground Programs

When RDOS is bootstrapped (started up), no foreground exists; all memory is allocated to the background and the CLI is running in the background. To execute a program, you type programname); the program is then read in from disk and executed in background memory.

In a mapped system, you create a foreground with SMEM; you can then execute a program in the foreground by typing the command

EXFG *programname*).

This brings in the program and executes it in foreground memory. In mapped systems, you can EXFG any program if you have allotted enough memory with SMEM; there is no functional difference between foreground and background programs. You can EXFG the CLI itself, if you have a foreground console to communicate with it. The example below shows two CLIs running.

(In an unmapped system, you can also use EXFG to execute a program in the foreground; but you must have configured the program with foreground address information beforehand. The program itself, not the SMEM command, creates the foreground.)

When you issue the EXFG command, there must be enough memory allocated to the foreground for the foreground program to execute. If there is not enough memory, you'll receive the error message:

INSUFFICIENT MEMORY TO EXECUTE PROGRAM

You can remedy this in a mapped system by allocating more memory to the foreground with SMEM; or you can decide to run a smaller program in the foreground. In an unmapped system, you must change the boundary information in the foreground program itself; do this with the RLDR command.

In either system, after the foreground program is running, the CLI prompt will return to the background console; you can then resume normal background operations.

Normally, when you run two grounds, the system console communicates with the background program and another console with the foreground program. Foreground and background programs can communicate via system calls; each can initialize and use directories and devices; each can open and read files. Each program can access the same directory simultaneously.

When a directory has been initialized by any ground, either ground can use it. If the other ground *does* initialize it, there will be no error message; the directory will simply have been initialized by both grounds. When a ground releases a directory which the other ground has also initialized, this message appears on the console:

DIRECTORY SHARED: directoryname

This is simply an informational message; it means that the directory has been released by one ground, but that the other is still using it.

If you try to release the master directory while the other ground is running, you'll get an error message. For example, assume that you're running two grounds and your master directory is DZ0. From the system console, you type:

RELEASE DZ0)
FOREGROUND ALREADY RUNNING: DZ0
R

Before you can release the master directory, you must terminate the foreground program.

Terminating the Foreground Program

To terminate the foreground, type CTRL-F from the background. CTRL-F isn't recognized from the foreground console. CTRL-F releases all devices and directories initialized by the foreground, restores total control to the background, and displays the message

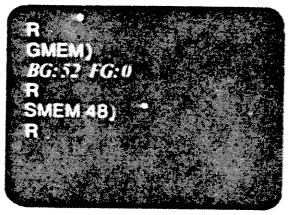
FG TERM

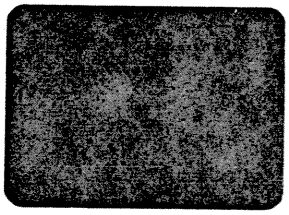
on the background console. It has no effect upon the background program in a mapped system. (In an unmapped system, it releases all foreground memory to the background.) If the foreground program has files open when you type CTRL-F, the updates won't necessarily be written to disk; thus if the updates are important, make sure the foreground closes all its open files before you terminate it.

You can also terminate any foreground program (except a text editor) by typing CTRL-A or CTRL-C from the *foreground* console. These work the same way as CTRL-F from the background, except that CTRL-C creates a breakfile called FBREAK.SV to save the foreground memory image.

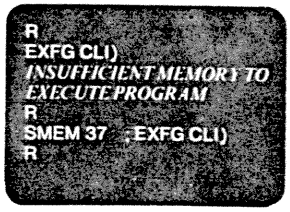
Foreground-Background Example

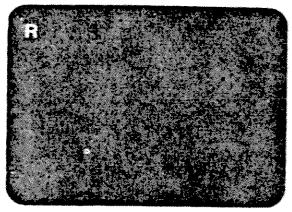
This example shows a mapped RDOS system with 128K bytes of memory. Mapped systems allot memory in blocks of 2,048 bytes, thus this system has 64 blocks of memory. In this system, RDOS requires 12 blocks, which means that the GMEM command returns a total of 52 blocks. In this system is a directory called MYDIR, which contains a file called MYFILE. The figures show dialog on the background console (left), then dialog on the foreground console (right).





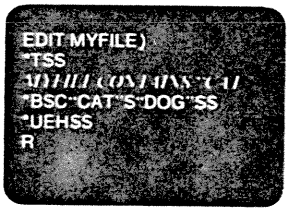
Check foreground/background figures; allot 48K blocks to the background.

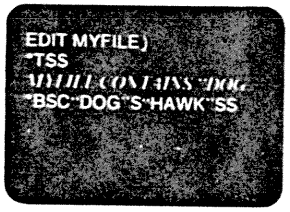




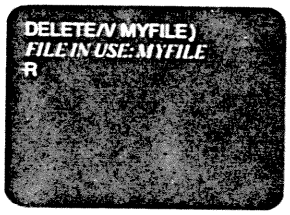
Execute the CLI in the foreground; receive an error message; allocate more memory and try again.

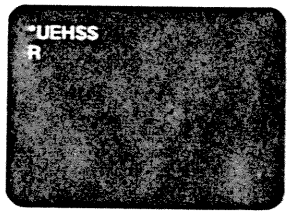
Initialize MYDIR in the background, then the foreground.



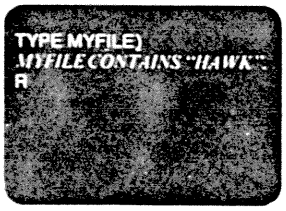


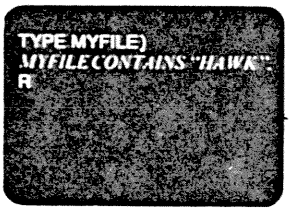
Edit MYFILE in the background with the text editor; change "CAT" to "DOG", then close MYFILE so that you can edit it in the foreground (you can't edit it from both grounds simultaneously). Change "DOG" to "HAWK" from the foreground.



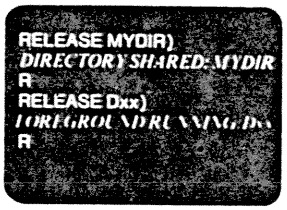


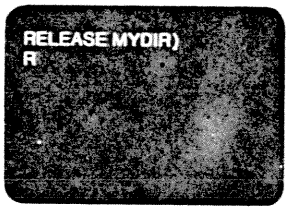
Try to delete file which is still open in the foreground. You can't delete it. Now, close file in foreground and return to foreground CLI.



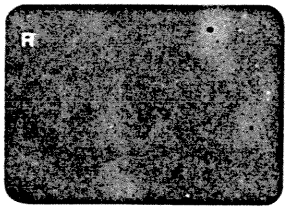


The foreground updates the file last, hence the foreground changes remain in MYFILE.





Release MYDIR from the background; the message informs you that the foreground has also initialized it. Try to release the master directory (Dxx); you cannot because the foreground program requires it. From the foreground console, release MYDIR; no other ground is using MYDIR, thus no message returns.



The foreground has released its directories, and closed its files, so terminate the foreground program (here, a CLI) with CTRL-F. Now, you can release the master directory.

End of Chapter

Chapter 4
CLI Commands

This chapter defines and describes each CLI command in two ways: by category and in alphabetical order. Table 4-1 presents commands by category; the remainder of the chapter describes them alphabetically. Unless a command is noted RDOS or DOS, it works in both systems.

Within the alphabetical section, each command entry begins with the command name, a capsule description, and a command format. To interpret the format properly, you must understand the documentation conventions as described in the Preface. Except where noted in the text, the syntax rules explained in Chapter 2 apply to each command format: you can substitute multiple spaces or a comma for a space, and you can use parentheses or angle brackets whenever you think they will save time. Italic brackets enclose optional entries, *except* in the RLDR command.

Generally, when a CLI command takes a filename argument, that argument can include a directory specifier (e.g., PRINT SUBDIR:FILENAME)). A few commands require the *filename* to be in the current directory; these are noted.

The template characters (- and *) can be used only for the following commands: BUILD, DELETE, DUMP, LIST, LOAD, MOVE, and UNLINK. If you use them elsewhere, the CLI will return an ILLEGAL FILENAME error message.

In some cases, we have used the up arrow (↑) line-continuation convention to prevent the format from overrunning the column edge. You can ignore this break and proceed to type as much of the command line as will fit on a line of your console. However, if you type more than 132 characters on a line, you'll receive the message

```
LINE TOO LONG
```

and the entire line will be aborted.

Table 4-1 divides the CLI commands into four categories: File Management, Directory, System Control, and System Utilities. Each entry includes the command name, a general format for using it, and a brief description of its action.

All the commands described under "System Utilities" invoke system utilities. These are programs which the CLI executes; they are not part of the CLI, and the CLI is not active when they run. When they terminate (either normally or after a fatal error), they return to the CLI.

Table 4-1. CLI Commands by Category

File Management Commands	Description
APPEND groupfilename filename...	Combine two or more files.
BPUNCH filename...	Punch a binary file.
BUILD outputfilename filename ...	Build a file from filenames.
CCONT filename blockct	Create a contiguous file.
CHATR filename *[+] [-]* attribs	Change a file's attributes.
CHLAT filename *[+] [-]* attribs	Change a file's link access attributes.
CLEAR *[filename]*	Set file use count to zero.
CRAND filename	Create a random file.
CREATE filename	Create a sequential file (in DOS, a random file).
DUMP dumpfilename *[filename...]*	Dump a file in CLI DUMP format.
ENDLOG *[password]*	Close the LOG file.
FILCOM filename1 filename2	Compare the contents of two files.
FPRINT filename	Print a file in octal, or other specified format.
LINK linkname resfilename	Create a link entry to a resfilename.
LIST filename	List the statistics of a file.
LOAD dumpfilename *[filename...]*	Load DUMPed files.
LOG *[password]*	Start recording in the log file.
MKABS savefilename binaryname	Make an absolute file from a save file.
MKSAVE binaryname savefilename	Make a save file from an absolute file.
MOVE directory *[filename...]*	Copy a file to any directory.
PRINT filename...	Print a file on the line printer.
PUNCH filename ...	Punch an ASCII file.
RENAME oldname newname	Rename a file.
REV filename	Display the revision level of program filename.
SAVE filename	Rename a breakfile.
TYPE filename	Type a file on the console.
XFER sourcefile destinationfile	Copy the contents of one file to another file.

Table 4-1. CLI Commands by Category (continued)

Directory Commands	Description
CDIR directoryname	Create an RDOS subdirectory or DOS directory.
COPY diskette1 diskette2	Copy diskette1 to diskette2 (DOS).
CPART partname blockct	Create a secondary partition (RDOS).
DIR directoryname	Change the current directory.
DISK	Display the number of blocks used and remaining on the current partition or DOS diskette.
DUMP outputfilename	Copy the contents of the current directory to outputfilename.
EQUIV newname oldname	Temporarily rename a disk or mag tape specifier (RDOS).
FDUMP *[MTn]*	Fast dump the current directory to mag tape (RDOS).
FLOAD *[MTn]*	Fast load a fast dumped (FDUMPed) file into the current directory (RDOS).
GDIR	Display the current directory name.
INIT directory or tapedrive	Initialize a directory or tape drive.
LDIR	Display the last current directory name.
LOAD dumpfilename	Reload DUMPed files.
LIST *[filename]*	List file information
MDIR	Display the master directory name.
MOVE directory *[filename...]*	Copy files to any directory.
RELEASE directory or tapedrive	Release a directory or tape drive.

System Control Commands	Description
filename *[.SV]*	Execute filename.
BOOT disk or system	Bootstrap a system from disk.
CHAIN filename	Overwrite the CLI with an executable program.
CLEAR *[filename]*	Set file or device use count to zero.
DISK	Display the number of disk blocks used and remaining.
ENDLOG *[password]*	Stop recording in the log file.

Table 4-1. CLI Commands by Category (continued)

System Control Commands	Description
EXFG program	Execute program in the foreground (RDOS).
GMEM	Display background/foreground memory areas (mapped RDOS).
GSYS	Display the current system name.
GTOD	Display the current system time.
FGND	Describe the foreground program status.
INIT directory or tapedrive	Initialize a disk directory or tape drive.
LOG *[password]*	Start recording in the log file.
MCABOOT (see command)	Transmit a system over an MCA line (RDOS).
MESSAGE text	Display a text message.
POP	Return to the program on the next higher level.
RELEASE directory or tapedrive	Release a directory or tape drive.
SDAY mm dd yy	Set the system calendar.
SMEM background	Set the background/foreground memory areas (mapped RDOS).
SPDIS devicename ...	Disable spooling to devicename. (RDOS)
SPEBL devicename ...	Enable spooling to devicename. (RDOS).
SPKILL devicename ...	Delete data spooled to devicename (RDOS)
STOD *[hh] [mm] [ss]*	Set the system clock.
TPRINT	Print the tuning file (RDOS).
TUOFF	Stop recording in the tuning file (RDOS).
TUON	Start recording in the tuning file (RDOS).

System Utilities Commands	Description
ALGOL filename...	Compile an ALGOL source file (RDOS).
ASM filename ...	Assemble a source file, producing an .RB file.
BASIC	Invoke the BASIC interpreter.

Table 4-1. CLI Commands by Category (concluded)

System Utilities Commands	Description
BATCH *[jobfile...]*	Invoke the Batch monitor, to execute Batch job streams (RDOS).
CLG filename...	Compile, load, and execute a FORTRAN IV source file.
DEB savefilename	Debug a program.
EDIT *[filename]*	Invoke the Text Editor.
ENPAT *[filename]*	Insert patch(es) in filename; also see PATCH.
FDUMP *[MTn]*	Fast dump the current directory to magnetic tape (RDOS).
FLOAD *[MTn]*	Fast load a fast-dumped (FDUMPed) file into the current directory (RDOS).
FORT filename ...	Compile a FORTRAN IV source program.
FORTRAN filename...	Compile a FORTRAN 5 source program (RDOS).
LFE (see command)	Create or edit RB Library Files.
MAC filename...	Assemble a source file into a relocatable binary (RB) file with the Macroassembler.
MEDIT terminals *[ticks]*	Invoke the Multiuser Text Editor (RDOS).
NSPEED *[filename]*	Edit text with the NOVA Supereditor.
OEDIT filename	Edit disk file locations with the Octal Editor.
OVLDR (see command)	Create an overlay replacement file.
PATCH (see command)	Install patch(es) created by ENPAT.
RDOSSORT (see command)	Sort a file or merge files with the Sort/Merge program (RDOS).
REPLACE savefilename	Replace overlays in an overlay file (RDOS).
RLDR (see command)	Process relocatable binary files to form an executable program.
SEDIT filename	Edit disk file locations with the Symbolic Editor.
SPEED *[filename]*	Edit text with the ECLIPSE Supereditor (RDOS).
SYSGEN (see command)	Generate a new operating system.
VFU filename	Create or load a VFU file for a data channel line printer (RDOS).

filename

Execute the program or macro named filename

Format:

filename *[.SV] [argument$_1$ ... argument$_n$]*

This command directs the CLI to find and execute the file named filename. The CLI searches first for a macro file (filename.MC), then for the save file (filename.SV). If it cannot find either, it returns the error message:

FILE DOES NOT EXIST: filename.SV

If you have both an .SV and .MC version of a file in the same directory, you can force the CLI to search for the save file by including the .SV extension.

The CLI will ignore arguments and switches if the filename has the .MC extension, and is therefore a macro file. If filename is a save file, it can access global switches, arguments, and local switches in CLI file COM.CM, as described in Appendix D.

Switches:

User-definable

Examples:

DP1:MYFILE)
Execute MYFILE in directory DP1. MYFILE could be a macro file (extension .MC), or a save file with the .SV extension.

MYFILE.SV)
Execute MYFILE in the current directory.

DP0:EDIT)
Execute the text editor program in DP0.

ALGOL

Compile an ALGOL source file (RDOS)

Format:

ALGOL filename ...

Compile and assemble a program written in ALGOL. As output, you can specify an assembled binary file, an intermediate source file, a listing file, or combinations of all three. The command will not work if its name, ALGOL, is changed.

On input the CLI searches for filename.AL; if it does not find this, it searches for filename.

If you omit switches from the command line, ALGOL produces an intermediate source file, filename.SR (compiler output), and a relocatable binary file filename.RB (assembler output). After a successful assembly, the intermediate source file is deleted; no listing is produced.

For more detail, see the *Extended ALGOL User's Manual.*

Global Switches:

/A	Do not assemble the compiled file.
/B	Brief listing (ALGOL input to the compiler only).
/E	Suppress console error messages from the compiler. (Assembler error messages are not suppressed.)
/L	Produce listing file (filename.LS).
/N	Check with assembler, but do not produce a binary file (useful for finding errors).
/S	Save the intermediate source file, under filename.SR.
/U	Append user symbols to binary output file.

Local Switches:

name/B	Name the binary output file name (overrides global /N).
name/E	send error listing to file name.
name/L	Send listing output to file name (overrides global /L).
name/S	Send the intermediate source file to file name.

Examples:

ALGOL MAIN)
Compile and assemble file MAIN, producing a binary file, without a listing.

ALGOL/A/L RAY)
Compile (but do not assemble) file RAY.AL or RAY; write the intermediate source file to disk file RAY.SR. ALGOL source listing goes to disk file RAY.LS.

ALGOL/E/B SUBR $LPT/L)
Produce binary file SUBR with a brief ALGOL source listing to the line printer; suppress compiler error messages.

APPEND

Copy one or more files under a new name

Format:

APPEND groupfilename filename$_1$ *[...filename$_n$]*

Create groupfilename and copy the contents of filenames into it. The old files are not changed. You can use directory specifiers for groupfilename or any filename.

Switches:

None.

Examples:

APPEND TAPEFILES MT0:1 MT0:2)

Create TAPEFILES, and copy the contents of tape files 1 and 2 (from drive MT0) into it.

APPEND PTRFILES $PTR/3)
LOAD SPTR, STRIKE ANY KEY

Create disk file $PTRFILES, and prompt for three tapes to be loaded into the paper tape reader; copy tapes to $PTRFILES.

ASM

Assemble source files to produce a relocatable binary (RB) file

Format:

ASM filename$_1$ *[...filename$_n$]*

Assemble one or more source files with the Extended Assembler. This produces a relocatable binary file (RB), which you then process with the RLDR command.

If you omit switches, ASM produces a relocatable binary file named filename.RB$_1$ and no listing. For more detail, consult the *Extended Assembler User's Manual.*

Global Switches:

/E	Suppress error messages to the console (unless there is no listing file).
/L	Create disk file filename.LS$_1$ and send listing to it; or append listing if file already exists.
/N	Do not build a relocatable binary file (useful for finding assembly errors).
/S	Skip pass two, and store the assembler's symbol table in file BREAK.SV. The console displays BREAK. You can then rename BREAK.SV (with the SAVE command) to store this symbol table which contains your own semipermanent symbols.
/T	Do not include symbols in the cross-reference listing (used with global /X).
/U	Include user symbols in the relocatable binary file.
/X	Produce symbol/page cross-reference listing. Assembler file XREF.SV must be available on disk.

Local Switches:

name/B	Give relocatable binary this name instead of filename.RB$_1$ (overrides global /N).
name/E	Send error messages to file name.
name/L	Send listing to file name (overrides global /L).
name/N	Do not include assembly listing of file name (used with global or local /L).
name/S	Skip file name on pass two of assembly. Use this switch only if file name does not contain any storage words.

Extensions:

The CLI searches for filename.SR; if it doesn't find that, it searches for filename. It names the relocatable binary file filename.RB$_1$, and filename.LS$_1$ (global /L), unless the command included the /B, /L, or /S local switch.

Error Codes:

If a line of source code contains an error, the assembler places a letter at the left margin of the offending line in the listing. It can insert no more than three codes per line.

Code	Meaning
A	Addressing error.
B	Bad character.
C	Colon error.
D	Radix error.
E	Equivalence error.
F	Format error.
G	Global symbol error.
I	Parity error (input).
K	Conditional assembly error.
L	Location counter error.
M	Multiply-defined symbol error.
N	Number error.
O	Field overflow error.
P	Phase error.
Q	Questionable line error.
R	Relocation error.
S	Symbol table overflow error.
T	Symbol table pseudo-op error.
U	Undefined symbol error.
X	Text error.
Z	Expression has illegal operand.

Examples:

ASM/N FILE1)

Assemble source file FILE1, and send error messages to the console; do not produce a relocatable binary or listing file. This checks for errors, which you can correct before an RB and listing file is produced.

ASM MYFILE SPTP/B SLPT/L)

Assemble MYFILE from disk; punch the relocatable binary on the paper tape punch; send listing to the first line printer.

ASM/L (A,B,C, DP4:D))

Assemble as separate files A, B, C, and D in DP4. Produce A.RB, B.RB, C.RB, and D.RB, and listing files A.LS, B.LS C.LS, and D.LS; place all output in the current directory. See Chapter 3 for an explanation of parentheses.

ASM/L PARU/S PROG1 PROG2)

Assemble PROG1 and PROG2, producing PROG1.RB; scan the user parameter file PARU.SR on pass one to find values for symbols in PROG1 and PROG2. Send the listing to PROG1.LS

BASIC

Invoke the BASIC interpreter

Format:

BASIC)

Invoke the BASIC interpreter, which allows you to work in the BASIC language. A BASIC save file, configured via BASIC System Generation (BSG), must exist before the system can execute this command. BASIC SYSGEN procedures are described in the *Extended BASIC System Manager's Guide.*

Switches:

None

Example:

```
R
BASIC)
.
.          BASIC Commands
.
BYE)       (Return to CLI)
R
```

BATCH

Execute a BATCH job stream (RDOS)

Format:

BATCH *[jobfile$_1$ ... jobfile$_n$] [outfile/O] [logfile/G]*

Invoke the BATCH monitor, which can process one or more serial job streams without operator intervention. Each job stream consists of one or more jobfiles, which are input via a device or disk file. Each user job in jobfile must contain certain job control commands; it may also contain data sets and reference devices. Many CLI commands and RDOS utilities are available under BATCH.

If you omit switches, the line printer is the output file (called SYSOUT), the card reader is the input file, and the console is the log file. When you enter the BATCH command, the BATCH monitor searches for jobfile.JB; if not found, a search is made for jobfile. Consult the *BATCH User's Manual* for more information.

Global Switches:

None

Local Switches:

name/O	Send output to file name; default (line printer) name is SYSOUT.
name/G	File name is the log file.

Example:

```
R
BATCH DAILY.JB MT0:1 LOG/G)
! .
.      } BATCH Commands
.
!EOF)
....
BATCH TERMINATED 8/19/77 10:48:05
R
```

The command line defines a job stream with jobs serially input via two files: the first is disk file DAILY.JB, and the second is file 1 of MT0. Job log information goes to a disk file named LOG, and the line printer is SYSOUT by default.

BOOT

Execute a different operating system or stand-alone program

Format:

BOOT { disk
       [directory:] sysname }

Release the current system and bootstrap system or program sysname into execution. The disk bootstrap program, BOOT.SV, must reside in the primary partition of the disk which holds sysname. All directories involved must be initialized. The disk must be one of the directories named in Table 2-2 (RDOS) or Table 2-3 (DOS). The directory: specifier cannot include an RDOS subdirectory or DOS directory.

The sysname can be the name of an operating system, or a properly-built stand-alone program (see below). It can also be the name of a link entry to a system, if both the system's save and overlay files are linked. If sysname is a link, all intermediate directories must be initialized. In every case, when the bootstrap succeeds, the directory containing sysname becomes the master directory.

If you omit a sysname (first form), BOOT will ask for one when it gains control. You then enter the system or program name (or simply RETURN to bootstrap a system with the default name, SYS.SV). You must use this form of the BOOT command to execute certain stand-alone programs.

Stand-alone Programs

A stand-alone program is a program which runs without an operating system. You can execute a stand-alone program directly via the BOOT command if it conforms to the following rules:

1. The save file must be randomly organized. Normally, it will be, but transferring it with the XFER command can change its organization.

2. It must begin at location 0. Location 0 must contain either 0 or a byte pointer to a text string (the program name). If it contains the latter, the text string will be displayed on the console.

3. Location 2 must contain either the save file starting address or -1. If it contains the starting address, the program will self-start; if -1, the computer will halt after the load (press CONTINUE to continue).

4. Location 5 must contain 0.

See RLDR global /C or /Z switch for loading information.

To bootstrap a program which does not follow these rules, execute a MKSAVE/Z command on it; then BOOT the directory which contains it. When BOOT asks *FILENAME?* type the program name, and append the /A switch. BOOT will load the program; you must then execute it via the front panel switches.

Restart Feature

If the data switches are all up when you type BOOT, BOOT will try to bootstrap the system specified. If you omit the sysname, it will try to bootstrap the default system name, SYS.SV. When it finds and bootstraps a system, it then attempts to chain to a file named RESTART.SV. This mechanism is part of a real-time process control restart feature, described under the .BOOT call in your system reference manual. If you don't want it, make sure the data switches aren't all up when you type BOOT.

For a microNOVA: see .BOOT command in your system reference manual for restart features.

Local Switch:

/A (In response to the *FILENAME?* query). This is a stand-alone program, which does not conform to BOOT conventions.

Examples:

```
BOOT DP4)
MASTER DEVICE RELEASED
FILENAME? MYSYS)
```

Load BOOT.SV from DP4, release the master directory, ask for a sysname, get answer (MYSYS), then bootstrap (MYSYS), then bootstrap MYSYS in DP4. MYSYS requests the date and time, then activates its CLI.

```
BOOT DP0F:SYS64K)
```

Bootstrap the system named SYS64K in directory DP0F.

```
BOOT DP0:RTOS)
```

Bootstrap the RTOS system save file in directory DP0.

```
BOOT DP1)
MASTER DEVICE RELEASED
FILENAME? FOO.SV/A)
```

Invoke BOOT and execute the absolute program FOO.SV. FOO.SV does not conform to BOOT conventions, thus the /A switch.

BPUNCH
Punch a file in binary

Format:

BPUNCH filename$_1$ *[filename$_2$ ...]*

Punch a given file or files in binary on the high speed punch. To punch an ASCII file, use PUNCH. The command is the eqivalent of the XFER command:

XFER filename(s) $PTP)

The files may come from any device.

Switches:

None.

Examples:

BPUNCH FEE.SR FI.SR FO.RB FUM.RB)

Punch source files FEE and FI, and relocatable binary files FO and FUM on the high speed punch.

BPUNCH $PTR)

Punch a paper tape duplicate of the tape in the high-speed reader.

BUILD
Build a file from filenames

Format:

BUILD outputfilename *[filename$_1$ ...]*

Build an output file from filenames in the current directory. You can use template characters and date switches to select filenames. The BUILD command deletes outputfilename (if it exists) before creating the new output file.

Global Switches:

/A	Include all filenames which have permanent and nonpermanent attributes. (By default, only nonpermanent files will be built into outputfilename).
/K	Do not include link entries.
/N	Do not include extensions to filenames when they are built into outputfilename.

Local Switches:

mm-dd-yy/A	Include only the names of files created this date or after. Arguments mm (month) and dd (day) can be one or two digits.
mm-dd-yy/B	Include only the names of files created before this date.
name/N	Do not include names that match name in this outputfile.

Template Characters:

Permitted

Examples:

```
BUILD ABC -.SR TEST -.)
```

Create file ABC, and write two categories of files into it: those whose names have the .SR extension, and those whose names begin with TEST and have no extension.

```
BUILD MYFILE -.RB ABC.RB/N)
```

Create MYFILE, and insert all RBs except ABC into it.

```
BUILD RECENT -.SR 8-8-77/A)
```

Build file RECENT, from all files with a .SR extension created on or after August 8, 1977.

```
BUILD OUTPUT -TEXT)
R
PRINT @OUTPUT@)
.
.
.
```

This sequences produces OUTPUT which contains all filenames whose last four characters are TEXT and have no extension; it then prints these files.

CCONT
Create a contiguous file

Format:

CCONT filename$_1$ blocks$_1$ *[filename$_n$ blocks$_n$]...*

Create one or more contiguous files with all data words zeroed. Each file has the C characteristic, no attributes, and the fixed length in blocks you specify. A disk block is 256 words long. For details on contiguous files, see Chapter 2 of your system reference manual.

Global Switches:

/N	Do not zero data words in the new file. By default, the system zeroes each data word.

Examples:

```
CCONT ALPHA 20)
```

Create the contiguous file, ALPHA, in the current directory, with a length of 20 disk blocks.

```
CCONT TEST 100 DP1:TEST1 51)
```

Create two files, TEST in the current directory and TEST1 on disk DP1.

CDIR

Create a subdirectory (RDOS) or directory (DOS)

Format:

CDIR (sub)directoryname

Create a (sub)directory with the .DR extension in the current directory. If the current directory is a subdirectory, you'll receive an error message. Each (sub)directory acts as a pointer to files created in it and has a fixed size of 512 bytes.

To prevent confusion, we recommend that you give each (sub)directory a unique name.

Switches:

None.

Example:

CDIR DP1:BETH)

Create (sub)directory BETH.DR in DP1. Equivalent commands would be:

```
DIR DP1)
CDIR BETH)
```

CHAIN

Overwrite the CLI with another program

Format:

CHAIN savefilename)

Overwrite the CLI by chaining savefilename into execution on the current level. Under RDOS, use CHAIN only in special circumstances and with caution. The savefilename should chain control back to the CLI via system call .EXEC (see Chapter 4 of your system reference manual).

In RDOS, the program which you chain issues a .RTN instruction from level zero in the background, the system will halt in exceptional status; also, CTRL-A and CTRL-C may cause exceptional status.

Note that you cannot CHAIN when the log file is open in the current ground; this file must be closed (ENDLOG) first.

Global Switch:

/D Pass control to the debugger.

Local Switches:

None.

Example:

CHAIN is useful when you have a program which needs to use five swap levels (when you execute a program from the level 0 CLI, it runs on level 1, hence has only levels 2, 3 and 4 available).

In DOS only, CHAIN can speed up utility program commands (e.g., CHAIN ASM MYFILE)).

CHATR

Change a file's attributes

Format:

CHATR filename$_1$ { + / − } attrib$_1$ [

[...filename$_n$ { + / − } *..attribs$_n$]*

Add or remove file access attributes *(attribs)* to a file. To add an attribute, precede it with a plus sign (+); to remove an attribute, precede it with a minus (-) sign; to remove all attributes, enter 0 as an argument. Enter multiple *attribs* as a single argument. You can deny *execute* access to a save file by removing its S attribute; while it lacks S, no one can execute it.

DOS users: When you copy any file from a write-protected diskette via MOVE or DUMP, the copy receives attributes PW and characteristic A, which means that you cannot alter or delete the copy.

Attributes:

N	Do not allow linking to this file (actually, the link can be created, but not used).
P	Make this a permanent file. A file cannot be deleted or renamed while it has this attribute. Note that none of the following commands recognize a Permanent file (unless you include local /A): BUILD, DUMP, LIST, LOAD, MOVE.
R	Read-lock this file. No one can read this file (or copy it via DUMP, FDUMP, MOVE, or XFER) while it has the R attribute.
S	Designate this a save file. RLDR assigns this when it processes binaries into a save file. If you remove it, a save file cannot execute.
W	Write-lock this file. No one can modify this file while it has the W attribute. (It can, however, be deleted.)
0	Remove all removable attributes (except S).
*	Retain all existing attributes.
?	User-defined attribute.
&	User-defined atribute.

File Characteristics (CHATR cannot change these):

A	This is an attribute-protected file. You can set A with system call .CHATR. Once set, A cannot be removed.
C	This is a contiguous file.
D	This is a random file.
T	This file is a partition (RDOS).
Y	This file is an RDOS subdirectory or DOS directory.

Examples:

```
CHATR YOURFILE 0 MYFILE R)
```

Remove all attributes of YOURFILE and read-lock MYFILE.

```
CHATR MYFILE -R+W)
```

Re-enable MYFILE for reading, but prevent it from being modified.

```
CHATR PASSWORDS R)
CHATR PASSWORDS +R)
CHATR PASSWORDS *R)
CHATR PASSWORDS -R)
```

The first command removes all attributes from PASSWORDS, then assigns the R attribute. The second and third commands have the same effect; each *adds* the R attribute to existing PASSWORDS attributes. The fourth command removes the R attribute.

CHLAT

Change a file's link access attributes

Format:

CHLAT filename$_1$ $\{\pm\}$ attrib$_1$ ↑

[...filename$_n$ $\{\pm\}$ *...attribs$_n$]*

Add or remove link access attributes *(attribs)* to a resolution file. This controls the type of operation link users can perform on this file from other directories. To add an attribute, precede it with a plus sign (+); to remove an attribute, precede it with a minus (-) sign; to remove all attributes, enter 0 as an argument. Enter multiple attributes as a single argument.

Attributes:

N	Do not allow linking to this file (actually, the link can be created, but not used).
P	Make this a permanent file. Link users cannot delete or rename a file while it has this attribute.
R	Read-lock this file. Link users cannot read this file (but can execute it) while it has the R attribute.
S	Designate this as a save file. RLDR assigns this when it processes binaries into a save file.
W	Write-lock this file. Link users cannot modify this file while it has the W attribute. (They can, however, delete it if they DELETE the link).
0	Remove all removable attributes (except S).
*	Retain all existing attributes.
?	User-defined attribute.
&	User-defined attribute.

Examples:

```
CHLAT YOURFILE 0 MYFILE RW)
```

Allow link users of YOURFILE to do anything; prevent link users from reading or modifying MYFILE.

```
DIR DP0:MYDIR)
R
LINK PASSWORDS/2)
R
```

This creates a link in MYDIR to file PASSWORDS in DP0. To continue this example, assume that PASSWORDS has no *file* access attributes, which means that anyone in DP0 can read it, modify it, delete it, etc.

```
DIR DP0)
R
CHLAT PASSWORDS R)
R
TYPE MYDIR:PASSWORDS)
FILE READ PROTECTED: MYDIR: PASSWORDS
.R
TYPE PASSWORDS)
.
.
```

PASSWORDS is read-locked to a link user only; because it has no CHATR read-lock, it can be read directly.

CLEAR

Set file use count to zero

Format:

CLEAR *[filename] ...]*

Clear the file use count in one or more SYS.DR entries. Each file's use count is 1 or more when the file is open; it should be 0 when the file is closed. If a system fails when a file is open, its use count will remain nonzero when the system is rebootstrapped, and you will be unable to delete or rename the file. (You can check use counts with the LIST/U or LIST/E commands.)

If an abnormal RDOS shutdown occurs, you'll receive the message PARTITION IN USE -- TYPE C TO CONTINUE when you next bring up the system. Type C, log on, then type CLEAR/A/V/D from the master directory and all other directories that were initialized at the abnormal shutdown. You'll also receive the PARTITION message if someone shut off power to the computer or master directory drive before RELEASEing the master directory. Again, CLEAR all directories that were initialized when power was turned off.

A DOS system gives no PARTITION message, but you should follow the same CLEAR procedure as for RDOS.

CLEAR works only from background level 0 (CLI level) when no foreground program is running (RDOS).

Global Switches:

/A	Clear use count in all files in the current directory (except the current CLI.OL, CLI.ER, sysname.OL, sysname.TU, and LOG.CM). To clear these files, enter their names as arguments to CLEAR. (Arguments are ignored when you use this switch.)
/D	Clear device entries also (RDOS).
/V	Verify filenames cleared on the console.

Local Switches:

None.

Examples:

```
CLEAR/A/V/D)
.
.
```

Clear use count of all files and devices in the current directory (except those mentioned under /A, above).

```
CLEAR/V DP4:TEMP)
CLEARED DP4:TEMP
R
```

Clear use count of file TEMP, in directory DP4.

CLG

Compile, load and execute a FORTRAN IV program

Format:

CLG filename$_1$ *[...filename$_n$]*

The CLG command can compile files using FORTRAN IV, assemble them with the Extended Assembler, load them with the Relocatable Loader, and execute the final save file. The FORTRAN compilation and assembly steps are optional. In the argument list, you can include FORTRAN IV source files *(filename.FR)* files in assembly language *(filename.SR)* or assembled binary files *(filename.RB)*. Output includes one or more temporary source files, one or more binary files, and the save file.

For this command to execute, the following files must be in the current directory (or available via links): CLG.SV, FORT.SV, FIV.SV, ASM.SV, RLDR.SV, RLDR.OL, SYS.LB, and FORT.LB. You can produce FORT.LB by merging your four original FORTRAN libraries under this name, with the LFE M command. For further details on FORTRAN IV, see the *FORTRAN IV User's Manual* and the *FORTRAN IV Runtime Library User's Manual.*

Unless you restrict output with global switches, all compilations, assemblies, and the load map will go to the listing file you specify with the /L switch.

In addition to the local switches below, you can append RLDR local switches to each binary name. You can create overlays with CLG as you can in the RLDR command line, by enclosing them in brackets.

Global Switches:

/B	List only the source (input) program.
/E	Suppress compiler error messages. Assembler error messages are not suppressed.
/M	Suppress load map (symbol table).
/T	This is a multitask program. (The multitask FORTRAN library, FMT.LB, must be available on disk.)

Local Switches:

name/A	Assemble and load this file; do not compile.
name/E	Direct error messages to this filename.
name/L	Direct listing output to this filename.
name/O	Execute RLDR phase only on this file; do not compile or assemble it.

Extensions:

On input, search for filename.FR; if not found, search for filename. If /A is specified, search for filename.SR; if not found, search for filename. If /O is specified, search for filename.RB; if not found, search for filename.

On output, produce temporary assembler source files, filename.SR. Produce binary files, filename.RB$_1$. Produce save file filename.SV$_1$ (unless local /S was included in the command line).

Examples:

```
CLG A B C)
```

Compile A.FR (or A), producing A.SR; assemble A.SR into A.RB and delete A.SR. Do the same with B.SR (or B) and C.SR (or C). Load A.RB, B.RB, C.RB and FORT.LB to produce A.SV. Execute A.SV.

```
CLG/B MAIN $LPT/L)
```

Compile MAIN.FR (or MAIN), list MAIN.FR on the line printer, and produce MAIN.SR. Assemble MAIN.SR into MAIN.RB and delete MAIN.SR. Process binary MAIN and the FORTRAN IV library producing MAIN.SV. Execute MAIN.SV.

```
CLG/T MAIN FB/O SR/A $LPT/L)
```

Here, program MAIN is a file in FORTRAN IV, FB is an assembled binary, and SR is in assembly language. The /T switch specifies multitask mode; for multiask compilation FMT.LB is called from disk.

The command processes all files as appropriate, producing MAIN.SV; then it executes MAIN.SV. It also lists all compiler, assembler, and loader output on the line printer.

COPY

Copy one diskette to another diskette (DOS)

Format:

COPY sourcediskette destinationdiskette

Copy all files and directories from one diskette to another. Before you issue this command, sourcediskette must be initialized and destinationdiskette must NOT be initialized. All files in destinationdiskette will be destroyed, and the files from sourcediskette will be moved to it, along with all file directory information. If destinationdiskette is not a Data General diskette, you must format it with the appropriate formatter program and run DOSINIT.SV before copying to it.

Because the root portion of BOOT.SV (blocks 0 and 1) is not part of the file structure, it will not be copied. If you will want to bootstrap from a destinationdiskette which lacks a bootstrap, you must install BOOT on it.

After the copy is complete, destinationdiskette will have the same system directory and directory structure as sourcediskette. The diskettes are virtually identical, therefore, both diskettes cannot be initialized at one time. DOS doesn't allow two directories with the same name to be initialized at one time; it will recognize only the first directory initialized until you release the first and INIT (or DIR to) the second.

Global Switch:

/L — List copied files on the line printer. If you have no line printer, type LINK $LPT $TTO) to get listings on the console.

Examples:

```
DIR DP0)
R
COPY DP0 DP1)
R
```

You can now remove the copy from slot DP1 and store it.

```
INIT DP2;INIT DP3)
R
COPY DP2 DP3)
DEVICE ALREADY INITIALIZED: DP3
R
RELEASE DP3)
R
COPY DP2 DP3)
R
```

DOS will not copy to an initialized diskette.

NOTE: For COPY to work, the current DOS system must permit at least two directories, as determined at SYSGEN by your answer to the *MAXIMUM NUMBER OF SUBDIRECTORIES* ... query.

CPART
Create a secondary partition (RDOS)

Format:

CPART partname blockcount

Create the secondary partition named partname, with the length specified in blockcount. The new partition will be a contiguous disk file and receive the extension .DR.

You cannot create a partition with less than 48 disk blocks. If the blockcount you enter is not an integer multiple of 16, the system will truncate it to the next lower multiple.

To prevent confusion, we recommend that you give each partition and subdirectory a unique name.

Switches:

None.

Example:

```
DIR DP2)
R
CPART ALEPH 128)
R
```

The DIR command makes primary partition DP2 the current directory; the CPART command creates secondary partition ALEPH on DP2. An equivalent command would be CPART DP2:ALEPH 128). ALEPH is 128 disk blocks long, and logically distinct from primary partition DP2; you will not be able to initialize the subdirectories which are built (CDIR command) within it without its name.

CRAND
Create a random file

Format:

CRAND filename$_1$ *[filename$_2$ ...]*

Create a randomly organized file in any directory. Each random file will have the D characteristic, no attributes, and a length of zero; it will grow as required during use. For details on random files, see Chapter 2 of your system reference manual.

Switches:

None.

Examples:

```
CRAND RANDFILE)
```

Create RANDFILE in the current directory.

```
CRAND ACCUFILE DP1:VELVET:GROWTHFILE)
```

Create ACCUFILE in the current directory, and GROWTHFILE in directory VELVET on DP1.

CREATE

Create a sequential file (RDOS)

Format:

CREATE filename$_1$ *[...filename$_n$]*

Create one or more sequential files. Each file will have zero length and no attributes. For details on sequential files, see Chapter 2 of your system reference manual.

In DOS, CREATE creates a random file.

Switches:

None.

Examples:

CREATE ALPHA)

Create a sequential file, ALPHA, in the current directory.

CREATE TEST TEST1 DP1:TEST2)

Create three filenames, TEST, and TEST1 in the current directory and TEST2 in the primary partition on disk DP1.

DEB

Load a program into memory and go to the debugger

Format:

DEB savefilename)

Debug a program about to be executed. A symbolic debugger must have been loaded as part of the program save file, as described under the RLDR command. The DEB command transfers control to the debugger in this save file; the debugger then outputs a carriage return and awaits debugging commands.

While debugging, you can examine memory, set break points, and run the program. After making any necessary changes, you can return to the CLI by issuing the ESC-V Debug command; this saves the program in file BREAK.SV. You can then rename BREAK.SV as described under the SAVE command.

Normally, you'll want to correct the source program at some point, then reassemble and reload it.

Switches:

None.

Examples:

DEB PGM.SV)	Invokes debugger, with file PGM.SV.
START+15/*006751*	Examine a location.
START+15$B	Set breakpoint at START+15.
$R	Start the program executing.
.	
.	
.	
7B START+15	Breakpoint.
0... 1... 2... 3...	
$P	Continue debugging.
.	
.	
$V	ESC echoes as $. Create the break file and return to CLI. (Only exit from IDEB.)
or CTRL A	Return to CLI if PGM.SV does not have a CTRL-A address (or if you don't want a break file).

R

DELETE

Delete a file or directory

Format:

DELETE filename, *[...filename,]*

Delete the files named. You may include directory specifiers if the directory has been initialized. If you try to delete a link, the link will persist but the *resolution file will be deleted* (attributes permitting). To remove a link entry, use the UNLINK command.

To delete a (sub)directory or partition, RELEASE it, then type DELETE directoryname.DR).

Global Switches:

/C	Confirm each deletion. The system repeats each filename, then waits for you to confirm the deletion by typing a carriage return. To prevent the deletion, press any key other than RETURN.
/L	List deleted files on SLPT (overrides /V).
/V	List names of deleted files on the console.

Local Switches:

mm-dd-yy/A	Delete only files created this date or after. Arguments mm (month) and dd (day) can be one or two digits.
mm-dd-yy/B	Delete only files created before this date.
name/N	Do not delete any files that match this name.

Template Characters:

Permitted only when the filename is in the current directory.

Examples:

DELETE LIMIT.- DP4:TEMP)

Delete all files having the name LIMIT and any extension (including null), e.g., LIMIT.SR, LIMIT.RB, LIMIT.SV, LIMIT. Then, delete file TEMP in DP4.

DELETE/V -.LS)

DELETED A.LS
DELETED COM.LS
DELETED MAP.LS

Delete all files with the .LS extension, and list their names.

DELETE/C A-B)	Ask for confirmation of each file before deletion:
AZYB:)*	Delete file AZYB.
AXWB: #	Do not delete AXWB.

When you confirm a deletion with a carriage return, the systen echoes an asterisk (*). Any other character evokes no echo.

DIR

Change the current directory

Format (RDOS):

DIR *[directory][:secondarypartition] [:subdirectory]*

Format (DOS):

DIR *[diskette][:directory]*

Change the current directory. At bootstrap time, the master directory becomes the current directory. The DIR command specifies another directory as the current directory. If the directory hasn't been initialized, DIR will initialize it.

Switches:

None.

Examples:

DIR ACCTSDUE)

Make directory ACCTSDUE the current directory.

DIR DP1:MYDIR)

Make directory MYDIR, in directory DP1, the current directory.

DIR %MDIR%:BILLING

Make directory BILLING, in the master directory, the current directory.

DIR DP0F:MYPART:SUBDIR)

Make SUBDIR, which is a subdirectory in secondary partition MYPART, on disk DP0F, the current directory (RDOS).

DISK

Display the number of disk blocks used and remaining

Format:

DISK)

Return a decimal count of the number of blocks left and the number of blocks used in the current partition or diskette. DISK returns two figures, and their sum is always integer-divisible by 16.

Aside from user files, the bootstrap root and the system require at least 16_{10} blocks; other directories require some space for their system and map directories.

Switches:

None.

Examples:

DISK)
LEFT: 520 USED: 88

This response, from a diskette, indicates that 520 out of a total of 608 blocks are still available.

DISK)
LEFT: 1478 USED: 8298

This response indicates that 1478 blocks from the original 9776 are still available for use.

DUMP

Copy one or more disk files in DUMP format

Format:

DUMP dumpfilename *[filename, ...][old filename/S]* new filename...

Copy files from the current directory onto a given file or device (dumpfilename). If you specify filenames (with or without a template), only matching filenames are dumped. The *filename* also can be a secondary partition, subdirectory, or DOS directory name (with the .DR extension) but it cannot be a disk name. If you omit *filenames*, all nonpermanent files in the current directory are dumped. This includes all subordinate directories, if any. In other words, if the current directory is a primary partition or diskette, the whole disk's nonpermanent files will be dumped; if it is a secondary partition, it and all its subdirectories and files will be dumped.

dumpfilename can be a disk directory (specifiers permitted), or it can be a mag tape, cassette, or paper tape file.

During the dump, directory information is written as a header to each dumped file. This information includes the filename, length, attributes and other data; it is taken from the directory's SYS.DR. When you LOAD the DUMPed file, the header data goes back into its directory's SYS.DR. For the format of dump files, see Appendix C.

A note to DOS users: If the source for the dump is a write-protected diskette, all files in the dump file will receive the attributes APW.

NOTE: DUMP's complementary command is LOAD. If you use FDUMP, a faster version of DUMP, you must reload the files with FLOAD.

Global Switches:

/A	Dump all files, permanent and nonpermanent.
/K	Do not dump links.
/L	List the dumped files on $LPT (overrides/V).
/S	Dump a file on segments of paper tape. The file is punched in segments of up to 20K bytes each. Each tape segment is headed by unique segment number, which enables the system to verify that tapes will be reLOADed in proper sequence.
/V	Verify DUMPed filenames on the console.

Local Switches:

mm-dd-yy/A	Dump only files created this date or after. Arguments mm (month) and dd (day) can be one or two digits.
mm-dd-yy/B	Dump only files created before this date.
name/N	Don't dump files that match name.
oname/S name	Assign name to this file in the dump, but retain its old name (oname) in the current directory.

Template Characters:

Permitted.

Examples:

```
DIR DP0F)
R
DUMP/A/L MT0:0 12-20-77/A)
```

Dump all directories and files in DP0F created on or after December 20, 1977, to file 0 of the tape on drive MT0. This tape then provides backup for these files. Listing of dumped files goes to the line printer.

```
DUMP/V DP1:SAVEFILES -.SV)
.
.
```

Dump to file SAVEFILES on DP1 all files in the current directory with the .SV extension.

```
DUMP/A MT0:3 APRIL/S APRIL.DU)
```

Dump permanent file APRIL to file 3 of the tape on MT0. Name the dumped copy of APRIL *APRIL.DU* (which will allow it to be loaded into the same directory as APRIL, if necessary).

```
DUMP/A/L DP4:770722.DU -.RB/N↑)
  7-22-77/A)
```

Create file 770722.DU (named for a date) on DP4 for dump; then dump all files (except .RBs) created on or after July 22, 1977 to file 770722.DU on DP4.

EDIT

Invoke the Text Editor program

Format:

EDIT *[filename]*

You can use the text editor to create and edit text files. If you include the optional *filename* argument the editor will automatically execute the command UY *filename* ESC ESC upon being loaded. The *filename* must already exist. For more information, see the *Text Editor User's Manual.*

Switches:

None.

Examples:

```
EDIT)
*                Program is ready to accept commands.

.
.                You issue editing commands.
.

*H$$             You terminate the editor and return to
                 CLI by pressing the H key followed by
                 the ESC key twice.
R
```

ENDLOG
Close the file opened by LOG

Format:

ENDLOG *[password]*

This command closes the log file which you opened by a previous LOG command. You must close this file before you can TYPE or PRINT it. To print or delete the log file after closing it, you must type its full name: LOG.CM, for a background log file, and FLOG.CM, for a foreground log file. If the previous LOG command included a *password* argument, you must use the password with the ENDLOG command.

This command, ENDLOG *[password]*, appears in the log file.

Switches:

None.

Example:

```
LOG/H GSTONE)
R
.
.
.
ENDLOG GSTONE)
```

The password GSTONE is required since it was used when the log file was opened.

ENPAT
Insert patch data in a patch file

Format:

ENPAT filename

A patch is a one-word change to a save or overlay file. The ENPAT and PATCH utilities allow you to enter multiple patches to .SV or .OL files, and install them easily. Patches are most often used to update operating system files; Data General supplies current patches with its system software.

The ENPAT command creates patch file filename, or opens it for appending if it already exists. ENPAT then asks five questions about each one patch, accepts valid answers, and places them in filename. To install the patches in filename, use the PATCH command.

For a save (.SV) file, a patch can be a symbol, octal number, or expression including a symbol, operator, and octal number. The most common operators are:

+ (addition)
- (subtraction)
@ (indirection)

The PATCH program will not be able to resolve symbols unless you have a load map of the save file on disk. You can instruct SYSGEN to save such a map with the SYSGEN local switch/L.

For an overlay (.OL) file, a patch must be an octal number or expression.

ENPAT asks the following questions for each one-word patch. If you give an invalid response, it returns an error message (explained in Appendix A), and repeats the question. First, it asks:

1. SAVE FILE (0) OR OVERLAY FILE (1)?

   Answer 0 if the patches in this file will be installed in a save file, 1 for an overlay file. Next, it asks:

2. PATCH LOCATION:

   Enter the location to be patched: number, symbol or expression, and). ENPAT now asks about the contents of this location:

3. CURRENT CONTENTS:

   Type in the current contents of the location, and). For most Data General-supplied patches, this is an octal number. Now, ENPAT asks:

4. NEW CONTENTS:

   Respond with the new contents for the file location specified in 2. Next, ENPAT asks about conditions for patch installation:

5. CONDITIONAL:

   If you want to install the patch unconditionally, type). To install the patch only if a symbol is defined in the load map, type the symbol name and). (You might do this if the patch relates to a given module, and you don't know if the module is part of your system. For example, assume that a patch relates only to a device driver named ALPHA. You want this patch installed only if ALPHA is defined in your system load map, so you'd type ALPHA) .) To install the patch only if the symbol is *not* defined in the load map, type a minus sign (-), then the symbol name; e.g., -ALPHA).

   If you want this patch installed conditionally, and the condition is the same as you entered for the previous patch, simply enter an up arrow (↑) and) in response to this question. This allows you to easily enter multiple patches which concern one condition.

   Next, ENPAT wants to know if you have more patches to enter:

6. EXIT (0=NO 1=YES?)

   Answer 0) to return to question 1. and enter another patch; answer 1) to create the patch file and return to the CLI.

Example:

Assume that you want to insert the following patches in file IPB, for later installation in save file SYS.SV:

Location	Old Contents	New Contents
IPBQ-1	100000+IPSTK	401 @0
(401 @0)+15	0	100000+IPSTK
401	401 @0	401 @0+16

```
ENPAT IPB)
CREATING NEW PATCHFILE
SAVE FILE (0) OR OVERLAY FILE (1) ? 0)
PATCH LOCATION: IBPQ-1)
CURRENT CONTENTS: 100000+IPSTK)
NEW CONTENTS: 401 @0)
CONDITIONAL: )
EXIT (0=NO 1=YES) ? 0)
SAVE FILE (0) OR OVERLAY FILE (1) ? 0)
PATCH LOCATION: (401 @0)+15)
CURRENT CONTENTS: 0)
NEW CONTENTS: 100000+IPSTK)
CONDITIONAL: )
EXIT (0=NO 1=YES) ? 0)
SAVE FILE (0) OR OVERLAY FILE (1) ? 0)
PATCH LOCATION: 401)
CURRENT CONTENTS: 401 @0)
NEW CONTENTS: 401 @0+16)
CONDITIONAL: )
EXIT (0=NO 1=YES) 1)
R
```

The PATCH command example continues this, and shows installation of the patches in patchfile IPB.

EQUIV

Rename a disk or tape drive (RDOS)

Format:

EQUIV newname oldname

Assign a temporary name to a disk or mag tape unit or cassette. Newname replaces the oldname until the device is RELEASEd. The oldname must be one of the disk or tape units shown in Table 2-2.

Properly applied, this command gives your programs device-independence; you can write a generic device specifier into your programs, and EQUIV it into a specific device at run time (see the example below).

You must EQUIV a device before it has been initialized; the new name will exist only until the device has been released. After release, all devices revert to their original specifiers. You cannot EQUIV a name for a secondary partition, subdirectory, or the master device.

Global Switch:

/P Pause for operator intervention.

Example:

While writing a program, you have written all magnetic tape file references to MYTAPE. This gives your program device independence at runtime. Before running the program, you issue the commands:

```
EQUIV/P MYTAPE MT0)
MOUNT MYTAPE ON UNIT MT0, STRIKE ANY KEY
R
INIT MT0)
FILE DOES NOT EXIST: MT0
R
INIT MYTAPE)
R
LOAD/N MYTAPE:1)
MASM2      05/17/77
MASM4.LS   06/01/77
R
EQUIV MT0 MYTAPE)
DIRECTORY IN USE
R
RELEASE MYTAPE)
R
INIT MYTAPE)
FILE DOES NOT EXIST: MYTAPE
R
```

After the EQUIV command, the system doesn't recognize the EQUIVed device by its original name (MT0); you must INIT it by its EQUIVed name. You can't EQIV a different name until you RELEASE it; after release, it discards the EQUIVed name and reverts to its original name.

EXFG

Execute a program in foreground memory (RDOS)

Format:

EXFG programname

The EXFG command executes a program in foreground memory. It works the same way as the filename command at the beginning of this chapter, except that it tries to execute the program between foreground boundaries, instead of in background space. You can terminate a foreground program by typing CTRL-F from the background console. In a mapped system, you establish the foreground boundaries with the SMEM command; in an unmapped system, you set boundaries in the program itself, with the RLDR /F and /Z address switches.

As shown in Chapter 3, foreground and background programs can access the same directory, initialize or release directories, and open the same file.

The way you use EXFG will depend on whether your system is mapped or unmapped. RDOS offers several system calls for handling foreground and background programs, as described in Chapter 6 of the *RDOS Reference Manual.*

Before you execute any program in the foreground, you may want to check its memory requirements with the SEDIT or OEDIT utilities.

Mapped Systems

In a mapped system, you can execute any program in the foreground if you have allotted enough memory with the SMEM command. You can execute noninteractive programs (like assemblies or compilations) in the foreground; or you can EXFG interactive programs like the CLI and access them through the foreground console(s).

A foreground CLI has different names for its files: LOG.CM, COM.CM, and CLI.CM are named FLOG.CM, FCOM.CM, and FCLI.CM.

To execute an assembly, compilation, or any program in the foreground, type the command line exactly as you would for the background, but precede it with EXFG *//E].*

Unmapped Systems

In an unmapped system, you must load a program with foreground memory boundary information before you can execute it in the foreground. Do this with switches in the RLDR command. Two utility programs you can run in an unmapped foreground are the Text Editor (EDIT.RB) and the Multiterminal Text Editor (MEDIT.RB). If you choose, you can configure one of these with boundary addresses via RLDR, and use it to edit text in the foreground (on a second console).

Before you EXFG any user program, load and run it in the background, and check its ZREL and NREL requirements with SEDIT, or OEDIT, or the load map before reloading it for execution in the foreground. Also, it'a a good idea to check the requirements of any background programs you plan to run concurrently (or CLI.SV, if you want to keep the CLI active in the background).

If you attempt to EXFG a program which would require space needed by the CLI, RDOS will reject the command, and display the message:

```
INSUFFICIENT MEMORY TO EXECUTE PROGRAM
```

When you execute a program in the foreground, the CLI remains active in the background. You can now try to execute another program in the background..

Global Switches:

/D	Pass program control to the debugger.
/E	Assign equal priority to foreground and background (normally, the foreground program has priority).

Examples:

Common:

```
EXFG DP1:RMON)
```

The system now tries to execute program RMON, on primary partition DP1, in foreground memory. In a mapped system, the foreground as set by SMEM must accommodate RMON; if RMON won't fit, the system will return the error message:

```
INSUFFICIENT MEMORY TO EXECUTE PROGRAM
```

In an unmapped system, RMON must have been loaded with foreground boundary information. If the foreground RMON creates for itself would overwrite CLI space in memory, the system will also display the message:

```
INSUFFICIENT MEMORY TO EXECUTE PROGRAM
```

In either system, if RMON executes, the CLI will remain active in the background; this user could then try to execute a different program in the background. Again, if the background lacks enough space to execute the program, RDOS will display the INSUFFICIENT MEMORY message.

Unmapped

Assume that program RECAST.SV has been run and debugged in the background, and that it has been written with system calls to communicate with the console. Also assume that the current RDOS system requires about 8,500 words, and that it is a 32K system. Using SEDIT or OEDIT, you check the NMAX of RECAST.SV and CLI.SV, then load RECAST for foreground execution:

```
RLDR RECAST.RB 250/Z 40000/F RECASTF.SV/S)
R
```

This configures RECASTF.SV to run the high section of user ZREL and NREL memory.

You now try to EXFG RECASTF; if it executes, you can try to execute another program in the background.

EXFG (continued)

Mapped

```
R
GMEM)
BG: 52 FG: 0
```

Check the memory allotment for each ground: 52K for background, none for foreground.

```
R
SMEN 20)
R
EXFG MAC/L/U PARU/S USR/S↑)
SOURCE<1,2,3,4,5>↑)
BACKUP PROG<1,2,3,4,5,6,7>)
R
```

Assemble a complex program in the foreground.

```
NEWFILE)
.
.
.
.
```

Execute NEWFILE in the background.

Also see the example near the end of Chapter 3.

FDUMP

Fast-dump all files in the current directory to magnetic tape (RDOS)

Format:

FDUMP *[MTn:tapefilenumber]*

The FDUMP command is a faster version of DUMP. You cannot specify filenames in this command; it copies all permanent and nonpermanent files in the current directory to tapefilenumber, on unit MTn. FDUMP works only with mag tape, and the dump files it produces must be loaded with FLOAD. It is faster than DUMP, and uses less tape.

With the /A switch (described below), you can specify a second drive to continue receiving the dump when file space on the first tape is exhausted. If you do not specify a second drive, FDUMP will prompt you to mount another tape when file space is exhausted.

FDUMP automatically INITs the drive specified, and RELEASEs it after command execution. FDUMP and FLOAD are utility programs which generally reside in the master directory; you can link to them from other directories.

Do *not* mix FDUMP and DUMP files on one tape. File access is easier if you FDUMP to file 0 only. FDUMP writes three EOF marks after each dumped file; this means that you *can* stack dumps on a single reel by dumping to file 0, 3, 6, 9, 12, and so on. When you FLOAD the dumpfiles, you will address these as files 0, 3, 6, 9, 12, and so on.

Note that FDUMP is a multitasking program, configured for a specific kind of system (e.g., unmapped NOVA); thus you cannot use your version on a different kind of system (e.g., mapped NOVA).

Global Switches:

/L	List filenames on the line printer as they are dumped.
/V	List filenames on the console as they are dumped.

Local Switches:

MTn:0/A	Continue dump on tape file 0 of alternate drive n when space is exhausted on first drive.
name/L	Send dumped filenames to file name.

Examples:

```
DIR DP0)
R
FDUMP/V MT0:0)
```

Fast-dump every file on DP0 to file 0 of the tape on drive MT0.

```
DIR DP0F:SUBDIR)
R
FDUMP MT0:0 MYFILE.-)
INVALID COMMAND STRING!
R
```

You cannot FDUMP specific filenames.

```
DIR DP0)
R
FDUMP/V MT0:0)
. . . .
. . . .
MOUNT NEXT REEL, STRIKE KEY WHEN READY
```

When space is exhausted, FDUMP prompts for another tape.

(The next example assumes one blank 2400-foot tape on MT0, another on MT1.)

```
DIR DZ0)
R
FDUMP/L MT0:0 MT1:0/A)
. . . .
```

Dump the contents of the 6060-series disk in unit 0 to the tape on MT0; when space runs out on this tape, continue the dump on file 0 of the tape on MT1. A listing goes to the line printer.

FGND

Describe whether or not a foreground program is running

Format:

```
FGND)
```

This command returns one of two messages, depending on whether or not a program is running in the foreground.

Switches:

None.

Example:

```
FGND)
NO FOREGROUND PROGRAM RUNNING
R
EXFG MYPROG)
R
FGND)
FOREGROUND PROGRAM RUNNING
R
```

FILCOM

Compare two files

Format:

FILCOM $filename_1$ $filename_2$

Compare two files, word by word, and print dissimilar word pairs in octal on your console (unless you use the /L switch). File organizations of the two files may differ; e.g., you can compare a random and contiguous file.

Local Switches:

name/L Send output to this file.

Example:

FILCOM YIN YANG $LPT/L)

Compare files YIN and YANG word-by-word. Print any dissimilar word pairs in octal on the line printer, along with their respective word displacements in the files:

025/	*044516*	*042530*
141/	*000014*	*020044*
142/	*000000*	*046120*
143/	*000000*	*052057*
144/	---	*046015*
145/	---	*000014*

YANG is two words longer than YIN --note the dashes for these locations in YIN. If either YIN or YANG were a null file, FILCOM would print the entire contents of the other file; it would print dashes for the null file.

FLOAD

Fast-load fast-dumped files into the current directory (RDOS)

Format:

FLOAD *[MTn:tapefilenumber]*

FLOAD is the counterpart of FDUMP; it loads all files in tapefilenumber into the current directory. The files must have been FDUMPed (not DUMPed). You cannot specify filenames in FLOAD; it tries to load all files in tapefilenumber. If a file in tapefilenumber already exists in the current directory, FLOAD prints a FILE ALREADY EXISTS message, does not load the file, and continues.

FLOAD automatically INITs (but does *not* RELEASE) the drive specified. The dump file header indicates the *current* directory, not the directory being loaded.

If you FDUMPed a file on multiple reels, on the same drive, FLOAD will request another tape when it has loaded all files on the current tape.

You cannot FLOAD a dumpfile which contains a partition into a secondary partition, subdirectory, or DOS directory. If you try, you'll get a DIRECTORY DEPTH EXCEEDED message and the load will abort. This might happen if you had dumped an entire disk, and tried to load this dumpfile when the current directory was a secondary partition, or subdirectory.

If the tape you are loading has stacked FDUMP files on it, these files have the same numbers for loading (0, 3, 6, etc.).

FLOAD, like FDUMP, is a multitasking program, configured for a specific kind of system; your version will not work on a different kind of system.

Global Switches:

/L Print filenames on the line printer as they are loaded.

/N Do not load files; display names on the console (or send them to the file specified by global or local /L).

/V Display filenames on the console as they are loaded.

Local Switches:

name/L Send filenames to file name.

Examples:

```
DIR DP0)
R
FLOAD/V MT0:0)
. . . .
. . . .
```

Fast-load all files in file 0 of the tape on MT0 into current directory DP0.

```
FLOAD/N MT0:0)
. . . .
. . . .
```

Display the filenames in MT0:0 on the console; do not load files.

In the last example for FDUMP, assume that you want to copy all files in the large dump to another big disk.

```
DIR DZ0)
FLOAD MT0:0 LOADLOG/L)
.
.
MOUNT NEXT REEL, STRIKE KEY WHEN READY
TAPE HAS WRONG REEL NO.
MOUNT CORRECT REEL AND STRIKE A KEY
.
```

This dump included two tapes. When FLOAD finished loading the first reel, it prompted for a second. This user mounted the wrong reel and struck a key; FDUMP then prompted for the correct reel. When the right reel was mounted and a key struck, FDUMP loaded the second reel. The load listing went to disk file LOADLOG.

FORT

Compile a FORTRAN IV file

Format:

FORT filename$_1$ *[...filename$_n$]*

Compile a FORTRAN IV source file. As output, you can specify a binary file, an intermediate source file, a listing file, or combination of all three. For this command to execute, the following files must be available in the current directory (or available via links): FORT.SV (the FORTRAN IV interface), FIV.SV (the compiler), and ASM.SV (the extended assembler). The command name, FORT, cannot be changed.

On input, the command searches for inputfilename.FR; if not found, it searches for inputfilename.

If you omit switches, the FORT command produces an intermediate source file, filename.SR (output of compilation). It then invokes an assembler which produces a binary file. After a successful assembly, the system deletes the intermediate source file. No listing is produced by the default command.

Global Switches:

/A	Suppress assembly (intermediate source file is deleted unless you use global/S). Useful for checking compile errors.
/B	List compiler source program input only.
/E	Suppress error messages from compiler. (Assembler error messages are not suppressed.)
/F	Equivalence FORTRAN variable names and statement numbers to symbols acceptable to the assembler. (Include /U to inform the Debugger of the /F names.)
/L	Produce listing file *(filename.LS$_1$)*.
/N	Do not produce relocatable binary file. Useful for checking assembly errors.
/P	Compile no more than 72 columns per line (as in punched card images).
/S	Save the intermediate source file.
/U	Append user symbols during the assembly phase (must be used with /F).
/X	Compile statements with *X* in column 1.

FORT (continued)

Local Switches:

name/B — Direct relocatable binary output to file name (overrides global /N).

name/E — Direct error messages to file name (overrides global /E).

name/L — Direct listing output to file name (overrides default name specified by global /L).

name/S — Direct intermediate source output to file name.

Examples:

```
FORT/L MAIN)
```

Compile and assemble FORTRAN IV program MAIN or MAIN.FR to produce binary file MAIN.RB; send both a compiler and an assembler listing to file MAIN.LS.

```
FORT/F/U DP1:TABLE $LPT/L)
```

Compile and assemble FORTRAN IV file TABLE.FR (or TABLE) on DP1 and produce binary file TABLE.RB. Write compiler and assembler listings to the line printer, and append user symbols during assembly.

FORTRAN
Compile a FORTRAN 5 file (RDOS)

Format:

FORTRAN *filename*$_1$ *[...filename*$_n$*]*

Perform a FORTRAN 5 compilation. Output may be a binary file, a listing file, or both. See the *FORTRAN 5 User's Manual* and the *FORTRAN 5 Supplement* for more information.

The CLI searches for *filename.FR;* if not found, it searches for *filename*.

Global Switches:

/B — Brief listing (compiler source program input).

/C — Check syntax only.

/D — Debug aid. Give line number and program name on all runtime error messages.

/I — Don't list source lines from INCLUDE files.

/K — Do not delete compiler temporary files after compilation.

/L — Produce a listing file named filename.LS$_1$.

/N — Compile, but do *not* create a binary file. Useful for checking compile errors.

/P — Punched card input: only the first 72 characters of each input line are used as compiler source code, but the entire input line is sent to the listing file (if one exists).

/S — Generate code for subscript checking.

/X — Compile statements with *X* in column 1.

Local Switches:

name/B	Give binary output file this name.
name/E	Send error output to file name.
name/L	Send listing output to file name (overrides global /L).

Examples:

```
FORTRAN/L MAIN)
```

Produce binary file MAIN.RB with both a compiler and a source listing to file MAIN.LS.

```
FORTRAN BMARK FLIST/L)
```

Compile FORTRAN 5 source program BMARK or BMARK.FR and output source and compiler listings to disk file FLIST.

FPRINT

Print a disk file in the specified format

Format:

```
FPRINT filename
```

Print a disk file in either bytes, decimal, hexadecimal, or octal, with printable ASCII characters on the right side. Any nonprinting characters are reported as periods (.). If you omit switches, the locations are printed in octal on the console. You can specify a first and last location with local switches.

FPRINT offsets locations by 16_8 unless you include the /Z switch.

Global Switches:

/B	Print in byte format.
/D	Print in decimal.
/H	Print in hexadecimal.
/L	Use the line printer.
/O	Print in octal (default).
/Z	Print locations starting at zero.

Local Switches:

n/F	Start at location n (octal).
name/L	Send output to file name (overrides global /L).
n/T	Stop at location n (octal).

Examples:

```
FPRINT/L TE1)
```

Print file TE1 on the line printer. The mode is octal by default.

```
FPRINT/B/L MYFILE 2000/F 3500/T)
```

Print MYFILE in byte format on the line printer, from location 2000 to 3500.

GDIR

Display the current directory name

Format:

GDIR

Switches:

None.

Examples:

GDIR)
MANHATTAN

MANHATTAN is the current directory.

RELEASE %GDIR%)

Release the current directory.

GMEM

Display background/foreground memory size (mapped RDOS)

Format:

GMEM

Display the current memory allotment to background and foreground program areas. The size is given in 2048-byte blocks. GMEM works only on machines with mapped addressing.

Switches:

None.

Example:

GMEM)
BG: 30 FG: 34

61,440 bytes of memory (30 times 2,048) are available to the background, and 69,632 bytes of memory are available to the foreground.

GSYS

Display the name of the current operating system

Format

GSYS

Switches:

None.

Examples:

```
GSYS)
SYS
```

The current operating system is named *SYS*. The system does not display its save file extension, .SV.

```
R
XFER/A STTI OPSYS.MC)
MESSAGE THE CURRENT SYSTEM IS %GSYS%)
CTRL Z
R
OPSYS)
THE CURRENT SYSTEM IS SYS64K
R
```

GSYS can be a system-defined variable; the current system is SYS64K.SV

GTOD

Display the time and date

Format:

GTOD

Switches:

None.

Example:

```
GTOD)
12/17/77 21:24:20
```

The message indicates that the time is 20 seconds after 9:24 p.m., and the date is December 17, 1977.

Local Switches:

name/L Send filenames to file name.

Examples:

```
DIR DP0)
R
FLOAD/V MT0:0)
. . . .
. . . .
```

Fast-load all files in file 0 of the tape on MT0 into current directory DP0.

```
FLOAD/N MT0:0)
. . . .
. . . .
```

Display the filenames in MT0:0 on the console; do not load files.

In the last example for FDUMP, assume that you want to copy all files in the large dump to another big disk.

```
DIR DZ0)
FLOAD MT0:0 LOADLOG/L)
.
.
MOUNT NEXT REEL, STRIKE KEY WHEN READY
TAPE HAS WRONG REEL NO.
MOUNT CORRECT REEL AND STRIKE A KEY
.
```

This dump included two tapes. When FLOAD finished loading the first reel, it prompted for a second. This user mounted the wrong reel and struck a key; FDUMP then prompted for the correct reel. When the right reel was mounted and a key struck, FDUMP loaded the second reel. The load listing went to disk file LOADLOG.

FORT

Compile a FORTRAN IV file

Format:

FORT filename$_1$ *[...filename$_n$]*

Compile a FORTRAN IV source file. As output, you can specify a binary file, an intermediate source file, a listing file, or combination of all three. For this command to execute, the following files must be available in the current directory (or available via links): FORT.SV (the FORTRAN IV interface), FIV.SV (the compiler), and ASM.SV (the extended assembler). The command name, FORT, cannot be changed.

On input, the command searches for inputfilename.FR; if not found, it searches for inputfilename.

If you omit switches, the FORT command produces an intermediate source file, filename.SR (output of compilation). It then invokes an assembler which produces a binary file. After a successful assembly, the system deletes the intermediate source file. No listing is produced by the default command.

Global Switches:

/A	Suppress assembly (intermediate source file is deleted unless you use global/S). Useful for checking compile errors.
/B	List compiler source program input only.
/E	Suppress error messages from compiler. (Assembler error messages are not suppressed.)
/F	Equivalence FORTRAN variable names and statement numbers to symbols acceptable to the assembler. (Include /U to inform the Debugger of the /F names.)
/L	Produce listing file *(filename.LS$_1$)*.
/N	Do not produce relocatable binary file. Useful for checking assembly errors.
/P	Compile no more than 72 columns per line (as in punched card images).
/S	Save the intermediate source file.
/U	Append user symbols during the assembly phase (must be used with /F).
/X	Compile statements with *X* in column 1.

FORT (continued)

Local Switches:

name/B	Direct relocatable binary output to file name (overrides global /N).
name/E	Direct error messages to file name (overrides global /E).
name/L	Direct listing output to file name (overrides default name specified by global /L).
name/S	Direct intermediate source output to file name.

Examples:

```
FORT/L MAIN)
```

Compile and assemble FORTRAN IV program MAIN or MAIN.FR to produce binary file MAIN.RB; send both a compiler and an assembler listing to file MAIN.LS.

```
FORT/F/U DP1:TABLE $LPT/L)
```

Compile and assemble FORTRAN IV file TABLE.FR (or TABLE) on DP1 and produce binary file TABLE.RB. Write compiler and assembler listings to the line printer, and append user symbols during assembly.

FORTRAN

Compile a FORTRAN 5 file (RDOS)

Format:

FORTRAN *filename*$_1$ *[...filename*$_n$*]*

Perform a FORTRAN 5 compilation. Output may be a binary file, a listing file, or both. See the *FORTRAN 5 User's Manual* and the *FORTRAN 5 Supplement* for more information.

The CLI searches for *filename.FR;* if not found, it searches for *filename.*

Global Switches:

/B	Brief listing (compiler source program input).
/C	Check syntax only.
/D	Debug aid. Give line number and program name on all runtime error messages.
/I	Don't list source lines from INCLUDE files.
/K	Do not delete compiler temporary files after compilation.
/L	Produce a listing file named filename.LS$_1$.
/N	Compile, but do *not* create a binary file. Useful for checking compile errors.
/P	Punched card input: only the first 72 characters of each input line are used as compiler source code, but the entire input line is sent to the listing file (if one exists).
/S	Generate code for subscript checking.
/X	Compile statements with *X* in column 1.

Local Switches:

name/B	Give binary output file this name.
name/E	Send error output to file name.
name/L	Send listing output to file name (overrides global /L).

Examples:

```
FORTRAN/L MAIN)
```

Produce binary file MAIN.RB with both a compiler and a source listing to file MAIN.LS.

```
FORTRAN BMARK FLIST/L)
```

Compile FORTRAN 5 source program BMARK or BMARK.FR and output source and compiler listings to disk file FLIST.

FPRINT

Print a disk file in the specified format

Format:

```
FPRINT filename
```

Print a disk file in either bytes, decimal, hexadecimal, or octal, with printable ASCII characters on the right side. Any nonprinting characters are reported as periods (.). If you omit switches, the locations are printed in octal on the console. You can specify a first and last location with local switches.

FPRINT offsets locations by 16_8 unless you include the /Z switch.

Global Switches:

/B	Print in byte format.
/D	Print in decimal.
/H	Print in hexadecimal.
/L	Use the line printer.
/O	Print in octal (default).
/Z	Print locations starting at zero.

Local Switches:

n/F	Start at location n (octal).
name/L	Send output to file name (overrides global /L).
n/T	Stop at location n (octal).

Examples:

```
FPRINT/L TE1)
```

Print file TE1 on the line printer. The mode is octal by default.

```
FPRINT/B/L MYFILE 2000/F 3500/T)
```

Print MYFILE in byte format on the line printer, from location 2000 to 3500.

GDIR

Display the current directory name

Format:

GDIR

Switches:

None.

Examples:

GDIR)
MANHATTAN

MANHATTAN is the current directory.

RELEASE %GDIR%)

Release the current directory.

GMEM

Display background/foreground memory size (mapped RDOS)

Format:

GMEM

Display the current memory allotment to background and foreground program areas. The size is given in 2048-byte blocks. GMEM works only on machines with mapped addressing.

Switches:

None.

Example:

GMEM)
BG: 30 FG: 34

61,440 bytes of memory (30 times 2,048) are available to the background, and 69,632 bytes of memory are available to the foreground.

GSYS

Display the name of the current operating system

Format

GSYS

Switches:

None.

Examples:

```
GSYS)
SYS
```

The current operating system is named *SYS*. The system does not display its save file extension, .SV.

```
R
XFER/A STTI OPSYS.MC)
MESSAGE THE CURRENT SYSTEM IS %GSYS%)
CTRL Z
R
OPSYS)
THE CURRENT SYSTEM IS SYS64K
R
```

GSYS can be a system-defined variable; the current system is SYS64K.SV

GTOD

Display the time and date

Format:

GTOD

Switches:

None.

Example:

```
GTOD)
12/17/77 21:24:20
```

The message indicates that the time is 20 seconds after 9:24 p.m., and the date is December 17, 1977.

INIT

Initialize a directory or tape drive

Format:

INIT { tape drive
directory *[subordinate-dir][:sub-subordinate-dir]* }

By initializing a device or directory, you introduce it to the operating system; before the introduction, all of its files are effectively closed. At any moment, many directories on a device can be initialized, while only one remains the current directory.

Bootstrapping automatically INITs the directory which holds the current system.

After you INIT a directory or device, you can access all files within it; you can use the colon specifier (e.g., DP0:AB) at will. All files on the initialized tape unit, directory or diskette are available until you release (RELEASE command) the device or directory.

The DIR command performs all INIT functions, and selects another current directory - which saves a step if you want to change the current directory. You cannot use DIR on a tape drive.

Global Switch:

/F — Full initialization (tapes or disk specifiers only). This clears all existing files and information from on the disk or tape; on a disk, it writes a new system file directory and map directory. If you issue INIT/F to a secondary partition or subdirectory, the /F switch is ignored and a simple INIT occurs. Full initialization of a tape unit rewinds the tape and writes two EOF's at its beginning, effectively erasing the tape by allowing the system to write files on it.

Local Switches:

None.

Examples:

```
INIT DP4)
```

Initialize the disk in DP4.

```
INIT/F MT1)
```

Mag tape or cassette initialization rewinds the tape to the beginning-of-tape mark. Full (/F switch) initialization of MT1 rewinds the tape on drive MT1, erases it and writes two EOF's at the beginning. (The system begins writing files on the tape at the double end-of-file.)

```
INIT DP1:ABC)
```

Initialize directory ABC on DP1. All files in DP1 and ABC are now accessible. If ABC is an RDOS partition, you must INIT or DIR to its subdirectories, or direct file access with explicit global/directory specifiers (including colon separators).

LDIR

Display the name of the last current directory

Format:

LDIR

Return the name of last directory "DIRed" to.

Switches:

None.

Example:

```
GDIR)
SUBDIR
R
DIR DP0)
R
LDIR)
SUBDIR
R
DIR %LDIR%)
R
GDIR)
SUBDIR
```

This sequence gets the current directory, changes it to DP0, changes it back, then uses LDIR as a CLI variable, and checks the result.

LFE

Edit library files

Format:

LFE A library$_1$ ... *[binary$_1$ ...]*

LFE D library *[newlibrary/O]* binary$_1$ ...

LFE I library *[newlibrary/O]* binary$_1$ ...

LFE N *[newlibrary/O]* binary$_1$ *[...binary$_n$]*

LFE M library$_1$ library$_2$ ... *[newlibrary/O]*

LFE R library *[newlibrary/O]* binary$_1$ binary $_1$, *[...binary$_n$ binary$_n$]*

LFE T library$_1$ *[...library$_n$]*

LFE X library binary$_1$ *[...binary$_n$]*

Invoke the Library File Editor utility, which allows you to create, edit, and analyze library files. Library files are groups of assembled binary files, which conventionally have the extension ".LB". You may want to assign this extension to the libraries you create. See the LFE manual for more information.

LFE works with copies of original files; it does not alter the originals.

Function keys A, D, I, M, N, R, T, and X direct LFE operation. The *library*, *binary*, and *newlibrary* words represent existing library names, relocatable binary files (RBs), and new library names, respectively.

Each LFE function key has the following effect:

A Analyze symbols and sizes of one or more libraries or *binaries*. To analyze separate RBs, append the local /B switch to the binary name. The analysis lists symbols, symbol types, and reference flags; it does not produce a new library file. The default listing file is the line printer.

Analyze Switches:

/M This global switch (A/M) instructs LFE to analyze all libraries following as a single library.

binary/B This *binary* is an RB file, not a library.

binary/F Insert a form feed before printing analysis of *binary*.

LFE (continued)

D Delete *binary* file(s) from library, producing *newlibrary*. If you omit *newlibrary/O*, LFE names the new library D.L1.

I Insert *binary* file(s), combining them with binaries in library, producing *newlibrary*. If you omit *newlibrary/O*, LFE names the new library I.L1. If you omit /A or /B (below), LFE inserts all binaries at the beginning of the new library.

Insert Switches:

binary/A Insert after. The binary name(s) following the switch will be inserted after binary.

binary/B Insert before. The binary name(s) following the switch will be inserted before binary.

M Merge libraries into one file named *newlibrary*. If you omit *newlibrary/O*, LFE names the new library M.L1.

N Create new library file *newlibrary* from one or more binary files. If you omit *newlibrary/O*, LFE names the new library N.L1.

R Replace *binary* file(s) in library with other binary file(s), producing *newlibrary*. The first binary argument is the original binary name; the second is the binary name that will replace the original. If you omit *newlibrary/O*, LFE names the new file R.L1.

T Send the titles of binaries in library to the listing file (SLPT by default).

X Extract one or more binary files from library. You must specify the original .TITL of each binary. Output is one or more binaries named *binary.RB*.

Local Switches:

name/L Send listing output (A or T keys) to file name. The default listing file is the first line printer, SLPT.

name/O Assign this name to the new library file. The name overrides LFE default names in the D, I, L, M, N, and R commands.

Extensions:

If you enter no extensions for library or binary, LFE searches for library.LB or binary.RB respectively. If it does not find them, it searches for library or binary. LFE does *not* assign the extension ".LB" to newlibrary names you specify; you may want to include it in the name. See the above description of keys for the default names of new libraries.

Examples:

```
LFE N $PTP/O A.RB C.RB/B)
```

Create a new library file from binaries A.RB and C.RB; punch it on the paper tape punch.

```
LFE A/M ZUT1.LB ZUT0.RB)
```

Analyze as one library ZUT1.LB and the binary file ZUT0.RB. Analysis goes to the line printer.

```
LFE I MAL.LB MAL1.LB/O RSK/A MOD1)
```

In library MAL.LB, insert binary MOD1.RB after binary RSK. Name the new library MOD1.LB.

```
LFE A MYLIB(1,2,3).LB MYLIB(1,2,3).AN/L)
```

Analyze libraries MYLIB1.LB, MYLIB2.LB, and MYLIB3.LB; store analysis in disk files MYLIB1.AN, MYLIB2.AN, and MYLIB3.AN.

LINK

Create a link entry to a file

Format:

LINK { resfilename/2 | linkname *[directory:]* resfilename }

Create a link entry in the current directory to a resolution file (resfilename) or to another link entry.

Use the first form to link to a file in the current directory's parent directory. The parent directory holds the current directory; it can be a disk (e.g., DP0) or an RDOS secondary partition. This form gives the link the same name as the resolution file. Since utility programs are usually in the master directory (along with the operating system), you can link to each utility file using this format, if the parent directory is the master.

Use the second format if: 1) you want your link's name to differ from the resolution file's (it will be an alias name); or 2) the resfilename is not in the current directory's parent directory. For an alias linkname, you can use any legal filename, but remember that the link will be useless if you forget its name. For either 1) or 2), if the resolution file is a save or overlay file, your linkname must include the .SV or .OL extension.

Before you can create or use a link, all directories involved in the resolution chain must be initialized.

The system gives no error message if it can't find the resolution file, so you should check a link after you create it to see if it works. For example, assume that the current directory is DP1, and you want to link to file EDIT.SV in DP0:

```
LINK EDIT.SV/2)
R
EDIT MYFILE)
LINK DEPTH EXCEEDED: EDIT.SV
R
```

Resolution file EDIT.SV doesn't exist on DP1 -- so the link was made to itself. Thus you must remove the link and recreate it with a directory specifier:

```
UNLINK EDIT.SV)
R
LINK EDIT.SV DP0:EDIT.SV)
R
EDIT MYFILE)
*
```

The editor prompt (*) means that the link works.

Some utilities use an overlay file or other files in addition to their save files; you must also link these files. See each utility command for the names of its files.

You can link to a file whose attributes forbid linking (CHATR and CHLAT commands), but you cannot use that link.

Switches:

None (except local /2)

Examples:

Assume that the current directory, DP0, has one secondary partition, SECONDPART, which has one subdirectory, SUBDIRA. Figure 4-1 shows this structure, along with links to the Text Editor. (We chose EDIT because it uses only one file.) For DOS, assume that SECONDPART is a directory, and skip sequences 2 and 3, below.

1.
```
DIR SECONDPART)
R
LINK EDIT.SV/2
R
```

   This creates link entry EDIT.SV to the editor in SECONDPART's parent directory: DP0. Anyone in directory SECONDPART can now edit a file without using a directory specifier.

2.
```
DIR SUBDIRA)
R
LINK TX.SV DP0:EDIT.SV)
R
```

   This creates an alias link entry TX.SV to EDIT.SV in DP0. From SECONDPART, the command TX will work exactly as the command EDIT works.

3.
```
LINK DP0:SUBDIR:EDIT.SV EDIT.SV)
R
```

   This creates the link in SUBDIR to the editor. We needed the first directory specifier because SUBDIR was not the current directory.

4. DIR DP0)
   R
   LINK ALIAS.SV EDIT.SV)
   R

   This sequence creates an alias name in the same directory as the editor. The command ALIAS will work exactly as the command EDIT works.

   To create a link entry to another disk or diskette, simply insert the directory specifier before the resfilename; e.g.,

   DIR DP1)
   R
   LINK EDIT.SV DP0:EDIT.SV) or %MDIR%:EDIT.SV)
   R

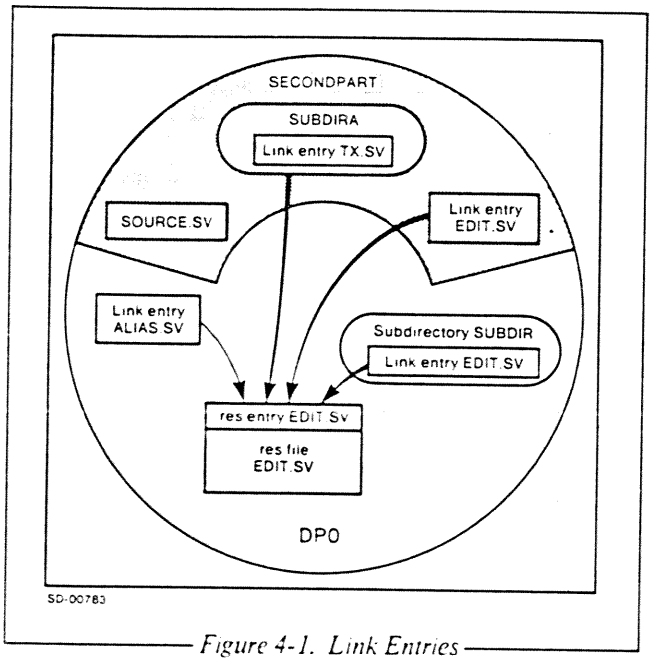


Figure 4-1. Link Entries

LIST
List file directory information

Format:

LIST { *[filename$_1$ ...]* / *[directory-specifier:filename]* }

List information about one or more files or link entries from the current directory or other directories. If you omit an argument, the system lists all nonpermanent files and link entries in the current directory.

For each file, LIST returns the filename and one or more of the following: file size in bytes, file access attributes, link access attributes, file creation date and time, date last opened, file starting address in octal (UFTAD of UFD), and use count. The link access attributes -- if any -- are, preceded by a right slash. The file starting address is enclosed within brackets.

For link entries, the list includes the link name, the resolution filename, and the directory specifier (if any) given when the link was created. A commerical at sign (@) means that the link was originally made to a file on its directory's parent partition or diskette.

The following lists file attributes (or link access attributes) and their meanings:

Character	Meaning
P	Permanent file: cannot be deleted or renamed.
S	Save file (core image).
W	Write-protected file: cannot be written.
R	Read-protected file: cannot be read.
A	Attribute-protected file, whose attributes cannot be changed. You cannot remove the A attribute.
N	No resolution entry allowed: links can be made, but not used.
&	First user-definable attribute (bit 9).
?	Second user-definable attribute (bit 10).

The following lists file characteristics and their meanings:

Character	Meaning
D	This file is randomly organized.
C	This file is contiguously organized.
T	This file is a partition (RDOS).
Y	This file is an RDOS or DOS directory.

Global Switches:

/A	List all files within the current directory, both permanent and nonpermanent, giving filename, file characteristics and attributes, byte count, and link access attributes.
/B	Brief listing: list only filenames.
/C	List creation time (month/day/year hour:minute).
/E	List every category of file information (overrides /B, /C, /F, /O, and /U switches).
/F	List the logical address of the first block in the file (0 if unassigned).
/K	Do not list links.
/L	Send a listing to the line printer.
/N	List links only.
/O	List date file last opened (month/day/year).
/S	Sort the output list alphabetically.
/U	List file use count (in decimal).

Local Switches:

mm-dd-yy/A	List files created this date or after. Arguments mm (month) and dd (day) can be one or two digits.
mm-dd-yy/B	List files created before this date.
name/N	Do not list files that match this name.

Template Characters:

Permitted only when the *filename* argument is in the current directory.

Examples:

```
LIST/E/A)
```

This command lists on the console all information for every file and link entry in the current directory. A typical line of information would look like this:

```
A.SV 8160 SD/N 12/23/77 13:56 12/23/77 [000164]0
```

In this example, A.SV consists of 8,160 bytes, is a randomly organized save file, and is protected from link access by the N attribute. It was created at 1:56 p.m. of the 23rd day of December, 1977, was last accessed on that same date, has a starting logical block address of 164, and has a file use count of zero.

Typical lines describing link entries would look like the following:

```
ABC.SV      DP0:DEF.SV
```

In this example, the link entry name is ABC.SV; it is an alias. The resolution file is DEF.SV in directory DP0.

```
EDIT.SV @:EDIT.SV
```

In this example, the link entry name is EDIT.SV; it was created with the same name as the resolution file, in a directory subordinate to the resolution file's.

```
LIST/K/S -.SV 11-2-77/A)
```

This command alphabetically lists all save files (.SV extension) created after November 1, 1977. It does not list links; output goes to the console.

LOAD
Reload dumped files

Format:

LOAD dumpfilename *[filename$_1$ ...]*

Load a previously dumped file from a given device or directory (input filename) into the current directory. If you omit filenames and switches, all nonpermanent files in the input file are loaded. With global switches, you can select filenames for LOADing, or you can simply display the filenames in the dump file.

The LOAD command can load only those files which were previously DUMPed; it cannot load a fast-dumped (FDUMPed) file or an XFERed file. Files to be loaded must bear different names from files in the current directory (unless you specify the /N, /O, or /R switches). Neither DUMPing nor LOADing change a file's access or link attributes, creation date, or directory characteristics.

If files were dumped on segments of paper tape using the DUMP/S command, you must follow the DUMP sequence when you LOAD them. Failure to follow the same sequence will evoke a CLI runtime error message.

Global Switches:

/A	Load all files, including permanent files.
/B	Display a brief listing of loaded files.
/E	Suppress nonfatal error messages.
/K	Do not load link entries.
/L	List loaded filenames on the line printer. (Overrides /V switch; if used with global /N, produces listing - files are not loaded.)
/N	Do not load files; output the filenames to the console.
/O	Delete current file if it exists and replace with file of the same name being loaded.
/R	Load most recent version of file. When a file to be loaded has the same name as a file in the current directory, the system checks both files' creation dates. If the existing file is older, it is deleted and replaced by the file awaiting loading. If the existing file is not older, it is retained and the dumped file is not loaded.
/V	Verify the load by listing filenames loaded on the console. Filenames in a directory are listed before the directory name, and they are indented 2 spaces; directory names are preceded by "*".

Local Switches:

mm-dd-yy/A	Load only files created this date or after. Arguments mm and dd can be one or two digits.
mm-dd-yy/B	Load only files created before this date.
name/N	Don't load files that match name.

Template Characters:

Permitted only when the *filename* argument is not in a directory in the dump file.

Examples:

```
LOAD/V MT0:1)
```

Load onto disk all previously dumped nonpermanent files on file 1 of the tape on MT0; verify names loaded.

```
LOAD/V $PTR -.SV)
LOAD $PTR. STRIKE ANY KEY.
EDIT.SV
ASM.SV
.
.
.
```

Load all nonpermanent files with the extension. .SV and list the files loaded.

```
LOAD/L/A MT1:6 -.SV 12-15-76/A TEST -.SV/N)
```

Load from file 6 of the tape on MT1 all files with the .SV extension except those files whose names begin with the characters TEST, and any files created before December 15, 1976; list them on the line printer.

```
DIR DZ0)
R
LOAD/V/R DP4:770722.DU -.SR)
```

Load (into directory DZ0) all .SR files from dumpfile 770722.DU, on DP4. (See DUMP command example.) If any file to be loaded has the same name as a file in DZ0, check the files' creation dates. Do not load the file unless the dumped version is newer.

LOG

Start recording in the log file

Format:

LOG *[password] [directory/O]*

The log file records CLI dialog that appears on the console in a file named LOG.CM (FLOG.CM for a foreground CLI). The CLI creates the log file if it doesn't exist, then records each line of CLI dialog in it. Only one current log file may exist at a time in any ground.

You cannot examine, print, or delete the log file while it is open. You can close it with the ENDLOG command (RELEASing the master device or giving the BOOT command also closes it). Unless it is deleted, the log file will retain all information; when the system is rebootstrapped, it will continue recording dialog when you type LOG again.

You can use a *password* to prevent the log file from being closed inadvertently. The password is an optional argument of up to 10 alphanumeric characters. If you specify *password*, you must use the same password in the ENDLOG command to close LOG.CM.

The *directory* argument indicates a destination for the log file other than the current directory. You must initialize the directory before using its name.

Global Switches:

/H	Place a header at the beginning of the log file. This header consists of the title - LOG FILE, directory, and date information.
/T	Trace the execution of CLI commands. This tells the CLI to write each command in its final form to the log file before executing it. All trace lines will be preceded by the symbols ==> when LOG.CM is printed.

Local Switch:

name/O	Write the log file in directory name.

Examples:

```
LOG/H GSTONE)
```

Record all CLI dialog to file LOG.CM in the current directory. The password is GSTONE, and on output, the header of this file will look like:

```
***LOGFILE***GATE [DP0:SYS] 12/17/77 3:0:0
```

GATE is the name of the current directory.

Assume that you have built a macro file named LAST.MC, which contains the following commands:

```
DIR %LDIR%;MESSAGE NOW IN DIRECTORY %GDIR%
```

Assume also that the value of %LDIR% is DP0F.

If you opened the log file with the global /T switch and type:

```
LAST)
```

the system would write the following lines into the log file:

```
LAST
==> LAST
==> DIR DP0F
==> MESSAGE NOW IN DIRECTORY DP0F
NOW IN DIRECTORY DP0F
R
```

If you had opened the log file *without* /T, these lines would appear in the log file:

```
LAST
NOW IN DIRECTORY DP0F
R
```

MAC

Assemble source files with the Macroassembler to produce a relocatable binary (.RB) file.

Format:

MAC filename$_1$ *[...filename$_n$]*

MAC is a macroassembler, which assembles assembly-language source files into a relocatable binary file. Unless you include switches, MAC produces a binary file named *filename.RB$_1$* and no listing file. To process MAC's output into a save file, use the RLDR command.

If you want to use MAC from another directory, create a link entry to three files: MAC.SV, MACXR.SV, and MAC.PS.

NOTE: If you receive many undefined symbol messages from MAC, try creating a new MAC.PS file. This is easy to do; see Chapter 6 of the *Macroassembler User's Manual.*

Global Switches:

/A — Add semipermanent symbols (defined by pseudo-ops) to the assembler cross-reference listing (used with global or local /L).

/E — Suppress error messages on the console (unless there is no listing file).

/F — Generate an extra form feed as necessary after each listing page to produce an even number of listing pages. By default, MAC generates one form feed after listing each *filename.*

/K — Keep MAC.ST after the assembly. By default, MAC.ST is deleted at that time.

/L — Append an assembly listing to disk file *filename.LS$_1$*, creating the file if necessary.

/M — Flag multiply-defined symbols on pass one.

/N — Do not build a relocatable binary file (useful for checking assembly errors).

/O — Override all listing suppression controls (as specified with .NOLOC, .NOMAC, etc.).

/S — Skip pass two and copy MAC.ST (symbol table and macro definitions) to file MAC.PS. Delete the old MAC.PS (if any) first.

/T — Recognize and store eight-character symbols from the source file(s). (Cross-reference will show five-character symbols.) Normally, MAC recognizes and stores only the first five characters of symbol names. The binary produced is in extended .RB format. For restrictions on use of this switch, see Chapter 6 of the *Macroassembler User's Manual.*

/U — Include user symbols in the relocatable binary output.

/Z — For DGC personnel: include DGC proprietary heading at the top of each listing page.

Local Switches:

name/B — Give relocatable binary file this name (overrides global /N).

name/E — Send error messages to file name.

name/L — Send listing and cross-reference to file name (overrides global /L).

name/S — Skip file name on pass two of assembly. Use this switch only if name contains no storage words. Macro definition files can be skipped on pass two.

name/T — Use file name as permanent symbol file for this assembly. If you omit this switch, the assembler uses file MAC.PS.

Extensions:

The CLI always searches for filename with the .SR extension; if not found, it searches for filename. The relocatable binary file is always named filename.RB$_1$, unless you give another name with local /B, append local /S to filename.RB$_1$, or use the .RB pseudo-op. The disk file produced by global /L is always named filename.LS$_1$, unless you append local /S to filename$_1$.

Error Codes:

If a line of source code contains an error, the assembler places a letter at the left margin of the offending line in the listing. It can insert no more than three codes per line.

Code	Meaning
A	Addressing error.
B	Bad character.
C	Macro error.
D	Radix error.
E	Equivalence error.
F	Format error.
G	Global symbol error.
I	Parity error (input).
K	Conditional assembly error.
L	Location counter error.
M	Multiply-defined symbol error.
N	Number error.
O	Field overflow or stack error.
P	Phase error.
Q	Questionable line error.
R	Relocation error.
U	Undefined symbol error.
V	Variable symbol error.
X	Text error.

Examples:

```
MAC/L Z)
```

Assemble source file Z.SR or Z, producing relocatable binary file Z.RB; send listing to file Z.LS.

```
MAC LIB/S A B C SLPT/L)
```

Assemble files A.SR or A, B.SR or B, and C.SR or C, producing A.RB. Scan file LIB (or LIB.SR) on pass one for macro definitions and values for externals in A, B, and C. Send a listing to the line printer.

```
MAC/L/U EX<1 2 3 4>)
```

Assemble EX1, EX2, EX3, and EX4 (with or without .SR extensions), producing EX1.RB. Include user symbols in EX1.RB; send a listing to EX1.LS. (See Chapter 3 for an explanation of angle brackets.)

MCABOOT

Boot an operating system under the other CPU via an MCA line (RDOS).

Format:

MCABOOT/F { MCAT:n | MCAT1:n } [{ *sysname/S* | *program/S* }] *[filename...]*

MCABOOT { MCAT:n | MCAT1:n } [{ *sysname/S* | *program/S* }]

The MCABOOT command can transmit an operating system or executable program via an MCA line. The command is most useful when the receiving CPU's disk has no RDOS system or an inappropriate RDOS system (if the receiver's disk had a system, it could be bootstrapped conventionally).

Read the formats as follows:

MCAT	Multiprocessor communications adapter transmitter.
n	MCA receiver (range 1-15).
sysname/S	System you want to transmit (you can omit this to transmit a system named SYS).
program/S	Any program which BOOT can execute (see BOOT command).
filename	A file(s) that you want to send along with the system or program (no directory specifiers allowed).

For a successful transmission, you must follow the order of arguments shown in the formats.

Use the first format only when the receiving CPU's disk is new, or holds valueless material; the /F switch specifies full initialization, which destroys all existing files on the disk. If you specify /F, the CLI save and overlay files will be sent along with any other filenames you enter; the receiving CPU will write all these to its disk after the full initialization.

Use the second format when the receiver's disk holds useful data. It performs a partial initialization with overlays, and does not send the CLI files. You can transmit only one *sysname* or executable program in the second format.

MCABOOT (continued)

For the transmission to succeed, both the sending and receiving CPUs must participate. From the time you issue the MCABOOT command, about 11 minutes can pass before the sending CPU times out (the default period is 655 seconds). During this time, or before you issue the command, the receiving CPU must request the transmission (the receiver can wait indefinitely). The operator at the receiving CPU requests a transmission by entering 100007_8 (for network MCA) or 100047_8 (for network MCA1), in the data switches, pressing RESET, then PROGRAM LOAD.

At transmission, if you specified an operating system, the MCA bootstrap will be sent, followed by the system save and overlay files. If you specify /F, CLI.SV and CLI.OL will also be sent, followed by the optional *filenames*. After the files have been transmitted, the system will request date and time information at the receiver's console, as in a traditional bootstrap. The CLI will then gain control at the receiving CPU.

If you have transmitted an executable program, it will gain control at the receiver.

Global Switch:

/F — Perform a full initialization on the receiver's disk (partial initialization with overlays is performed if you omit this switch).

Local Switch:

name/S — Send a system (or program) name (other than the default system (SYS.SV/SYS.OL)). You can omit the .SV extension.

Examples:

```
MCABOOT/F MCAT:2)
```

This command fully initializes the disk of CPU2, and sends the default operating system, SYS.SV/SYS.OL, to CPU2. Because full initialization was specified, CLI.SV and CLI.OL are also sent. CPU2 then writes the files to its disk, and requests log-on information on its console.

```
MCABOOT/F MCAT:2 FORT.SV FIV.SV FORT.LB)
```

This command transmits the default system (SYS.SV/SYS.OL) to CPU2 in the MCA system, with full initialization. The transmitting CPU, attached to the first MCA system, sends the FORTRAN IV compiler (FORT.SV and FIV.SV) and the FORTRAN IV runtime library. CPU2 also receives the CLI save and overlay files.

```
MCABOOT MCAT1:3 32K.SV/S)
```

This command transmits an operating system named 32K.SV, and its associated overlay file, 32K.OL, to CPU3, for a partial initialization with overlays. Both the transmitter and the receiver are attached to the second MCA system.

MDIR

Display the master directory name

Format:

MDIR

Print the name of the master directory on the console. This directory contains the system save and overlay files, plus push space for program swaps; for RDOS, it also contains the spool files and tuning file (if any).

Switches:

None.

Example:

```
MDIR)
DP0F
RELEASE %MDIR%)
MASTER DEVICE RELEASED
```

MEDIT

Invoke the Multiuser Text Editor (RDOS)

Format:

MEDIT terminals *[clock-units]*

MEDIT allows several users to edit text at the same time. Each user is served as though he were the only one, except for a possible slight degradation in response time. See the *Text Editor User's Manual* for more information.

terminals — sets the maximum number (decimal) of text-editing terminals for MEDIT to support. If you specify 8, for example, the system will support terminals 0-7.

clock-units — represents the number of system clock-units; it is optional. After this number of units has passed, task rescheduling will be forced.

Switches:

None.

Example:

```
MEDIT 6)
R                    MEDIT is ready to accept commands
.                    on six terminals.
.
                     Users issue editing commands.
CTRL-C               Recognized from master console
                     only.
R
```

MESSAGE

Insert a text string for later display

Format:

MESSAGE *['']*textstring *['']* . . . *[textstring] ['']*

After you enter your textstring, CLI displays your message on the console. A global switch allows you to delay further execution of the command line until someone strikes a console key.

This command is very useful for sequences of commands.

You can enter the textstring inside quotes (e.g., "HELLO"), or omit quotes. If you use quotes, the MESSAGE command will return all characters literally, except for the quotes (which are textstring delimiters), and any semicolons, carriage returns, and form feeds (which are command delimiters). A single quote within a quoted string is illegal. The textstring cannot include more than 72 characters.

If you omit quotes, the CLI will try to interpret certain characters if they are part of the string (e.g., %GDIR%, @TEST@). You cannot make angle brackets (< >), parentheses ( ), commas, or slashes (/) part of an unquoted string. Unquoted textstring delimiters are command delimiters; e.g.,), a semicolon (followed by another command), etc. The textstring cannot include more than 132 characters.

Global Switch:

/P — Pause after displaying textstring, display the message STRIKE ANY KEY TO CONTINUE, and wait for someone to strike a key on the console.

Local Switches:

None.

Examples:

```
XFER/A $TTI LOGON.MC)
DIR USERSDISK; CHATR SYS.<SV OL>) +W+P↑)
CHATR CLI.<SV OL ER> +W+P)
MESSAGE WELCOME ABOARD. USER DIRECTORIES)
MESSAGE AVAILABLE ARE: ; LIST -.DR)
CTRL-Z
R
```

Program LOGON would display the MESSAGE text, and obey the other commands:

```
LOGON)
WELCOME ABOARD. USER DIRECTORIES
AVAILABLE ARE:
SECONDPART.DR 507904 CTY
        SUBDIR.DR 512   DY
        SUBDIRA.DR512   DY
        .
        .
        .
R
```

You could build file ALLDONE the same way, and use the /P switch to delay execution until someone pressed a key:

```
XFER/A $TTI ALLDONE.MC)
CHATR SYS.<SV OL> -W-P CLI.<SV OL> -W-P)
MESSAGE THE CURRENT DIRECTORY IS %GDIR%;)
MESSAGE MOUNT DUMP TAPE ON MT0, AND READY)
MESSAGE DRIVE;)
MESSAGE THE TIME IS %TIME% ON %DATE%;)
MESSAGE/P ENJOY.)
INIT MT0; DUMP/V MT0:0)
CTRL-Z
R
```

MKABS

Make an absolute binary file from a disk save file

Format:

MKABS savefilename absolute-filename

Use MKABS to make an absolute binary file from a memory image (save) file.

After you convert an executable program into an absolute binary, you can execute it without a disk, via the binary loader.

Global Switches:

/S — Starting address switch. The starting address of the save file as specified in USTSA of the file will be used as the address for the absolute binary start block. Default is a null start block which halts the binary loader when loading is completed.

/Z — Begin save file at memory location zero; default is 16_8. This configures the file as a stand-alone program, which you cannot execute under RDOS, but which can execute by itself. (See RLDR global switch /Z.)

Local Switches:

n/F — n is the first address, relative to save file location 0, from which the absolute binary file is to be created.

n/S — n is the starting address. The absolute binary start block will have the address specified by the octal number that precedes this switch.

n/T — n is the last address, relative to save file location 0, to become a part of the absolute binary file.

Extensions:

The CLI searches for savefilename.SV; if it does not find it, it searches for savefilename.

Examples:

```
MKABS/Z FOO SPTP)
```

Punch an absolute binary file on the paper tape punch from file FOO.SV or, if not found, from FOO.

```
MKABS/Z A SPTP 1000/S)
```

Punch an absolute binary file with a start block that specifies 1000_8 as the starting address.

```
MKABS/Z SFIVE.SV SFILEABS.SV)
```

Make absolute disk file SFILEABS.SV from disk file SFIVE.SV.

MKSAVE

Make a disk save file from an absolute binary file

Format:

MKSAVE absolute-filename savefilename

Use MKSAVE to create a memory image (save) file from an absolute binary file. The disk file will automatically be given the S attribute. The system appends the .SV extension to your specified savefilename.

Global Switch:

/Z — Start save file at memory location zero rather than 16_8. Caution: See RLDR global /Z.

Example:

```
DIR DP0)
MKSAVE/Z $PTR DP1:A)
```

Make a disk save file called A.SV, in DP1 from the absolute binary file loaded in the paper tape reader. A.SV begins at memory location zero; you can execute it by typing BOOT DP0), and responding to the FILENAME? query with A.SV/A).

MOVE

Copy files from the current directory to any directory

Format:

MOVE destination-directoryname *[filename] ...]*

Copy one or more files (filenames) from the current directory to a destination-directory. As with DUMP, the directory information for each file is preserved - name, length, attributes, creation and last access time.

The destination-directory can be a partition or subdirectory; *filename* must be a filename in the current directory. You cannot MOVE a directory. If you omit *filenames* and switches, all nonpermanent files in the current directory are moved.

DOS users: If you copy a file with MOVE from a write-protected diskette, the copy automatically receives the attributes APW, which means that you can never delete or modify the copy.

Global Switches:

/A — Move all files, including permanent files.

/D — Delete original files after copying them.

/K — Do not move links.

/L — List moved filenames on the line printer (overrides /V switch).

/R — Move most recent version of the file. When a file to be moved has the same name as a file in destination-directory, the system checks both files' creation dates. If the file in destination-directory is older, it is deleted and *filename* replaces it. Otherwise, it is not moved.

/V — Verify the move by listing the names of the moved files on the console.

Local Switches:

mm-dd-yy/A	Move any file created this date or after. Arguments mm (month) and dd (day) may be one or two digits.
mm-dd-yy/B	Move any file created before this date.
name/N	Do not move files that match name.
oldname/S newname	Assign newname to the preceding file (but retain its oldname in the current directory).

Template Characters:

Permitted.

Examples:

```
MOVE/V DP1 TEXT-.-)
```

Copy all nonpermanent files in the current directory which begin with the letters TEXT to directory DP1. Verify moved files on the console.

```
MOVE/D/K/V SOURCE -.SR PROG-.-/N)
```

Copy all nonpermanent files in the current directory (except link entries and files beginning with PROG) to directory SOURCE. Verify moved files on the console, and delete the original files after the move.

```
DIR ACCTSDUE)
R
MOVE/A/V/R DZ0:LEGALNOTES -.AD 1-2-77/B)
ADRIANCO.AD
JOHNSMART.AD
.
.
R
```

This example assumes that bills are filed in directory ACCTSDUE. As each bill file is created in ACCTSDUE, it receives the extension .AD (account due). The MOVE command moves all bill files created before January 2, 1977, into directory LEGALNOTES on unit DZ0, and verifies moved files on the console.

NSPEED

Invoke the NOVA Supereditor

Format:

NSPEED *[filename]*

Edit ASCII text on a NOVA computer. Superedit features multibuffer editing, multiple I/O files, macroprogramming, and numeric variables. If the *filename* does not exist, Superedit creates it; if it does exist, Superedit opens it. For ECLIPSEs, see SPEED.

To use Superedit from another directory, you must link to both of its files; these are NSPEED.SV and SPEED.ER.

For more information, see the *Superedit User's Manual.*

Switches:

None.

Example:

```
NSPEED FILEX)
CREATING NEW FILE
!
.
.
.
!UE$$
!H$$


R
```

Superedit creates and opens FILEX.
! is the Superedit prompt.

Enter editing commands

Update the file.
The H ESC ESC command terminates Superedit and returns control to the CLI.

OEDIT

Edit octal locations in a disk file

Format:

OEDIT filename

Invoke the octal editor to examine and modify in octal, decimal, or ASCII any location in any disk file. The symbolic editor (see SEDIT) offers slightly more versatility than OEDIT. For details on OEDIT, see the *Octal Editor User's Manual.*

Switches:

None.

Extensions:

The octal editor searches for whatever filename and extension are given.

Examples:

OEDIT FOO.SV)	If OEDIT finds FOO.SV, it opens it, then types a period (.) prompt.
.	
.	
.14/ *001672*	Proceed with editing.
.	
.	
.	
$Z	To return to the CLI, type ESC Z (echoed as $Z).
R	

You can find the current NMAX requirement for any save file by issuing the OEDIT command:

404-16/ *nnnnnn*

and find the ZMAX value by entering:

401-16/ ======

The values *nnnnnn* and ====== are returned by OEDIT, and indicate *filename's* current NMAX and ZMAX requirements.

OVLDR

Create an overlay replacement file (RDOS)

Format:

OVLDR savefilename old-ovly-descrip/N↑
new-ovly-name(s)↑)
old-ovly-descrip/N new ovlyname(s)

This command creates and loads an overlay replacement file for the overlays of a program which was loaded (RLDR) with overlays. It can replace up to 127 overlays. The new overlays will not replace the old ones until you issue the REPLACE command.

old-ovly-descrip describes the overlay which will be replaced by new-ovly-name(s). This can be either the octal representation of the overlay segment and overlay number within the segment, or it can be the symbolic overlay name - if the name was assigned with the .ENTO pseudo-op in the root program.

See Chapter 4 of the *RDOS Reference Manual* or the *Extended Relocatable Loaders Manual* for more on overlays.

OVLDR requires a symbol table in savefilename; RLDR will do this only if you specify the global/D switch, or include a .EXTN .SYM. in a program module.

Global Switches:

/A	Produce an additional symbol table listing with symbols ordered alphabetically. You must also append the local /L switch.
/E	Display error messages on the console when a listing file has been specified (local/L). By default, when you specify a listing file, error messages to the console are suppressed.
/H	Print all numeric output in hexadecimal (radix 16). By default, output is in octal.

Local Switches:

name/E	Send error messages to file name.
name/L	List the symbol table on file name. The table lists symbols in numeric order.
old-ovly/N	Old overlay descriptor precedes this switch, and new overlay relocatable binary name(s) follow.

Examples:

```
OVLDR/A/E ROOT OLD0/N NEW0 NEW1↑)
  ROOT.OM/L)
```

This command line creates overlay replacement file ROOT.OR. When ROOT.OR replaces the original overlay file, ROOT.OL, overlays NEW0 and NEW1 will replace old overlay, OLD0, in ROOT's overlay file. The .ENTO pseudo-op was used in each overlay; thus we could reference overlays by name instead of by overlay number and node number. OVLDR's error messages and memory map of new symbols will be written to disk file ROOT.OM. Error messages will also go to the console.

The RLDR line which loaded ROOT might have looked something like this:

```
RLDR/D ROOT [OLD0,OLD1] ↑)
ROOT1 [OLD2,OLD3])
```

PATCH

Install patches in a save or overlay file

Format:

PATCH *[savefilename/S] [patchfilename/P]* ↑)
[loadmap-filename/L]

The PATCH utility installs patches which you inserted in a patchfile with the ENPAT utility. To apply Data General-supplied patches to an operating system, you must have instructed SYSGEN to save a load map file. If no load map exists for a program, you can patch it by appending the global /N switch to PATCH (although you won't be able to use symbols in the patchfile). If you omit arguments, PATCH will ask for them.

When PATCH runs, it creates a patch dialog file named savefilename.PD, deleting any file of the same name first. This file records patch date, time, and locations for the last patch of savefilename. Patches applied during the last run of PATCH are marked with an asterisk (*) in the dialog file. You can TYPE this file at will.

Patch error messages are explained in Appendix A.

Global Switches:

/I	Do not display comments from the patchfile on the console.
/N	There is no load map available.

Local Switches:

name/L	name is the load map filename. You must include the extension to this name.
name/P	name is the patchfilename, including extension (created by ENPAT).
name/S	Apply the patch file to program name. You can omit the .SV extension.

PATCH (continued)

Examples:

```
PATCH SYS/S SYS.LM/L IPB/P)
3 APPLICABLE PATCH(ES)
3 PATCH(ES) NEEDED TO BE INSTALLED
R
```

This command installs the patches entered in the ENPAT example. Note that PATCH describes the number of applicable patches, and the number it installs.

```
PATCH/N MYPROG/S MYPROG.P1/P)
2 APPLICABLE PATCH(ES)
2 PATCH(ES) NEEDED TO BE INSTALLED
R
```

Here, the patches inserted in patchfile MYPROG.P1 are applied to user program MYPROG. There is no load map, hence the global /N switch. (Eventually, for permanence, MYPROG's author should make corrections to the source version, then reassemble and reload it.)

POP

Return to the next higher level program

Format:

```
POP
```

When a user program has swapped in the CLI on level two or below, POP directs the system to resume execution of the user program. Programs can swap in the CLI via system call .EXEC. See Appendix D for an example, or .EXEC in your system reference manual for details on .EXEC.

The CLI is initially on level zero; when you execute another program via the CLI, the CLI is swapped out of memory and the new program is brought in and executed on level one. The new program swaps itself out, and the CLI back by issuing .RTN (or .ERTN). POP effectively issues .RTN from the CLI.

You cannot POP from the level zero CLI.

Switches:

None.

Examples:

XCLI.SV, a user program running at level one, wishes to suspend its operation temporarily so that it can ask the CLI to perform some routine maintenance function. Then XCLI.SV wishes to resume its own operation. Thus a return (via CTRL-A or any other means) to the level zero CLI would be inadequate, since XCLI could not resume; it would have to begin again. To use the CLI temporarily, XCLI swaps in the CLI on level two via system call .EXEC. After using the CLI at level two to perform whatever functions were needed, the operator issues the POP command. This restores XCLI in main memory; it continues from the point of interruption. Appendix D shows an assembled version of XCLI.

PRINT

Print an ASCII file on the line printer

Format:

PRINT filename$_1$ *[...filename$_n$]*

Copy the contents of ASCII filename(s) on the line printer (SLPT). PRINT is the equivalent of a series of XFER/A filename $LPT commands. To print a binary file, use FPRINT.

The filename(s) may be on any device. If the system detects a parity error on paper tape, it prints a left slash (\) in place of the bad character, and displays the message PARITY ERROR on the console; printing continues.

Switches:

None.

Examples:

PRINT FOO.SR DP0F:COM.SR EXT.SR)

Print source files FOO.SR, COM.SR on DP0F, and EXT.SR on the line printer.

PRINT ARREARS:<JAN,FEB,MAR>.BL)

Print files JAN.BL, FEB.BL, and MAR.BL in directory ARREARS.

PRINT DP1:MYFILE/2)

Print MYFILE, which is on the primary partition of DP1, twice.

PUNCH

Copy an ASCII file on the paper tape punch

Format:

PUNCH filename$_1$ *[...filename$_n$]*

Copy the contents of ASCII filename(s) to paper tape on the high-speed punch ($PTP). PUNCH is the equivalent of a series of XFER/A filename $PTP commands. To punch a binary file, use BPUNCH.

The filename(s) may be on any device. If the system detects a parity error on paper tape, it punches a left slash (\) in place of the bad character, and displays the message PARITY ERROR on the console; punching continues.

Switches:

None.

Example:

PUNCH DP0:ALPHA.SR BETA.SR)

Punch files ALPHA.SR and BETA.SR on the high-speed punch.

RDOSSORT

Invoke the RDOS Sort/Merge Program (RDOS)

Format:

RDOSSORT inputfilename$_1$ *[inputfilename$_n$]*
[outputfilename/O] key$_1$ *[key$_n$] [arguments]*

Use the RDOSSORT command to invoke the Sort/Merge Program, which can rearrange, delete, and/or combine disk or tape files. The *RDOS Sort/Merge Manual* covers this utility. You can run Sort/Merge from the console or under BATCH, on a mapped machine, or in the background of an unmapped machine. The Sort function reads records from an input file (inputfilename) and sorts them according to the key specifications, switches and arguments you specify in the command line. If you want a sorted output file, this input file must be disk-resident.

In Merge mode, the program reads records from up to six disk or tape input files and produces a single output file. Use the global /M switch to select the Merge function.

In the format, inputfilename is the name of a disk or tape file. If you omit *outputfilename/O,* there will be no output file; this is useful if you want only a key file or listing. The data in inputfilename will be sorted according to the key you specify; up to eight keys are permitted. These keys make up the Control Word, which will be compared to the field in each input record. You specify each key as: b.f, where b is the starting character number, and f is the character field length. For example, 7.10 specifies a 10-character key in character positions 7-16 of the record.

In the *arguments* portion of the command line, you can specify the following parameters:

record size - The program assumes that the input records are 80 characters (bytes) long, the length of a normal console line. If your records are not 80 characters, enter their decimal bytelength followed by the local /R switch.

input file collating order - By default, records are collated in ascending ASCII sequence (by ascending byte number). You can reverse this by appending the global /D switch, or specify your own collating order by appending local /S to the filename which describes your sequence.

input file field delimiters - To specify a lower limit, append local /B to the file containing the lower limit for the major key field; for an upper limit, append /U to the filename containing the upper limit. If you omit either delimiter, all records input will be output.

output fields - You can specify from one through eight output fields, which will determine the format of records in the output file. You specify an output field as you do a key, except that a colon replaces the period (b:f). If you enter no output field specifier, each record will be output in the same form as you input it.

output files - You specify the listing file with local /L; default is the console. Unless you specify a file for sorted keys with /K, no key file will be produced.

work files - During a sort operation, the Sort/Merge program uses up to six work files. It creates these in the current directory, and names them *SORTWn.TP,* where *n* is 1 through 6. In many sorts, work file 1 is most active, followed by 4, 2, 5, 3, and 6. If you want greater sorting efficiency, you can arrange the work, input, and output files on different devices, according to priority. Use the local /W switch to specify an alternate work file. See the *Sort/Merge User's Manual* for more information.

Global Switches:

/D	Sort records in decending ASCII order (default is ascending).
/M	Merge record (the default operation is Sort).
/N	Do not list sort or merge statistics (by default, Sort/Merge lists statistics).

Local Switches:

name/B	File name contains the lower limit field.
name/D	In Sort mode, delete inputfile name after sorting it. The system ignores this switch if you specify no output file.
name/K	After sorting keys, write them to file name.
name/L	List sort output on file name (overrides global /N).
name/O	name is the sorted or merged output file.

n/R	Decimal number n is the input record size (in bytes) (default is 80).
name/S	File name specifies the collating sequence (default is ascending ASCII).
name/U	File name contains the upper limit field.
name/W	File name is a user-defined work file. You can specify up to six work files.

Example:

```
RDOSSORT MEMBERS.DA DGSORT/O↑)
COLLAT/S 61.12 141.10 SLPT/L 180/R↑)
1:30 61:60 121:30 BOTLIMIT/B TOPLIMIT/U)
```

The summary of statistics for this command might be:

```
RDOS Sort/Merge          06:43:07  07/02/76
Program                  :- Sort mode
Input Filename(s)        :- MEMBERS.DA
Output Filename          :- DGSORT
Record Size (Bytes) :- 180
Collating Sequence       :- User Specified
Sequence Filename        :- COLLAT
Sorted Key Filename      :- None Specified
Lower Limit Filename     :- BOTLIMIT
Upper Limit Filename     :- TOPLIMIT
Input Field Specifiers   :-  Start  Byte/Length
                             (Bytes)
                             61 -12
                             141 - 10
Output Field Specifiers  :- Start Byte/Length
                             (Bytes)
                             1-30
                             61-60
                             121-30
Input File Records       :- 3458
Sort In Record Count     :- 3458
Sort Out Record Count    :- 366
```

RELEASE

Release a directory or tape drive from the system

Format:

RELEASE { tape drive / directory }

Logically remove a tape drive or directory from the system. When you RELEASE a mag tape unit, the tape is automatically rewound. After release, a directory or device is closed to access until you initialize it with an INIT or DIR command. Be sure to release a disk before physically removing it. Releasing a directory releases all subordinate directories.

To shut down the system, release the master directory; this releases all initialized devices in the system, including tape transports. After you have released the master directory, the message MASTER DEVICE RELEASED will appear on the console. If a foreground program is running in RDOS, you must type CTRL-F on the background console before releasing the master device - you cannot release it from the foreground.

Switches:

None.

Examples:

```
RELEASE DP1)
```

Release DP1 and all its directories and permit the disk to be removed from drive 1. DP1 was not the master device.

```
RELEASE MT0)
```

Rewind and release MT0.

```
RELEASE %MDIR%)
MASTER DEVICE RELEASED
```

Shut down the system. The CLI variable %MDIR% contains the name of the master dirctory.

RENAME
Rename a file

Format:

RENAME oldname$_1$ newname$_1$ *[...oldname$_n$ newname$_n$]*

The system allows you to rename any file which has not been protected with the P attribute; this includes the system utilities. Some utilities use a special COM.CM file (Appendix D), and will not work if you rename them. No save file can execute without the .SV extension.

Switches:

None.

Examples:

```
DELETE Q.SV)
R
RENAME QTEST.SV Q.SV)
R
```

Replace the old version of Q.SV with a new version, which was previously named QTEST.SV. (QTEST.SV was a debugged version of Q.SV.)

```
RENAME DP0:A1.DR DP0:A.DR B1 B)
```

Rename directory A1 to A on DP0; rename file B1 to B in the current directory.

```
XFER/A $TTI A)
LONGFILENMR
          ↑— Press CTRL-Z here
RENAME T @A@.01 T1 @A@.02 T2 @A@.03)
R
```

This shows an easy way to assign the same name string to existing files. The @ signs specify the *contents* of file A: the string LONGFILENM. The old names T, T1, and T2 become LONGFILENM.01, LONGFILENM.02, and LONGFILENM.03.

REPLACE
Replace overlays in an overlay file (RDOS)

Format:

REPLACE savefilename

REPLACE is the active sequel to the OVLDR command. It replaces overlays in an overlay file named savefilename.OL with the overlays you specified in the OVLDR command. Actual replacement will occur as soon as there are no outstanding overlay load requests.

Switches:

None.

Example:

```
REPLACE ROOT)
```

Replace the specified overlays in file ROOT.OL with new overlays in file ROOT.OR. The OVLDR command created the overlay replacement file ROOT.OR.

REV

Display the revision level of a save file

Format:

REV savefilename *[.SV]*

savefilename must have the "S" attribute. The system returns the major revision number followed by a period and a minor revision number. Both major and minor revision levels can be in the range 0 through 99.

Use the .REV assembler pseudo-op to assign a major and minor revision level number to a source file. If you omit this pseudo-op from all source programs, then the revision level of the save file will be displayed as "00.00".

Certain compilers (e.g., FORT) assign their own revision level to the binary file; it is then carried over to the save file.

Switches:

None.

Example:

```
REV CLI)
CLI.SV 06.20
```

The major revision level of this CLI is 06, and the minor revision level is 20.

RLDR

Load relocatable binaries (RBs) to produce an executable save file

Format:

RLDR { binary$_1$ ... *[library]*... }
{ root binary$_1$ ... *[overlay-binary$_1$ ...] [library...]* }

Invoke the Relocatable Loader utility to load relocatable binary files or libraries and produce a save file. By default, the save file receives the name of the first binary in the command line, with the extension .SV. If you specify overlays, the RLDR creates an overlay file for the save file, and gives it the same name, with the extension .OL.

Consult the *Extended Relocatable Loaders Manual* for details on RDLR.

Use the root-binary format to create an overlay file for your program. RLDR places each root-binary in the .SV file, and each *overlay-binary* in the .OL file. During execution, all root-binaries will remain in memory all of the time, while the *overlay-binaries* will be called from the .OL file into their assigned memory nodes, according to the instructions in your program.

Each pair of square brackets ([ ]) defines a node in the .SV file and a segment in the .OL file. (Square brackets are *not* notation conventions in RLDR command formats.) Nothing resides in the node of the .SV file; RLDR simply reserves an area of memory to receive an overlay from the node's corresponding .OL file segment. Each comma within square brackets delimits an overlay. The individual overlays will reside independently in the .OL file, yet they will be contiguous to one another in the order given in the RLDR command line. Overlays assigned to the same node (placed within the same pair of brackets) will overwrite their node as the executing program loads (.OVLODs) them one-by-one. For example, the command line RLDR ABLE [A, B] loads binary ABLE as a root program, ABLE.SV; ABLE has one node for binaries A or B. The node in the disk file will maintain itself in memory when ROOT executes; ROOT can load either A or B into this node without affecting the rest of itself. For more on overlays, consult the Loaders manual and Chapter 4 of your system reference manual.

To load a program, RLDR uses its own save and overlay files (RLDR.SV and RLDR.OL) and the system library (SYS.LB). All these files are normally available in the master directory. To use RLDR from another directory, you must create links to each of these files from the directory.

RLDR (continued)

Note: Because the system library (SYS.LB) differs for each type of system (e.g., unmapped NOVA and mapped NOVA), a program loaded under one type of system will probably not execute under another type of system. To load for a different kind of system, you must obtain the proper system library for the target system and ensure that RLDR searches it, not the current library, during the load. You can do this by loading from a (sub)directory which holds links to RLDR and the target library.

If you will want to debug the program, use the global /D switch; this includes both a debugger and program symbol table in the program save file. To include local user symbols, also use local /U (you must have specified *global* /U to the assembler or compiler).

RLDR can also produce a load map (memory map), which shows where program modules will reside when the program executes. This load map can help you patch the save file on disk; you can print it or save it on disk with the local /L switch.

If a program will run more than one task, you must specify multiple tasks, and the number of I/O channels for the program, to RLDR. RLDR will then include the multitask scheduler in the program. To specify tasks and channels, use either a .COMM TASK statement in your program, or RLDR local switches /C and /K. If you're working in an advanced language (as opposed to assembly language), your compiler manual will tell you how to specify multiple channels/tasks.

Global Switches:

/A Produce an additional load map, with the symbols ordered alphabetically (you must also include local /L).

/B Use short TCBs in the save file (effective in unmapped NOVA multitask programs only).

/C Load for compatibility with RTOS. Start user NREL code at 440_8 and the save file at 0 (as with local /Z switch); do not search SYS.LB unless its name is included.

/D Include the symbolic debugger and program symbol table with the program. To load the interrupt-disable debugger, include its name (xIDEB.RB) somewhere in the command line. The program symbol table will not be written to the *save* file unless you include the /D switch. Use global /S with /D to store the symbol table in high memory (instead of directly above the program).

/E Display error messages on the console when a listing file has been specified (local /L). By default, when you specify a listing file, error messages go to it, not to the console.

/G When overlays reference named common, print a warning message at each occurrence. By default, RLDR prints a warning message at the first occurrence only.

/H Print numeric output in hexadecimal (radix 16). By default, numbers are printed in octal.

/I Do not create UST, TCB, or other system tables; start NREL code at 445_8 and ZREL code at 50_8. The save file cannot execute under any Data General operating system.

/K Store RLDR's internal symbol table for this program in file binary.ST. RLDR does not save this file unless you include /K.

/M Do not display load map or error messages on the console.

/N Do not search the system library (SYS.LB). By default, RLDR searches the system library at the end of the command line to try and satisfy undefined symbols.

/O Do not include program symbol table in the save file, even though the debugger is included (used with global /D).

/P Print the starting NREL value of each RB as it is loaded.

/S Leave the symbol table in high memory (you must also include /D).

/X Used during SYSGEN to specify system overlays.

/Y — See Global X.

/Z — Load the save file to start execution at location 0. Be careful with the global /Z switch; the resulting save file cannot execute under RDOS or DOS. Use this switch to load stand-alone programs which use page zero locations 0-15_8. You can then process the save file via MKABS/Z to produce an absolute binary that can execute in stand-alone mode, via the binary loader.

Local Switches:

n/C — Allot n I/O channels to the program. This octal value overrides any value specified in a .COMM TASK statement. If you omit both /C and .COMM TASK, RLDR allots eight channels to the program.

name/E — Send error messages to file name.

n/F — Start program's NREL address space at octal address n, for execution in an unmapped RDOS foreground. You must also include local /Z.

n/K — Allot octal n tasks to the program. This overrides any value in a .COMM TASK statement. By default, the RLDR allots one task.

name/L — Send the load map to file name. This map will list symbols in numeric order. It won't be saved unless you include this switch.

n/N — Force the NREL pointer, which indicates the location for loading the next file, to octal address n. Address n must exceed the current NREL pointer value. (The pointer value is originally 400_8 plus space for a User Status Table, TCBs, etc.; RLDR moves it upward as it loads each binary.)

name/S — Give the program file name, with the .SV extension, and the overlay file name, with the .OL extension.

name/U — Include user symbols from binary file name in the symbol table. Local symbols are those used exclusively within this binary. This works only if you specified local /U to the assembler.

[binaries]/V — Load binaries (delimited by brackets) as virtual overlays (mapped RDOS only). All virtual overlays must precede other overlays in the load command line; e.g., RLDR ROOT [R1,R2]/V ROOT1 [R3,R4].

n/Z — Start program's ZREL address space at octal address n. By default, ZREL code starts at location 50_8.

Extensions:

The CLI searches for each binary with the RB extension; if it does not find it, it searches for binary. You must include library name extensions, if any.

Examples:

```
RLDR A B C DP4:D)
```

Load files A, B, and C from the current directory, and file D from DP4; produce save file A.SV in the current directory. All messages go to the console. Neither the symbol table or the load map are saved.

```
RLDR/D A B C)
```

Load files A, B, and C with the symbolic debugger, to produce file A.SV.

```
RLDR/E MYFILE MYFILE.LM/L)
```

Load MYFILE to produce MYFILE.SV; create listing file MYFILE.LM and send the load map and all messages to it. Additionally, send error messages to the console.

```
RLDR MYPROG PROG1 MYLIB.LB $LPT/L 10/K 20/C)
```

Load MYPROG, PROG1, and library MYLIB.LB to produce MYPROG.SV. All RLDR messages (including the load map) go to the line printer. MYPROG.SV is a multitask program; it can run 10_8 tasks and use 20_8 channels if it wants.

```
RLDR R0 [A,B,C,D] R1 R2 [E,F G,H] R0.LM/L)
```

Load binaries R0, R1, and R2 into R0.SV; create overlay file R0.OL with two segments. R0.SV can load overlays A, B, C, or D into the first node (node 0); it can load overlays E, F and G, or H into the second node (node 1). The load map and console messages go to file R0.LM.

SAVE
Rename the break file

Format:

SAVE filename

Rename file BREAK.SV (FBREAK.SV for the foreground) to filename.SV. SAVE is commonly used to save the core image of a program interrupted by a CTRL-C break or by the debugger $V command. If filename already exists in the current directory, the system deletes it. You cannot precede filename with a directory specifier. For more information on BREAK.SV, see Keyboard Interrupts in Chapter 3 of your system reference manual.

Switches:

None.

Examples:

DEB ALPHA)	Enter debugger to correct location PP
PP/ *LDA 2 0* LDA 2 @0 3)	
$V	Exit from debugger
BREAK	
R	
SAVE ALPHA)	Save core image file as ALPHA.SV.

SDAY
Set today's date

Format:

SDAY month day year

Set the system calendar. You can enter the year as either two or four digits (e.g., 77 or 1977). Use spaces or commas to separate the date arguments.

Switches:

None.

Example:

SDAY 10 17 1977)

Set the system calendar to October 17, 1977.

SEDIT

Analyze or edit a file on disk

Format:

SEDIT filename

Invoke the symbolic editor to examine, analyze, or modify the contents of a disk file. The filename can be any nonsequential disk file. If you omit an extension, SEDIT searches for filename.SV; if does not find it, it searches for filename. To specify an overlay file, include the .OL extension. For SEDIT commands, consult the *Symbolic Editor User's Manual.*

Global Switches:

/N	Do not search for the symbol table. Use this switch if there is no symbol table, or to edit a text or nonsave file.
/Z	This file starts at location 0.

Examples:

```
SEDIT MYPROG.SV)
SEDIT REV x.x                    SEDIT finds and opens
                                 MYPROG.SV.
MYPROG%)                         Proceed with editing.
.START+10/ 106413 )
.$;)
.START+10/ SUB # 0 1 SNC )       Examine a location.
.
.
.$Z                              Return to the CLI by
                                 typing ESC Z.
DONE
R
```

You can find the current NMAX and ZMAX requirements for any program by typing the SEDIT commands:

```
404/ nnnnnn
401/ zzzzzz
```

SEDIT returns values *nnnnnn* and *zzzzzz*, which are the User Status Table values for NMAX and ZMAX.

SMEM

Set the memory size of each ground (mapped RDOS)

Format:

SMEM background

Set the amount of memory which will be available to the background program. The foreground will receive the remainder. When you bootstrap RDOS, it gives all available memory to the background. For the background, enter the decimal number of 2048-byte blocks (pages) you want.

You can issue SMEM only in a mapped system from the background CLI, while no foreground program is running.

Switches:

None.

Examples:

```
SMEM 24)
```

Allocate 24 2048-byte blocks of memory to the background, and the remaining blocks to the foreground.

```
SMEM 48)
```

Allocate 48 memory blocks to the background, and all remaining user memory to the foreground.

SPDIS
Disable device spooling (RDOS)

Format:

SPDIS device *[...device$_n$]*

Disable spooling on device. RDOS automatically spools data sent to any user device defined as spoolable, and to all the following devices:

$DPO, $LPT, $LPT1, $PLT, $PLT1, $PTP, $PTP1, $TTP, $TTP1, and the teletypewriter versions of $TTO and $TTO1.

During a spool operation, RDOS queues data sent to output devices in disk buffers; this frees the CPU for further processing while the devices receive the queued data. If a spool operation requires more disk space than is available, the system itself will issue the equivalent of an SPDIS command.

After you type the SPDIS command, the system will stop spooling data to the disabled device's disk buffer. Output will continue until all data waiting in the buffer has been processed.

Switches:

None.

Example:

```
SPDIS $LPT)
```

This command prevents data output to the line printer from being spooled. To reinstitute spooling, you must issue the command SPEBL $LPT. If output is currently being spooled to the line printer, spooling will stop after the current spool is completed.

SPEBL
Enable device spooling (RDOS)

Format:

SPEBL device$_1$ *[...device$_n$]*

Re-enable spooling on a device for which spooling has been disabled. The device may be any user device defined as spoolable, or any of the following:

$DPO, $LPT, $LTP1, $PLT, $PLT1, $PTP, $PTP1, $TTP, $TTP1, and the teletypewriter versions of $TTO and $TTO1.

RDOS does not automatically spool to the plotter ($PLT, $PLT1), thus you must explicitly enable spooling to these devices.

Switches:

None.

Example:

```
SPEBL $LPT)
```

Re-enable spooling to the line printer.

SPEED
Invoke the ECLIPSE Supereditor

Format:

SPEED *[filename]*

Edit ASCII text on an ECLIPSE computer. Superedit features multibuffer editing, multiple I/O files, macroprogramming, and numeric variables. If the *filename* does not exist, Superedit creates it; if it does exist, Superedit opens it. For NOVAs, see NSPEED.

To use Superedit from another directory, you must link to both of its files; these are SPEED.SV and SPEED.ER.

For more information, see the *Superedit User's Manual.*

Switches:

None.

Examples:

```
SPEED FILEX)
CREATING NEW FILE        Superedit creates and
                         opens FILEX.
!                        ! is the Superedit prompt.
.
.                        Enter editing commands
.
!UE$$                    Update the file.
!H$$                     The H ESC ESC
                         command terminates
                         Superedit and returns
                         control to the CLI.
R
```

SPKILL
Delete the spool queue (RDOS)

Format:

SPKILL device$_1$ *[...device$_n$]*

Stop a spool operation by deleting the spool files enqueued to device. The device may be any user devices defined as spoolable or any of the following:

$DPO, $LPT, $LPT1, $PLT, $PLT1, $PTP, $PTP1, $TTP, $TTP1, or the teletypewriter versions of $TTO and $TTO1.

After you kill spooling on a device, data on the output spool is lost. Spooling will resume to the device at the next command which sends data to the device; e.g., PRINT MYFILE).

Switches:

None.

Example:

```
SPKILL $LPT)
```

This command stops the spooling of data to the first line printer.

STOD
Set the time

Format:

STOD *[hour] [minute] [second]*

Set the system clock (which is a 24-hour clock). Use spaces or commas to separate the time arguments.

Switches:

None.

Example:

```
STOD 21 24 0)
```

Set the system clock to 9:24:00 p.m.

SYSGEN
Generate a new RDOS or DOS system

The SYSGEN command word varies with the kind of computer you have, as follows:

Machine	Command Word
ECLIPSE with INFOS	ISYSGEN
Other ECLIPSE	BSYSGEN
Mapped NOVA 3	NSYSGEN
Other NOVA or microNOVA	SYSGEN

The SYSGEN command generates a new operating system. All SYSGEN questions will be displayed on the console.

You will find the entire RDOS SYSGEN procedure in the manual *How to Load and Generate Your RDOS System.* This manual also explains the RDOS tuning feature. To generate a DOS system, see the *DOS Reference Manual.*

Format:

SYSGEN *[newsysname/S] [dialogfile/V] [loadmapfile/L]*

SYSGEN *[newsysname/S] [olddialogfile/A]* ↑)
[newdialogfile/V] [tuningfile/T] [loadmapfile/L]

Use form one of this command to generate a brand new RDOS or DOS system, or a different version of an existing system. In the first format, you choose the *newsysname* for the system and append the /S switch to this name. Do not use the name of the current system as *newsysname.* (If you omit a name, the system will assign the default name of SYS000.SV.) The filename you specify for *dialogfile* (by using the /V switch) will hold the SYSGEN dialog. The load map, which you may use for system diagnosis or to patch your system, will be output to the console unless you specify a *loadmapfile* with the /L switch.

Use the second format only when you want to tune an existing RDOS system. If you specified tuning during the original SYSGEN, you can use the system's tuning file to generate a more efficient version of the original system. Be sure to turn tuning off (TUOFF)) before generating a tuned system from the current system. Never generate a system which has the same name as a system with a tuning file.

The new version will be saved under *newsysname.SV*. *Olddialogfile* is the dialog filename of the system you want to use as a basis for tuning. The /A switch directs the current system to generate the copy from *olddialogfile*. *Newdialogfile* is your choice of a name for the new dialog file. *Tuningfile* is the name of the SYSGENed tuning file *(newsysname.TU)*; you must append the /T switch. Again, the load map will be displayed on the console unless you specify a *loadmapfile*.

After you answer the last SYSGEN question, the SYSGEN program analyzes your answers to its questions and then places the names of required modules and libraries for this system in CLI file CLI.CM (FCLI.CM for a foreground CLI). Then it processes CLI.CM with RLDR. If you use the global /N switch, SYSGEN will skip the RLDR step, and leave the RLDR command line in (F)CLI.CM. You can invoke the RLDR phase once, before executing another SYSGEN, with the command @CLI.CM@).

Global Switch:

/N	After SYSGEN do not build the new system with RLDR (see paragraphs above).

Local Switches:

name/A	Generate a system from dialog file name.
name/L	Send the load map to file name. (Default is the console).
name/S	Give the finished system this name.
name/T	Use tuning file name (RDOS). SYSGEN will use this file to enter more efficient responses, thus overriding the original, or supplementing your current SYSGEN responses.
name/V	Save the SYSGEN dialog for the system being generated in file name.

Examples:

```
SYSGEN SYS1.<SV/S SG/V LM/L>)
```

This command activates the system generation program, which allows you to generate a new operating system by answering a series of questions. The new system will be named SYS1 and will reside in files SYS1.SV and SYS1.OL. The dialog file will be SYS1.SG. The /N switch is not specified, therefore SYS1 will be loaded automatically; load errors and the load map will go to disk file SYS.LM.

The following example shows RDOS system SYSXX, which was generated to include tuning, and has had tuning turned on for awhile.

```
SYSGEN SYS2/S SYSXX.SG/A↑)
  SYSXX.TU/T SYS2.SG/V $LPT/L)
```

Generate a tuned system, SYS2, from the SYSXX.SG dialog file. The tuning file provides the most efficient answers from its experience with SYSXX. The tuned version is named SYS2 and is stored as SYS2.SV/SYS2.OL. The new dialog file is named SYS2.SG; the load map and load messages go the the line printer.

TPRINT
Print the tuning file (RDOS)

Format:

TPRINT *[sysname]*

Print the tuning file for sysname.TU (you need not enter the *sysname*).

The tuning file is always in the directory which holds the current RDOS system; the TUON command creates it. It contains use information on system stacks, cells, buffers and overlays. For each of these, the information includes: the number in the system, the number of requests, number of faults, and the percentage of faults. A fault is a request for the item which had to be delayed because the item was not immediately available.

If you specified an overlay report at SYSGEN, you can include the global /O switch to print an overlay frequency report on each system overlay. The tuning chapter of the *RDOS Reference Manual* describes the function of each overlay.

NOTE: This command is meaningless unless (a) tuning was requested at SYSGEN, and (b) tuning was turned on (TUON) for some time during system operation.

Global Switches:

/L Print on the line printer.

/O Print an overlay frequency report.

Example:

```
TPRINT/L SYSXX)
```

Print the tuning file for the system SYSXX on the line printer.

TUOFF
Stop recording in the tuning file (RDOS)

Format:

TUOFF

Stop recording use data concerning buffers, cells, stacks, and overlays in the tuning file. TUOFF does not delete the contents of the tuning file.

Switches:

None.

Example:

```
TUOFF)
```

Terminate recording in the tuning file.

TUON

Start recording in the tuning file (RDOS)

Format:

TUON

Use TUON to turn the tuning function on. Assuming that the current operating system was generated with the tuning feature, it will start recording use data concerning stacks, cells, buffers, and overlays. For each of these, the report will include: the number in the system, total requests, total faults, and the percentage faulted.

If you specified an overlay report at SYSGEN, RDOS will also start compiling an overlay frequency report. The first time you issue TUON for any tunable RDOS system, the system creates the tuning file and names it sysname.TU. Tuning data is recorded in this file whenever sysname is running until you issue the TUOFF command. Be sure to type TUOFF) before generating a tuned system from the current system's tuning file.

Switches:

None.

Example:

```
TUON FIRSTSYS)
```

Initiate recording in the tuning file.

TYPE

Display a file on the console

Format:

TYPE filename$_1$ *[...filename$_n$]*

Copy an ASCII file or files on the console. This command resembles a series of XFER/A filename $TTO commands. To type a binary file, use FPRINT You can suspend output by typing CTRL-S, and resume it by typing CTRL-Q. This can help you read long files if your console is a CRT.

Under some systems, the TYPE command types only 80 characters per line on the console. If you have a DASHER, or any other 132-column console, and want full-page width, use the XFER command.

The source files may come from any device. When the system detects a character with bad parity on paper tape, it types a left slash (\) and displays the message PARITY ERROR; typing continues.

Switches:

None.

Example:

```
TYPE A.SR B.SR DP1:XX.SR)
```

Display disk files A.SR and B.SR in the current directory, and source file XX.SR in DP1, on the console.

UNLINK
Remove a link entry

Format:

UNLINK linkname$_1$ *[...linkname$_n$]*

Unlink files by deleting one or more link entry names. This does not affect the resolution file.

Global Switches:

/C — Confirm each removal. The system repeats each link name on the console, and waits for you to confirm the deletion by typing a carriage return. To prevent the deletion or unlinking, press any key other than RETURN.

/L — List the deleted link names on $LPT (overrides /V).

/V — Verify each link entry deleted.

Template Characters:

Permitted only when the filename argument is in the current directory.

Example:

```
UNLINK/C TEST.-)
```

Request a confirmation before deleting each link entry:

```
TEST.SR)*      Remove link TEST.SR;
TEST.RB)*      Remove link TEST.RB;
TEST.SV%       Don't remove link TEST.SV.
R
```

When you confirm a deletion with a carriage return, the system echoes with an asterisk (*). Otherwise, it echoes nothing.

VFU
Edit a format control file for a data channel line printer (RDOS)

Format:

VFU *[switches]* vfu-filename

The VFU utility allows you to create and edit format control files for data channel line printers; it can also load format control files into the printer's memory. You can specify three settings for the printer forms in each vfu-filename: tab stops, form size in lines, and multiple line-number/channel-number pairs.

VFU has four functions:

1) It creates files (always appending the .VF extension to your specified filename).

2) It displays files.

3) It edits existing .VF files.

4) It loads an existing .VF file into the printer's memory.

It can also enable and disable access to the printer's memory by user programs.

You specify each VFU function with a global switch. You must specify an enable or disable command alone; otherwise, you can enter multiple commands with multiple switches. In a multiple command line, the Create and Edit commands are processed first, followed by load commands, then display commands. You need not enter the .VF extension to access VFU files.

Create a .VF File (VFU/C)

When you type VFU/C filename, the VFU program announces itself, then displays the new filename, and asks:

1. TAB CONTROL:
   WANT STANDARD TABS (EVERY 8 COLUMNS)?
   ENTER Y/N = = >

   If you want standard tab stops (at column 0, 7, 15, ...127), type Y ; VFU then skips to question 3. To set your own tabs, answer N ; VFU then asks:

2. ENTER COLUMN NO. (1-132) OR CR = = >

   Enter each column number at which you want a tab stop in this file; press) after typing each number. VFU will repeat this question until you type only); it then asks:

3. VFU CONTROL:
   WANT STANDARD (11 INCH)?
   ENTER Y/N ==>

   The answers you give to quesions 3. and 4. cannot be changed in this file. A "standard" form is 11 inches (66 lines) long, has channel one set for line 1, and channel 12 set for line 63. If you want standard control for the forms which you will print using this file, type Y ; VFU then skips to question 5. To specify a different form length, type N ; VFU then asks:

4. ENTER FORM SIZE IN LINES (1-143) ==>

   Enter the number of vertical lines in the forms which you will print using this file.

   Next, VFU asks:

5. ENTER LINE NUMBER OR CR ==>

   Specify a vertical line in which you want to set a channel "hole", then type). VFU asks:

6. ENTER CHANNEL NO. (1-12) ==>

   Specify the channel number which you want to associate with the line number you gave in the last question. VFU then asks questions 5 and 6 again; it continues asking these questions until you type) in response to 5.

VFU now creates vfu-filename.VF and returns to the CLI.

Display a File (VFU/L for first line printer, VFU/V for console)

Global Switches

/S — Display settings on the second line printer (e.g., VFU/L/S).

Local Switches

name/L — Send settings to file name (overrides global /L).

/V — Display vfu-filename's settings on the console.

VFU displays the tab-stop and VFU channel settings or sends them to the specified file. It shows line-number/channel settings in the form "l-c", where l is the line number of a "hole" and c is the channel number of a "hole". If you defaulted creation question 3, thus specifying 11-inch forms for a file, VFU channels would be shown as:

```
1-1 63-12
```

Edit a File (VFU/E)

If you specify a display option with /E (i.e., VFU/V/E), the vfu-filename will be displayed. VFU then asks about tab control:

1. TAB CONTROL:
   ENTER COLUMN NO. (1-132) OR CR ==>

   Specify a column number where you wish to set a new tab or clear an old tab, and type). Then, VFU asks:

2. ENTER SET(S) OR CLEAR(C) ==>

   To set a tab at the column number specified in step 1., type S; to clear a tab at this column, type C. VFU now asks question 1. again; it continues this loop until you type) in response to 1.

   Now, VFU asks vertical format questions:

3. VFU CONTROL:
   ENTER LINE NUMBER OR CR ==>

   Respond with the line number of a channel "hole" which you wish to set or clear, and). VFU then asks about the channel number associated with this line number:

4. ENTER CHANNEL NO. (1-12) ==>

   Enter the channel number which you want to associate with the line number (for Set) or which is already associated with the line number (for Clear). Now, VFU asks:

5. ENTER SET(S) OR CLEAR(C) ==>

   To set a new channel "hole", type S; to clear an existing "hole", type C. VFU then repeats questions 3, 4, and 5 until you respond with) to 3.

VFU now updates the existing vfu-filename with the new setting, and displays the new settings on the display file you specified (with /V, etc.).

Load a File into the Printer's Memory (VFU or VFU/X for $LPT, VFU/S for $LPT1)

For the first data channel line printer, type VFU vfu-filename; for the second DCH printer, type VFU/S vfu-filename. VFU then displays a prompt message. When you strike a key and the printer is ready, VFU kills spooling and XFERs the vfu file into the printer's memory. You can then print files on the forms which need the format control contained in the vfu file.

Access Control (VFU/A and VFU/D)

User programs can access the printer's memory directly to change tab and VFU settings after you type VFU/A or VFU/A/S for the second printer. The printer memory is unprotected until someone loads a VFU file or disables access with VFU/D (VFU/D/S).

Examples:

```
VFU/C PAYROLL1)
DATA CHANNEL LINE PRINTER FORMAT
CONTROL PROGRAM
CREATING PAYROLL1.VF
TAB CONTROL:
WANT STANDARD TABS (EVERY 8 COLUMNS) ?
ENTER Y/N ==> N
ENTER COLUMN NO. (1-132) OR CR 3)
ENTER COLUMN NO. (1-132) OR CR 9)
ENTER COLUMN NO. (1-132) OR CR 16)
ENTER COLUMN NO. (1-132) OR CR 28)
ENTER COLUMN NO. (1-132) OR CR 50)
ENTER COLUMN NO. (1-132) OR CR )
VFU CONTROL:
WANT STANDARD (11 INCH) ?
ENTER Y/N ==> N
ENTER FORM SIZE IN LINES (1-143) ==> 44)
ENTER LINE NUMBER OR CR ==> 1)
ENTER CHANNEL NO. (1-12) ==> 1)
ENTER LINE NUMBER OR CR ==> 4)
ENTER CHANNEL NO. (1-12) ==> 2)
ENTER LINE NUMBER OR CR ==> 9)
ENTER CHANNEL NO. (1-12) ==> 3)
ENTER LINE NUMBER OR CR ==> 41)
ENTER CHANNEL NO. (1-12) ==> 12)
ENTER LINE NUMBER OR CR ==> )
R
```

This sequence created VFU file PAYROLL1.VF. The next sequence shows a combined display/edit and loading of this file.

```
VFU/V/E PAYROLL1)
DATA CHANNEL LINE PRINTER FORMAT
CONTROL PROGRAM
PAYROLL1.VF      06/15/77 14:22:16
TAB STOPS:
3, 9, 16, 28, 50
VFU CHANNELS:
1-1, 4-2, 9-3, 41-12

EDITING PAYROLL1.VF
TAB CONTROL:
ENTER COLUMN NO. (0-132) OR CR ==> 8)
ENTER SET(S) OR CLEAR(C) ==> S)
ENTER COLUMN NO. (0-132) OR CR ==> 9)
ENTER SET(S) OR CLEAR(C) ==> C)
ENTER COLUMN NO. (0-132) OR CR ==> )
VFU CONTROL:
ENTER LINE NUMBER OR CR ==> 14)
ENTER CHANNEL NO. (1-2) ==> 4)
ENTER LINE NUMBER OR CR ==> )
PAYROLL1.VF      06/15/77 14:28:25
TAB STOPS:
3, 8, 16, 28, 50
VFU CHANNELS:
1-1, 4-2, 9-3, 14-4, 41-12
R
VFU PAYROLL1)

DATA CHANNEL LINE PRINTER FORMAT
CONTROL PROGRAM
PREPARE TO LOAD PAYROLL1.VF
WAIT UNTIL OUTPUT TO THE PRINTER HAS
COMPLETED. MAKE SURE PRINTER IS READY
AND ON-LINE.
STRIKE ANY KEY WHEN READY.
```

Someone now strikes a key.

```
R
```

The next sequence enables, then disables, access to the printer's memory by the program INVOICEAPR.SV.

```
R
VFU/A)
DATA CHANNEL LINE PRINTER FORMAT
CONTROL PROGRAM
ENABLING ACCESS TO PRINTER CONTROL
MEMORY
R
INVOICEAPR)
.
.
R
VFU/D)
DATA CHANNEL LINE PRINTER FORMAT
CONTROL PROGRAM
DISABLING ACCESS TO PRINTER CONTROL
MEMORY
R
```

XFER

Copy the contents of a file to another file

Format:

XFER sourcefile destinationfile

Copy a file on any device to another file on any device. The sourcefile and destinationfile can be any disk filename or device, as shown in Table 2-2 or 2-3. Each can include directory specifiers, but cannot *be* a directory. When you XFER a file to disk, XFER creates the destinationfile name and copies the contents of sourcefile into it. The old file's name, attributes, creation date, etc., are not copied.

In RDOS, if you omit switches, XFER organizes the destination file sequentially. RDOS cannot execute sequential files, so, if you XFER a save file to disk, be sure to append the local /R switch to the destination filename. In DOS, if you omit switches, XFER organizes the desination file randomly. In either system, be sure to CHATR a destination save file +S after the transfer.

During the transfer, the system will detect parity errors in all ASCII source files, and in binary files on magnetic or cassette tapes. On a parity error, it displays a PARITY ERROR message. For a paper tape file, it displays a backslash (\) for each bad character. For mag tape or cassette files, it aborts the command after detecting a parity error.

You can use XFER to copy text directly into a file via the form:

XFER/A $TTI filename *[.MC]*)

Type in the CLI commands you want, then terminate the transfer with CTRL-Z. See Chapter 3 for examples.

Global Switches:

/A — This is an ASCII transfer. Transfer the file line by line, taking appropriate read/write action, such as inserting line feeds after carriage returns (if this pertains to the destination file).

/B — Append the source file to the end of the destination file. You must use this to XFER to an existing disk or tape file.

Local Switches:

destinationfile/C — Organize the destination file contiguously (both files must be disk files).

destinationfile/R — Organize the destination file randomly (see comments above).

Examples:

```
XFER $PTR Q)
LOAD $PTR, STRIKE ANY KEY
```

Create disk file Q; then wait for sequence to load the reader and strike a key. Then copy the tape in the reader to file Q.

```
XFER $PTR $PTP)
```

Copy the tape in the paper tape reader to the paper tape punch.

```
XFER MYPROG.SV DP1:MYPROG.SV/R)
```

Copy ABLE.SV from the current directory to file 0 of the tape on drive MT0. The copy receives no name on the tape. This example continues:

```
XFER ABLE.SV MT0:0)
```

Copy ABLE.SV from the current directory to file 0 of the tape on drive MT0. The copy receives no name on the tape. This example continues:

```
XFER MT0:0 ABLE.SV/R
FILE ALREADY EXISTS: ABLE.SV
R
```

Normal filename rules apply when you XFER a file to a disk directory.

```
XFER/A/B QTY:7 DP4:MUXNOTES)
```

Append the text input from ALM or QTY line 7 to a disk file in DP4 named MUXNOTES. The user on QTY:7 can terminate input with CTRL-Z.

End of Chapter

Appendix A
Error Messages

This appendix describes all the error messages that you may receive while operating the CLI. Some error messages can result only from misuse of RDOS or DOS systems or task calls in a user program; the explanation of these is preceded by the letters "OS". Normally, these represent the CLI's attempt to describe the error when you have specified .ERTN as an error return from a system or task call. If you receive such an error message, check the descriptions of the questionable call in your system reference manual.

Most utility program error messages aren't covered here. If you receive an error message from a utility, consult the appropriate utility manual. Most Data General utility programs take the error return to the CLI when they cannot proceed; the CLI then attempts to interpret the error code passed in AC2. Generally, the CLI's explanation of the problem is accurate, but the argument it gives can be misleading. See "Error Handling" in Chapter 2 for more on this.

If you have an INFOS file system, consult the INFOS manual for an explanation of INFOS error messages.

Occasionally, you may receive irrelevant error messages. These can be caused by hardware problems, among other things. If they recur, run the appropriate diagnostic program on your equipment.

Certain serious error conditions can halt the system in exceptional status. These are described in the Exceptional Status Appendix of your system reference manual.

Table A-1. Error Messages

Message	Meaning/Action
A ZERO .XMT OR .IXMT MESSAGE	OS. Attempt to transmit a zero word message.
ADDRESS ERROR IN .SYSTM ARGUMENT	OS. A program tried to reference an address outside address space (mapped RDOS only).
ALM LINE NOT READY	OS. The ALM line modem is not ready.
ATTEMPT TO CREATE A ZERO LENGTH CONTIGUOUS FILE	Via CCONT or system calls .CCONT or .CONN.
ATTEMPT TO READ INTO SYSTEM SPACE	A program tried to read into an RDOS/DOS reserved address (unmapped system only).
ATTEMPT TO RELEASE AN OPEN DEVICE	OS. Attempt to release a tape unit with an open file.
ATTEMPT TO RESTORE A NON-EXISTENT IMAGE	You issued the POP command from a level 0 CLI.

Table A-1. Error Messages (continued)

Message	Meaning/Action
ATTEMPT TO WRITE AN EXISTENT FILE	OS. Your program attempted to write an existing file.
BLANK TAPE	OS. A program issued an .INIT or .MTOPD call to a drive which contained a new tape. Type INIT/F to initialize new tapes.
CHANNEL ALREADY IN USE	OS. The I/O channel specified is in use by another file.
CHANNEL CLOSED BY ANOTHER TASK	OS. Two tasks share an I/O channel, and one of them closed the channel before the other could complete its I/O.
CHECKSUM ERROR	Detected during input.
COMMON SIZE ERROR	OS. Communications area (.ICMN) size too small.
COMMON USAGE ERROR	OS. Other program defined no communications area.
CONSOLE INTERRUPT RECEIVED	OS. On a read from or write to a multiplexed line, a console interrupt occurred.
CANNOT CHECKPOINT CURRENT BG	OS. The background program is not checkpointable (.EXBG). Causes can be: multiplexed I/O, user interrupt routine, outstanding .RDOPR/.WROPR, or BG is already checkpointed. See .EXBG in your system manual for other causes.
DEVICE ALREADY IN SYSTEM	The device or directory has already been initialized.
DEVICE NOT IN SYSTEM	You tried to access an uninitialized directory or device; INIT it (or DIR to it).
DEVICE PREVIOUSLY OPENED	OS. Attempt to open a mag tape or cassette that is already open.
DEVICE TIMEOUT	OS. 10-second disk timeout occurred on a nonmaster device. Try inserting a disk, and putting the drive on-line.
DIRECT I/O ACCESS ONLY	OS. A program tried to perform nondirect-block I/O on a file which requires direct-block I/O.
DIRECTORY DEPTH EXCEEDED	You tried to create a directory within an equivalent directory (e.g., a subdirectory while in a subdirectory); or you attempted to LOAD (FLOAD) a dump file containing a directory into an equivalent directory.

Table A-1. Error Messages (continued)

Message	Meaning/Action
DIRECTORY IN USE	You tried to: 1) release, delete, or rename a directory which has an open file; 2) delete or rename a directory or device which has been initialized; or 3) initialize a directory which the other CPU is using, in a dual processor system under IPB. For 1), check file use counts in the directory (LIST/U), and CLEAR/A/V/D if necessary.
DIRECTORY NOT INITIALIZED	You tried to access an uninitialized directory or device.
DIRECTORY SHARED	This message occurs when a directory used by both grounds is released by one ground. It is merely a warning.
DIRECTORY SIZE INSUFFICIENT	You (or a program) tried to create a partition with fewer than 48 (60_8) disk blocks.
DISK FORMAT ERROR	This can occur when you INIT a disk. It can often be simply a warning, after which the system does initialize the disk, or it can indicate a fatal format error. If possible, DUMP the disk and run DKINIT.SV (DOSINIT.SV) or a formatter program on it.
DUPLICATE READ OR DUPLICATE WRITE	OS. Multiplexed lines. Read from a line from which a program is reading, or write to a line which a program is writing.
END OF FILE	End of file detected (often on tape, when you try to access a file beyond the logical end-of-tape). Try a lower file number.
ERROR IN USER TASK QUEUE TABLE	OS. Illegal information in task queue table (.QTSK).
ERROR: CAN'T MAKE SENSE OUT OF THIS LOAD MAP (PATCH command)	The load map you specified to PATCH has a symbol whose name exceeds six characters, or a premature end-of-file was detected.
ERROR: CONTENTS OF PATCH LOCATION DOES NOT MATCH NEW OR CURRENT LOCATION SPECIFIED (PATCH)	The contents you specified in ENPAT must match the original contents of the disk location. Check the location with a disk editor.
ERROR: FIRST CHARACTER IN PATCH FILE LINE IS NOT A VALID CHAR	Create another patchfile, using ENPAT.
ERROR: INDIRECTION ERROR - ADDRESS OUT OF RANGE (PATCH)	The patchfile specified a negative address.

Table A-1. Error Messages (continued)

Message	Meaning/Action
ERROR: INSUFFICIENT MEMORY TO HOLD CORE RESIDENT LOAD MAP (PATCH)	PATCH stores the load map above NMAX, below HMA, and uses four words for each symbol. You may need to generate a smaller system to PATCH.
ERROR: argument ALREADY SPECIFIED (PATCH)	You specified the save file, load map, or patch file twice in the PATCH command line.
ERROR: UNABLE TO EVALUATE xxxx FIELD (PATCH)	Create another patchfile, using ENPAT (ENPAT checks for valid location fields).
EXEC ERROR ON CHAIN: LOG FILE IF ANY CLOSED	Fatal CLI runtime error. The CLI was unable to execute a CHAIN command. Attempting to CHAIN closes all files. The CLI will restart.
FATAL OUTPUT ERROR	Fatal CLI runtime error, encountered while the CLI was trying to write to a channel. The CLI will restart.
FATAL SYSTEM UTILITY ERROR	An unrecoverable error was detected within a utility. This can occur if the utility required a support file (e.g., MACXR.SV) and couldn't find it on disk.
FATAL UTILITY ERROR	OS. A program has passed ERFUE to the CLI via .ERTN.
FILE ALREADY EXISTS	You tried to create a file whose name exists in the current directory; or you tried to move or load such a file into the current directory without appropriate switches.
FILE ATTRIBUTE PROTECTED	You tried a forbidden operation on a file whose attributes were fixed via the system .CHATR call.
FILE DATA ERROR	File read error. If on tape, this can mean that the tape heads need cleaning.
FILE DOES NOT EXIST	The file does not exist in the current or specified directory.
FILE IN USE	You tried to delete, dump, rename, or alter a file which is in use. Check its use count with LIST/U filename; then CLEAR it if necessary.
FILE NOT OPEN	OS. The file has not been opened.
FILE POSITION ERROR	OS. A program tried to set an illegal pointer position in a file (.SPOS).
FILE READ PROTECTED	Its attributes forbid reading.
FILE SPACE EXHAUSTED	Out of disk space in the current partition or diskette; on mag tape - EOT reached while writing.

Table A-1. Error Messages (continued)

Message	Meaning/Action
FILE WRITE PROTECTED	Its attributes prevent it from being modified.
FILES MUST EXIST IN THE SAME DIRECTORY	The files specified are in different directories.
FOREGROUND ALREADY RUNNING	A foreground program is running, and you tried to: 1) change BG/FG memory allotments; or 2) CLEAR file use counts to 0; or 3) execute a program in the foreground.
ILLEGAL ARGUMENT	An illegal character in an argument.
ILLEGAL ATTRIBUTE	You specified an undefined attribute.
ILLEGAL BLOCK TYPE	You tried to LOAD a file which is not in DUMP format or to FLOAD a file which is not in FDUMP format, or to XFER a dumped file. Use appropriate command.
ILLEGAL CHANNEL NUMBER	OS. A system call specified a channel number greater than 377_8.
ILLEGAL COMMAND FOR DEVICE	OS. A program tried to perform illegal I/O (e.g., free-form I/O on disk data).
ILLEGAL DIRECTORY NAME	Illegal character in directory name or specifier.
ILLEGAL FILE NAME	The legal characters are A-Z, 0-9, and $.
ILLEGAL INDIRECT FILE NAME	Unmatched @ in command line.
ILLEGAL NUMERIC ARGUMENT	Nonnumeric character in numeric argument, numeric argument too large, or incorrect base.
ILLEGAL OVERLAY NUMBER	OS. The specified overlay does not exist.
ILLEGAL PARTITION VALUE	Results from SMEM command, in mapped system only. 1) You tried to allocate more memory to BG than is available to both grounds; or 2) BG memory allotment is too small for BG CLI; or 3) You typed SMEM from any level other than 0.
ILLEGAL SYSTEM COMMAND	You typed a command which is not supported in the current environment (e.g., SMEM in an unmapped system).
ILLEGAL VARIABLE	Variable undefined, or unmatched % in command line.
ILLEGAL TEXT ARGUMENT	Unmatched quotes (") in message command.

Table A-1. Error Messages (continued)

Message	Meaning/Action
INSUFFICIENT CONTIGUOUS BLOCKS	Not enough contiguous blocks are available to create your desired contiguous file or partition, or, the current directory lacks space for the contiguous file you tried to LOAD (FLOAD) or MOVE into it. Try retyping the command line with overlay (.OL) filenames before other filenames. (Overlay files are contiguous.)
INSUFFICIENT MEMORY TO EXECUTE PROGRAM	The program requires more memory than is available in the background (or mapped foreground after EXFG or .EXFG).
INSUFFICIENT ROOM IN DATA CHANNEL MAP	OS. Improper channel size specified in .IDEF call.
INVALID BAD BLOCK TABLE	If you can INIT the disk, DUMP it and run DKINIT.SV on it; if you can't INIT it, run the DKINIT PARTIAL command on it.
INVALID TIME OR DATE	Attempt to set an illegal time or date.
LINE TOO LONG	Command line limit (132 characters) exceeded on console input or read/write I/O.
LINK ACCESS NOT ALLOWED	Attempt to access a linked file whose attributes forbid linking (CHLAT).
LINK DEPTH EXCEEDED	You tried to LINK to a resolution file via more than nine intermediate links, or you attempted to reference a resolution file via more than ten intermediate link entries.
LOG FILE ERROR	Fatal CLI runtime error, while CLI was writing to the log file. The CLI will attempt to close the log file and restart.
MAP.DR ERROR	File system MAP.DR inconsistency detected. This is serious. Try to dump the disk, then fully initialize (INIT/F) it.
MCA REQUEST OUTSTANDING	1) OS. The MCA transmitter is trying to transmit, and the receiver has issued no receive request. 2) A program tried to transmit on an MCA channel on which either the receiver or transmitter was waiting for the other to send or receive.
MT OP ERR n (FDUMP, FLOAD)	For FDUMP, write ring may be on reel; or you may not have stacked dumps properly by incrementing file numbers by three. A valid stack example is: DIR MYDIR;FDUMP MT0:0;DIR YOURDIR;FDUMP MT0:3; DIR HERDIR;FDUMP MT0:6, etc. For FLOAD, you may not have specified stacked file numbers properly, incrementing by three. You'd FLOAD the example above as: DIR MYDIR;FLOAD MT0:0;DIR YOURDIR;FLOAD MT0:3; DIR HERDIR;FLOAD MT0:6, etc.

Table A-1. Error Messages (continued)

Message	Meaning/Action
MTn NOT READY - MAKE IT READY! (FDUMP)	The tape drive specified is not ON LINE.
NO DEBUG ADDRESS	You issued the DEB command for a file which was not loaded with a Debugger (RLDR global /D switch).
NO DIRECT I/O	OS. File not accessible by direct-block I/O.
NO FILES MATCH SPECIFIER	No match on template characters (- and *).
NO MCA REQUEST OUTSTANDING	No outstanding receive request by an MCA device.
NO MORE DCBS	You tried to initialize too many directories/devices at any given moment. Release one, or generate a new system which allows more to be INITed at one time.
NO ROOM FOR UFTS	Not enough channels defined for program before assembly (.COMM TASK) or loading (/C switch in RLDR). This can also occur if you try to execute a stand-alone program from the CLI. Use the BOOT command.
NO SOURCE FILE SPECIFIED	Your utility command requires a source file.
NO STARTING ADDRESS	You tried to execute a program which does not specify a starting address. See the next error message.
NO STARTING ADDRESS FOR LOAD MODULE	RLDR command. The user program was written without a starting address, and cannot be executed. Specify a START address after the last .END pseudo-op.
NO SUCH DIRECTORY	The specified directory hasn't been initialized, or isn't where you specified, or doesn't exist.
NOT A COMMAND	CLI error, resulting from modification of CLI.SV or CLI.OL.
NOT A LINK ENTRY	You tried to UNLINK a file lacking the link characteristic.
NOT A SAVE FILE	The file must be a program, and it requires the save and random attributes. XFER it with the local /R switch, and/or CHATR it +S.
NOT ENOUGH ARGUMENTS	Your command requires more arguments.
name OPEN ERR - FILE NOT DUMPED (FDUMP)	Filename is in use (open), hence could not be dumped. The dump continues.
OPERATOR MESSAGES NOT SYSGENED	OS. Your program contains OPCOM or .RDOPR/.WROPR calls, yet the operator message feature wasn't selected at system generation.

Table A-1. Error Messages (continued)

Message	Meaning/Action
OUT OF TCB'S	OS. All TCBs are in use. Process the program with RLDR again, and specify more tasks with the /K switch.
PARITY ERROR	Parity error on read or write sequential of mag tape or cassette. Possibly the tape heads need cleaning.
PERMANENT FILE	You tried to delete a file which has the P attribute (CHATR or .CHATR).
PHASE ERROR	OS. Attempt to reference a location with less than the minimum valid value.
PROGRAM NOT SWAPPABLE	OS. The program given as an argument to .EXEC cannot be swapped into memory.
PUSH DEPTH EXCEEDED	OS. Attempt to .EXEC from level four.
QTY ERROR	Simultaneous read or write to same multiplexor line.
RDOS ERROR	This means that a system utility or user program took the error return through .ERTN, but the CLI could not interpret the value returned in AC2. (The value in AC2 was not defined to the CLI.) This can happen when any program lacks disk space to create the desired files, or if you are using incompatible revisions of system utility programs (e.g., if the Macroassembler and CLI are of different revision levels). If disk space is the problem, use the DISK command, provide more disk space or execute the command in another directory; if incompatibility is the problem, use the REV command and try a different rev of the utility, CLI, or system.
READ FRAMING ERROR	OS, multiplexed lines. A hardware framing error occurred.
READ OVERRUN ERROR	OS, multiplexed lines. A hardware overrun error occurred.
name READ-LOCKED - NOT DUMPED	FDUMP filename has the R (read-protected) attribute. Dumping continues.
SIGNAL TO BUSY ADDRESS	OS. Message address is already in use.
SPOOL FILES ACTIVE	Attempt to release the master directory while it is spooling data to output devices. Either wait, or kill spooling to devices with SPKILL.
STACK OVERFLOW	Fatal CLI runtime error, which sometimes occurs when the CLI tries to execute a very long command line. The CLI will restart.

Table A-1. Error Messages (continued)

Message	Meaning/Action
SYNTAX ERROR:ILLEGAL NESTING OF < > AND ()	The command line contained a parenthesis-angle bracket pair.
SYNTAX ERROR:ILLEGAL NESTING OF () AND []	Parentheses aren't allowed inside brackets (RLDR overlay definition).
SYNTAX ERROR INSIDE []	Illegal character or sequence inside brackets.
SYNTAX ERROR:UNMATCHED OR NESTED []	Brackets must be matched, but not nested.
SYNTAX ERROR:UNMATCHED OR NESTED ()	Parentheses must match, and can't be nested.
SYNTAX ERROR: "<" WITHOUT ">" OR ">" WITHOUT "<"	Unmatched angle brackets.
SYS.DR ERROR	Inconsistency in file system SYS.DR detected. Dump the disk if possible; then fully initialize (INIT/F) it.
SYS ERR RTN OFFSET nnnnnn	FDUMP, FLOAD. RELEASE the drive, and try the command again.
SYSTEM DEADLOCK	The system has run out of buffers. Wait until a task or program has finished its I/O.
SYSTEM STACK OVERFLOW	System stack capacity has been reached and exceeded. Wait a minute or so, then rebootstrap the system.
TAPE HAS WRONG REEL NO.	FLOAD. You are loading an FDUMPed reel out of sequence. Mount the correct reel and strike a key. This can also happen if you FLOAD stacked dump files with the wrong file numbers; see error message MT OP ERR n.
TASK ID ERROR	OS. Task ID not valid in current environment.
TASK NOT FOUND FOR ABORT	OS. A program tried to abort a nonexistent task.
TEXT ARGUMENT TOO LONG	The CLI encountered more than 72 characters between quotes ('').
TOO MANY ARGUMENTS	You entered too many arguments to the command.
TOO MANY SOFT ERRORS	The DOS soft error count has reached its maximum for this diskette. Copy the diskette with DOSINIT.
TOO MANY LEVELS OF INDIRECT	You've established too many levels of indirect files (@filename@).

Table A-1. Error Messages (concluded)

Message	Meaning/Action
TOVLD NOT LOADED FOR QUEUED OVERLAY TASKS	OS. The multitask overlay loader has not been loaded from the system library with your program, as required by your environment. Insert a .EXTN .TOVLD in the source program, then re-assemble and reload it.
TRANSMISSION TERMINATED BY RECEIVER	The MCA transmission was terminated prematurely because its length exceeded that requested by the receiver.
UNIT IMPROPERLY SELECTED	Mag or cassette tape or disk drive not turned on, or not ON LINE.
VIRTUAL BUFFER FILE ERROR	Fatal CLI runtime error, relating to a problem in the CLI buffer files CLI.T(0,1,2,3,4), CLI.S(0,1,2,3,4) or CLI.C(0,1,2,3,4). The CLI will restart and try to solve the problem.
YOU CAN'T DO THAT	You attempted something grossly invalid, like typing ENDLOG without giving the password specified in LOG, or running an ECLIPSE program on a NOVA.

End of Appendix

Appendix C
Dump File Format

The DUMP command copies file data from an input file to an output file; while copying each file, it formats data in the outputfile into eight different kinds of blocks. When you LOAD the file, the data in these blocks is placed in the file directory's SYS.DR. Most of the tapes you received from Data General are in DUMP format (others are in XFER format). The block types described here apply to DUMP, not FDUMP.

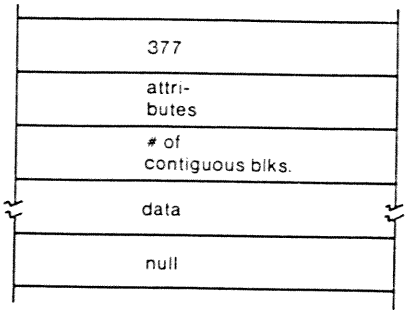
The output file produced by DUMP is called a dump file, although it often consists of many disk files. The system assigns an end block to each partition, subdirectory and DOS directory in a dumpfile, and to the last data file in the dumpfile. Each dump file is formatted into the following types of blocks:

Types	Block Identifier Byte
1. Name blocks	377
2. Data blocks	376
3. Error blocks	375
4. End blocks	374
5. Time blocks	373
6. Link Data blocks	372
7. Link Access Attribute blocks	371
8. End of Segment blocks (paper tape)	370

Every dumpfile begins with a name block and ends with an end block. Each subdirectory in the dumpfile also ends with an end block (e.g., in the dumpfile produced by DUMP MT0:0 SUBDIR.DR -.SR), both the end of the contents of SUBDIR and the last -.SR file would be followed by an end block). An RDOS secondary partition does *not* end with an end block in the dumpfile, but an end block is provided for it at the end of the dumpfile. For example, assume that secondary partition 2NDPART contains one subdirectory. The dumpfile producd by the command DUMP MT0:0 2NDPART.DR -.SR) would have three end blocks -- one at the end of the subdirectory in 2NDPART, and two at the end of the dumpfile.

These formats are given in the following illustrations:

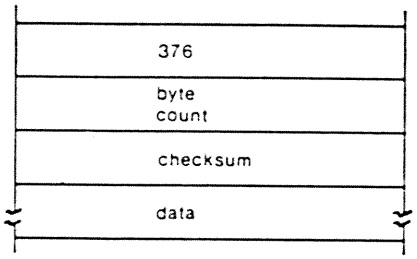
NAME BLOCK



size	contents
1 byte	block type identifier.
2 bytes	attributes.
2 bytes	no. of contiguous blocks only if contiguous characteristic set.
variable	filename. (A period is added to each filename without an extension.)

NOTE: The filename is written by means of system call .WRS, so if reading from paper tape, you must use system call .RDS to avoid parity problems.

DATA BLOCK

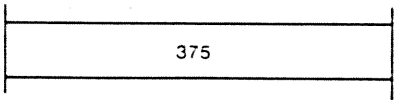


Size	Contents
1 byte	block type identifier.
2 bytes	byte count.
2 bytes	checksum (word count modulo 2 + total contents of all full words).
n bytes	data (file contents).

no terminator

NOTE: Any odd byte is written in "data" and is added in "byte count" but it is not summed. There is no end-around carry.

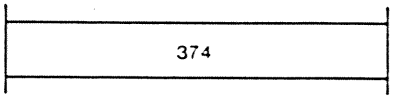
ERROR BLOCK



Size	Contents
1 byte	block type identifier.

NOTE: This block causes the file you are currently loading to be read through to the end but not to be written out.

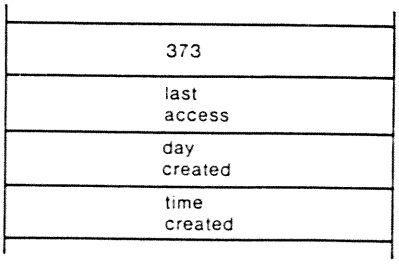
END BLOCK



Size	Contents
1 byte	block type identifier.

TIME BLOCK

(follows name block except on links)

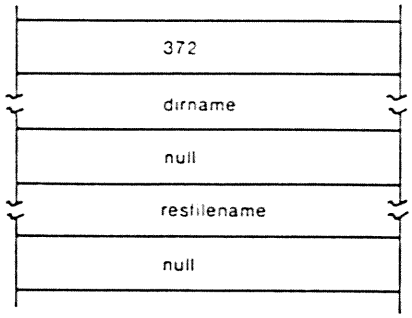


Size	Contents
1 byte	block type identifier.
2 bytes	Julian day last accessed (opened) since Jan. 1, 1968 (day 1).
2 bytes	Julian day created.
2 bytes	left byte contains hour, right byte contains minute created.

no terminator

LINK DATA BLOCK

(If the file is a link entry, this follows the name block.)

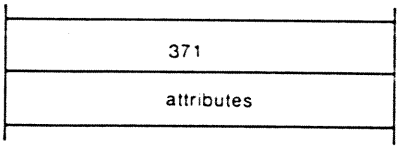


Size	Contents
1 byte	block type identifier.
variable	alternate directory name (if any).
variable	resolution filename terminated by null.

(2 nulls if link has neither alternate directory nor alias.)

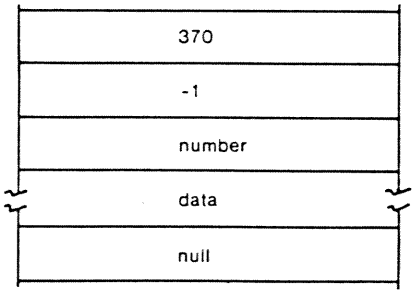
NOTE: These 2 names are written via system call .WRL, so if reading from paper tape, use .RDL to avoid parity problems.

LINK ACCESS ATTRIBUTES BLOCK



Size	Contents
1 byte	block type identifier
2 bytes	link access attributes

END OF SEGMENT BLOCK (PAPER TAPE)



Size	Contents
1 byte	block type identifier.
2 bytes	-1.
1 byte	segment number.
variable	filename.

End of Appendix

Appendix D
Extended Uses of the CLI

This appendix describes the CLI's mechanism for handling keyboard commands and explains how you can use it to evoke the CLI from a currently-executing program.

Although the CLI's primary function is to serve as an interface to the operating system and your computer, you can also use it to execute commands from within an assembly-language program, and to pass arguments to your program.

Before you can do either of these things, you must understand the special files which the CLI uses to communicate with user and utility programs. One file, called CLI.CM, exists specifically to hold certain unedited commands for the CLI. The CLI stores the RLDR command line for SYSGEN in CLI.CM. A second file, COM.CM, is built by the CLI to hold your edited command line and switches. If the command line invokes a utility program, the utility examines COM.CM; if the command line invokes a user program, the program itself can examine COM.CM. Passing instructions to a program via COM.CM is described in the second section of this appendix.

If you are running a CLI in the foreground of a mapped system, the foreground CLI filenames (and all other special CLI files) begin with an F. COM.CM, for example, is FCOM.CM for the foreground CLI. In all other respects, the files are identical to those in the background.

Swapping and Chaining to the CLI

You can swap or chain to the CLI (filename CLI.SV) via the system call .EXEC. If you .EXEC the CLI with AC2 not equal to 0, the CLI will search for the special file CLI.CM, interpret the command string that it reads from CLI.CM, and execute these commands exactly as if they had been input from the console. These commands won't be noted in the log file. When the CLI has executed all commands, it will display its R prompt. You can then return to your program by entering the POP command from the console.

If you choose to return to the next higher-level program without console intervention, you can insert POP into CLI.CM as the last command to be executed.

Note that when you .EXEC the CLI in this way, and it detects an irrecoverable error in CLI.CM (like an error inside @ signs), it takes the error return from .EXEC to the program with error code 16_8 (ERFUE= "FATAL UTILITY ERROR") in AC2. Depending on your needs, you might want to have the program check for error ERFUE and take appropriate action. (The CLI will not take the error return from background level 0; instead it will go to the console for input.)

Returning with Error Status to the CLI

When a program executed from the CLI returns to the CLI via the system call .ERTN (return from a swap with error status), the CLI examines the contents of AC2. If AC2 contains mnemonic EREXQ (code 17_8), the CLI will search for CLI.CM and execute its contents as described above. If AC2 contains any other error code, the CLI will interpret the error and display it on the console (unless AC2 contains error code ERNUL, 20_8 , in which case the CLI displays nothing).

```
FILE DOES NOT EXIST: MYPROG.SV
```

would indicate that error ERDLE had occurred somehow in program MYPROG.SV and that MYPROG had taken the error return and reported the error to the CLI via the .ERTN system call.

The following figure shows the assembled listing of a program which stores commands in CLI.CM, invokes the CLI on level two, and then POPs itself back into execution.

```
                        .TITL XCLI
                        .NREL
        000001          .TXTM 1

    00000'020430 START: LDA 0, CFILE    ;Pointer to CLI.CM.
    00001'006017        .SYSTM          ;System,
    00002'014003        .OPEN 3         ;open CLI.CM on channel 3.
    00003'000422        JMP ER          ;Mandatory error return
                                        ;location.
    00004'020431        LDA 0, CMANDS   ;Pointer to CLI commands.
    00005'024442        LDA 1, BCOUNT   ;Bytecount of commands,
                                        ;in COM.CM, for .WRS.
    00006'006017        .SYSTM          ;System,
    00007'016403        .WRS 3          ;Write commands to CLI.CM
                                        ;on channel 3. .WRS permits
                                        ;multiline commands.
    00010'000415        JMP ER          ;Required location.
    00011'006017        .SYSTM
    00012'014403        .CLOSE 3        ;Close channel to update file.
    00013'000412        JMP ER          ;Required.
    00014'020434        LDA 0, CLISV    ;Pointer to CLI.SV.
    00015'126400        SUB 1, 1        ;Clear AC1 for swap.
    00016'152520        SUBZL 2, 2      ;Pass nonzero value in AC2.
    00017'006017        .SYSTM          ;Swap in the CLI on level 2.
    00020'003400        .EXEC           ;CLI.CM's POP command will bring
                                        ;this program back on level 1.
    00021'000404        JMP ER          ;Mandatory.
    00022'006017        .SYSTM          ;OK- Return to
    00023'004400        .RTN            ;level 0 CLI.
    00024'000401        JMP .+1         ;Reserved, never taken.

;Take error return, let the CLI report error status:

    00025'006017 ER:    .SYSTM
    00026'006400        .ERTN
    00027'000401        JMP .           ;Error return is never taken.

 ;Other labels:

    00030'000062"CFILE: .+1*2
    00031'041514        .TXT "CLI.CM"   ;Command file CLI.CM.
          044456
          041515
          000000

    00035'000074"CMANDS: .+1*2
    00036'042111        .TXT "DISK;LIST/N;POP<15>" ;Commands for CLI.CM.
          051513
          035514
          044523
          052057
          047073
          050117
          050015
          000000
    00047'000024 BCOUNT: (BCOUNT-CMANDS)*2 ;Number of bytes to write
                                           ;to CLI.CM.
    00050'000122"CLISV: .+1*2
    00051'041514        .TXT "CLI.SV"   ;CLI name, for swap.
          044456
          051526
          000000

                        .END START
```

Figure D-1. Program Which Uses CLI.CM

CLI Reserved Files

In addition to (F)CLI.CM and (F)COM.CM, the CLI uses the following files to perform. You may never want to use these files, but be aware of their names and don't try to create files with identical names.

CLI.SV, CLI.OL, CLI.ER. The CLI save, overlay, and error interpretation files. You can examine the CLI.ER (including ASCII error messages) with the FPRINT/Z command.

(F)CLI.Tn (F)CLI.Cn (F)CLI.Sn These are the CLI virtual buffer files. n corresponds to the system level on which the CLI is operating; it can be 0, 1, 2, 3, or 4.

(F)TML.TM. CLI creates this file for the alphabetical sort required by a LIST/S command. It is deleted after the sort.

Using the CLI's Command File (COM.CM)

When the CLI reads a command line, and does not recognize the first word as a simple command, it assumes that the command involves a user program or utility program. It then builds file COM.CM (FCOM.CM if it is operating in the foreground; mapped RDOS only) to contain the edited command line, including filenames, switches, and other arguments. It then brings in the utility program, which always examines COM.CM, or a user program, which may examine COM.CM. The COM.CM built for each utility program has a specific format, which the utility expects; the COM.CM built for a user program also has a specific format, which you can use to pass instructions or arguments to your program.

For example, assume that you issue the command line:

```
FOO)
```

The CLI does not recognize FOO as a known command word. It therefore builds a command file with the word/byte organization shown in Figure D-2.

word	byte/contents	byte/contents
0	F	O
1	O	null
2	0	0
3	0	0
4	377_8	

Figure D-2. COM.CM File for Command FOO)

Each character of the filename occupies a byte. The filename is terminated by a null byte. 2 words (4 bytes) are reserved for global switches of FOO. Each letter switch sets a bit. /A sets bit 0 of the first word, etc., as shown in the switch/bit correspondence diagram in Figure D-3. The CLI uses 377_8 for the terminal byte.

Bit 15 of the second word in the global switches field is always set for files running under BATCH (RDOS only). This informs RDOS that it is running BATCH instead of the CLI.

When a square bracket is detected in an RLDR command line, it is passed as a filename with bit 10 in the second word of the switch information set (this bit follows the Z switch position). A comma within a square bracket is treated the same way (bit 10 set).

For any non-CLI command you type, the COM.CM is arranged the same way. Each filename argument gets a byte for each character and is terminated by a null byte; four bytes are reserved for the switches of each filename. The CLI does not interpret switches when it builds the command file; it simply sets the appropriate bit. Assume that you type the command:

```
FOO/B A ZZZ/X MUMBL)
```

The CLI would then build the COM.CM file shown in Figure D-4. (You can examine COM.CM after any command with command FPRINT/Z/B COM.CM.)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Word 0 (2 in figure D-2)	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
Word 1 (3 in figure D-2)	Q	R	S	T	U	V	W	X	Y	Z						

Figure D-3. Bits Set for Switches by the CLI

Word	Left byte/ contents	Right byte/ contents	
0	F	O	Command file name FOO, terminated by null byte.
1	O	null	
2	1B1	0	Global switch field of FOO. Bit 1 (B switch) on.
3	0	0	
4	A	null	Argument A, terminated by null byte.
5	0	0	2 words (4 bytes) reserved for local switches of A. None set.
6	0	0	
7	Z	Z	Argument ZZZ, terminated by null byte.
8	Z	null	
9	0	0	Local switches of ZZZ. Bit 7 of word 2 (switch X) on.
10	1B7	0	
11	M	U	Argument MUMBL, terminated by null byte.
12	M	B	
13	L	null	
14	0	0	Local switches of MUMBL. None set.
15	0	0	
16	377_8		

SD-00785

Figure D-4. COM.CM File for Command FOO/B zzz/x MUMBL)

Since the CLI does not interpret switches, you can write programs which interpret switches, and act on what they read.

A read line (.RDL) from a disk file always terminates on a null (or on a carriage return and form feed). This null can help you to read COM.CM and FCOM.CM arguments. Figure D-5 illustrates how a background program could read the first argument of the command file as well as its global switches. (We have removed the location data from the assembler listing.)

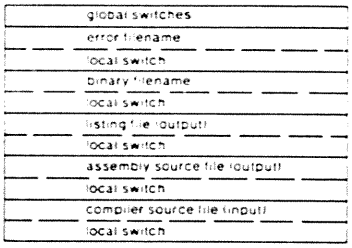
```
                        .
000001                  .TXTM 1
                        .
                        .
                        .
020420 READ:    LDA 0, COMCM    ;Pointer to COM.CM
000017          .SYSTM          ;System,
014005          .OPEN 5         ;Open COM.CM on channel 5.
000412          JMP ER          ;Required error location.
020421          LDA 0, ARG1     ;Pointer to first argument (which
                                ;is the filename of this program).
000017          .SYSTM          ;System,
015405          .RDL 5          ;read the first argument in
                                ;COM.CM. The null terminator
                                ;also transferred.
000406          JMP ER          ;Mandatory.
020430          LDA 0, GSWITCH  ;Pointer to global switches for
                                ;first argument.
024432          LDA 1, C4       ;Number of bytes to read.
000017          .SYSTM          ;System,
015005          .RDS 5          ;read the 4 bytes which specify
                                ;switches.
000401          JMP ER
                        .               ;This code derives the switches from
                        .               ;values in the 4 bytes, and directs
                        .               ;program control according to what it
                        .               ;finds.  Parameter file PARU.SR
                        .               ;defines switch bit settings. The
                        .               ;program can then proceed to check
                        .               ;the second argument, and its
                        .               ;local switches, if it wants.
        ;Fatal error handler:
000017 ER:      .SYSTM
000400          .ERTN
000401          JMP .
        ;Other labels:

000042"COMCM:   .+1*2           ;byte pointer
041517          .TXT "COM.CM"   ;to COM.CM.
046456
041515
000000
000054"ARG1:    .+1*2           ;Pointer to space for
000012          .BLK 10.        ;first argument (decimal).
000102"GSWITCH: .+1*2           ;Pointer to space for
000002          .BLK 2          ;global switches.
000004 C4:      4
                        .
```

Figure D-5. Reading File COM.CM

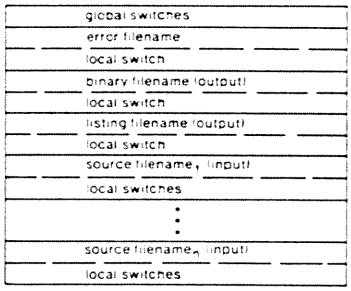
Utility Program COM.CM Structures

Following is a series of illustrations depicting the command file structures for all system utilities available via the CLI. The CLI always builds COM.CM this way, regardless of the order you specified in the utility command line.

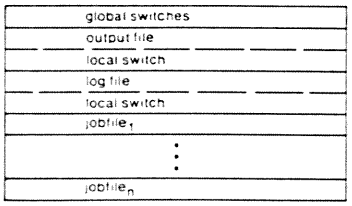
ALGOL



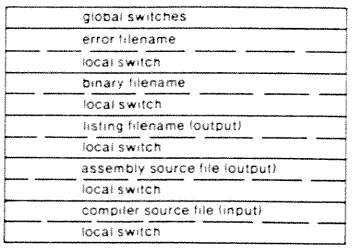
ASM



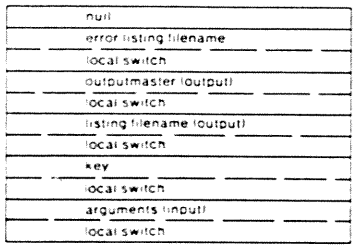
BATCH



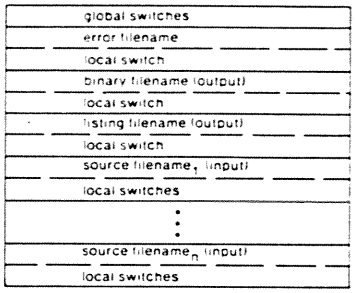
FORT
or
FORTRAN



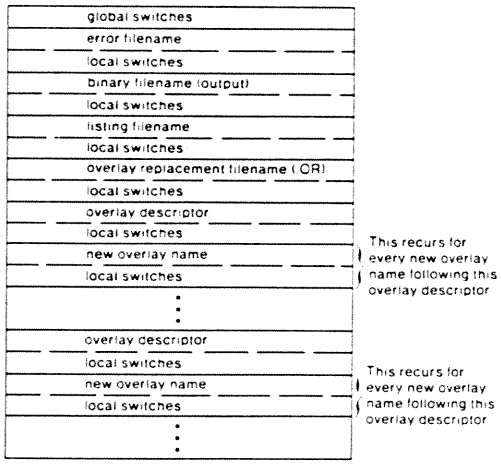
LFE



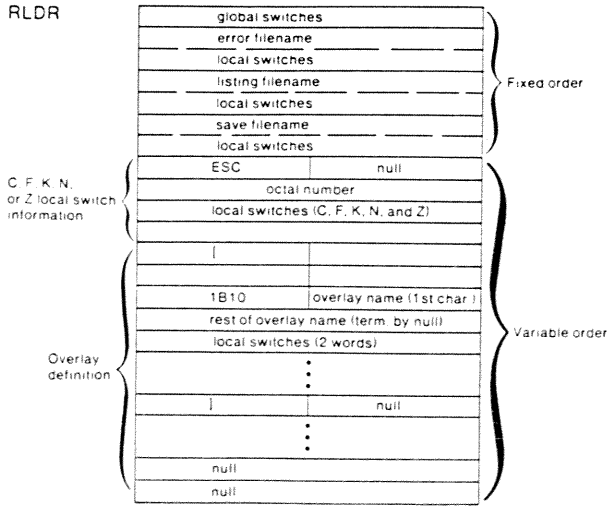
MAC



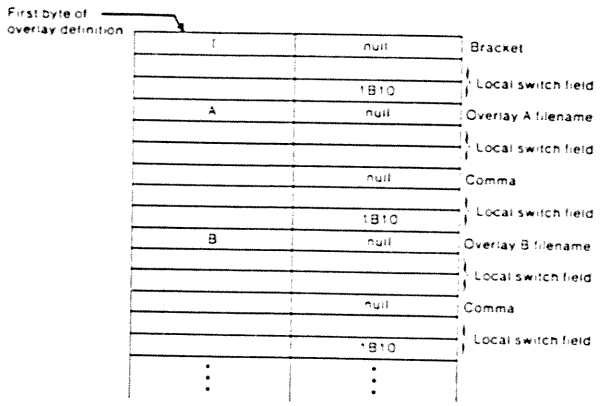
OVLDR



RLDR

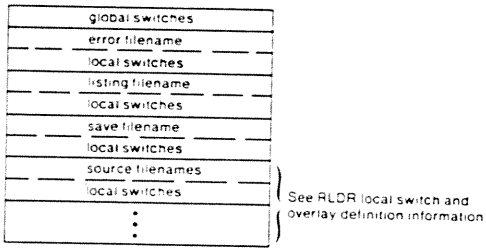


Given the overlay definition *[A, B, C]*, the overlay definition field would look like:



Local switch information fields (C, F, K, N, or Z) and overlay definition fields may occur in any order following the fixed portion of the command file.

CLG



Source filenames are listed in the order that they appear in the command line; source filenames receive the extension .FR by default. The following local switch operations occur in CLG:

/A Source file argument is given the extension .SR.

/C Channel argument is converted to octal (RLDR phase).

/K Task argument is converted to octal (RLDR phase).

/O Source file argument is given the extension .RB.

End of Appendix

Index

Within this index, the letter "f" means "and the following page"; the letters "ff" mean "and the following pages". All entries are lowercase, except CLI commands (e.g., APPEND), filenames (e.g., FLOG.CM), and acronyms (e.g., CLI). Italic entries indicate primary page references.