DataGeneral

Software Documentation

RDOS, DOS, and DG/RDOS Command Line Interpreter

RDOS, DOS, and DG/RDOS Command Line Interpreter

069-400015-01

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. 069-400015

Printed in the United States of America
Revision 01, December 1984

NOTICE

RDOS, DOS, and DG/RDOS
Command Line Interpreter
069-400015

Revision History:

Original Release - September 1983
First Revision - December 1984
Addendum 086-000098-00 - May 1985

Effective with:

RDOS Rev. 7.3
DOS Rev. 3.5
DG/RDOS Rev. 1.09

A vertical bar or an asterisk in the margin of a page indicates substantive change or deletion, respectively from the previous revision.

DataGeneral

ADDENDUM

Addendum to RDOS, DOS, and DG/RDOS Command Line Interpreter

086-000098-00

This addendum updates manual 069-400015-01. See Updating Instructions on reverse.

Ordering No. 086-000098

Printed in the United States of America
Revision 00, May 1985

069-400015-01

NOTICE

Addendum to
RDOS, DOS, and DG/RDOS
Command Line Interpreter
086-000098

Updating Instructions

This addendum (086-000098-00) to the RDOS, DOS, and DG/RDOS Command Line Interpreter corrects the first page of the Table of Contents.

To update your copy of 069-400015-01, please remove manual pages and insert addendum pages as follows:

Remove	Insert
Title/Notice	Title/Notice
vii/viii	vii/viii

Insert this sheet immediately behind the new Title Notice page.

The addendum number appears on all pages in this addendum.

Preface

This manual describes the Command Line Interpreter — the CLI. The CLI is the user's primary interface with Data General's Real-Time Disk Operating System (RDOS), the Disk Operating System (DOS), or RDOS for DESKTOP GENERATION™ systems (DG/RDOS). The CLI is the same for the three operating systems, except for a few features and commands, as noted in the text.

In this manual we explain CLI concepts and features in the context in which a user will encounter them. In the course of presenting material for general purpose then special purpose use, the discussions include examples of relevant CLI commands. The CLI Command Dictionary, in Chapter 10, contains a fuller discussion of each CLI command and utility.

Organization of Manual

Below we summarize the contents of the 10 chapters and 3 appendixes.

Chapter 1, Overview, summarizes CLI uses and features.

Chapter 2, Getting Started with the CLI, covers command line components, entering and correcting commands, CLI control characters and special symbols, command line shorthand, error messages, and operating system shutdown. All new users of the CLI should read this and Chapter 3 for basic information on using the operating system.

Chapter 3, Using Files and Directories, explains disk preparation, disk file and directory structure, filename conventions and file types, displaying and manipulating files, creating and using a directory structure, file links accross directories, and file characteristics. Read this and Chapter 2 for a foundation for using your operating system.

Chapter 4, Grouping CLI Commands: Macro and Indirect Files, documents the use of macro and indirect command files for automating CLI sessions. This chapter also explains CLI variables, the MESSAGE command, and command line interpretation and execution orders.

Chapter 5, Using Input/Output Devices, describes the I/O devices available, and how to initialize and access devices and files, move data, store data on backup media, and display data.

Chapter 6, Executing Foreground and Background Programs, explains how to set up and use foreground-background processing on either unmapped or mapped systems.

Chapter 7, Invoking System and Program Utilities, summarizes the utilities available for program development, file backup, and system administration.

Chapter 8, The CLI — Assembly Language Interface: CLI.CM and COM.CM, covers the use of the two CLI files to store commands for program use.

Chapter 9, The Batch Monitor, documents how to use Batch Monitor commands to simulate a CLI session. The chapter discusses Batch input and output for files and devices, and the BATCH command. A Batch Monitor Command Dictionary, detailing each of the Batch commands, concludes the chapter.

Chapter 10, Command Dictionary, is an alphabetical collection of all CLI commands and utilities. For each command and utility, the dictionary entry includes a capsule description, a format line, a list of global and local switches, and a description of arguments, uses, and ramifications. Each entry concludes with one or more examples of using the command or utility.

Appendix A, CLI Command Summary, briefly lists each command, its arguments, and its description.

Appendix B lists Error Messages.

Appendix C charts the ASCII character set.

Reading Path

New users of RDOS, DOS, or DG/RDOS should read Chapters 2 and 3 to build a foundation for using the operating system. Chapters 4 and 5 offer more detailed information on slightly less basic uses. Chapters 6, 7, 8, and 9 document special features of the CLI that users will consult on occasion. Chapter 10, the Command Dictionary, serves as a reference for all users for all CLI commands and utilities. Chapter 1 summarizes CLI features for readers wanting an overview.

Related Manuals

Below we list manuals for the RDOS, DOS, and DG/RDOS operating systems.

Introduction to RDOS (DGC No. 069-400011)

How to Load and Generate RDOS (DGC No. 069-400013)

RDOS System Reference (DGC No. 093-400027)

RDOS/DOS Assembly Language and Program Utilities (DGC No. 069-400019)

RDOS/DOS Superedit Text Editor (DGC No. 069-400017)

RDOS/DOS Text Editor (069-400016)

RDOS/DOS Debugging Utilities (DGC No. 069-400020)

RDOS/DOS Sort/Merge and Vertical Format Unit Utilities (DGC No. 069-400021)

Guide to RDOS Documentation (069-400012)

RDOS, DOS, DG/RDOS Backup and Move Utilities (DGC No. 069-400022)

How to Generate Your DOS System (DGC No. 093-000222-02)

Using DG/RDOS on DESKTOP GENERATION™ Systems (DGC No. 069-000056)

Typesetting Conventions

We use these conventions for command formats in this manual:

COMMAND required *[optional]...*

Where	Means
COMMAND	You must enter the command (or its accepted abbreviation) 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 this argument. Don't enter the brackets; they only set off what's optional.
...	You can repeat the preceding entry or entries. The explanation will tell you exectly what you can repeat.

Additionally, we use certain symbols in special ways:

Symbol	Means
↵	Use the NEW LINE or carriage return key on your terminal. The carriage return key could be marked RETURN or CR.

All numbers are decimal, except for device codes and numbers marked octal. For example

27 buffers	means 27 decimal.
device code 27	means 27 octal.
say 27 octal	means 27 octal.

We show commands in UPPERCASE, but you can type them in lowercase, UPPERCASE, or any combination. In examples, we use

THIS TYPEFACE TO SHOW YOUR ENTRY
THIS TYPEFACE FOR SYSTEM QUERIES AND RESPONSES.

R

is the RDOS, DOS, and DG/RDOS operating system CLI prompt.

Contacting Data General

- If you have comments on this manual, please use the prepaid Remarks Form that appears after the Index.
- If you need additional manuals, please use the enclosed TIPS order form (USA only) or contact your Data General sales representative.

End of Preface

Contents

Chapter 1 - Overview of the CLI

Chapter 2 - Getting Started With the CLI

Chapter 3 - Using Files and Directories

Chapter 4 - Grouping CLI Commands: Macro and Indirect Files

Chapter 5 - Using Input/Output Devices

Chapter 6 - Executing Foreground and Background Programs

Chapter 7 - Invoking System and Program Utilities

Chapter 8 - The CLI — Assembly Language Interface: CLI.CM and COM.CM

Chapter 9 - The Batch Monitor

Chapter 10 - CLI Command Dictionary

Tables

Table

Illustrations

Figure

Chapter 1
Overview of the CLI

The CLI — the Command Line Interpreter — serves as an interpreter between the user and the operating system. With the CLI, you can communicate with RDOS, DOS, or DG/RDOS to exercise most features of the particular operating system.

RDOS supports real-time control, program development, or batch operations. RDOS runs on a wide range of NOVA® and ECLIPSE® configurations, including those that support Microproducts peripherals. RDOS is compatible with DOS, which runs on the microNOVA® as well as NOVA and ECLIPSE. RDOS is also compatible with DG/RDOS, which runs on DESKTOP GENERATION™ computers.

You use CLI commands to manage every aspect of the system. For example, you use CLI commands to start and stop the operating system, to manage files, to direct input and output, and to develop programs. You also use CLI commands to invoke utilities of the operating system, such as the SPEED text editor or the DDUMP diskette dump program. System utilities include programs for data backup, data transfer, text editing, program compilation, assembly, loading, and debugging.

You also use the CLI to invoke the Batch Monitor, a program that lets you simulate a CLI session in deferred processing. You would group Batch commands, which are very similar to CLI commands, in a Batch command file. You could then process these commands at a time of your choosing. Chapter 9 of this manual covers Batch processing.

Throughout this manual, we use the term RDOS to include DOS and DG/RDOS; where the subject is specific to an operating system, we use the explicit name.

Uses of the CLI

When the operating system starts, it first runs the CLI. You then use CLI commands to tell the system what to do. You use it to manage the file system, control various aspects of the CLI environment and system configuration, invoke other utilities and user programs, and manage processes and other programs. CLI commands fall into these categories:

- File Management — Commands that create, manipulate, and delete files, For example, you use CLI commands to copy, append, and type out or print files.
- Directory Management — Commands that control the dividing of disk media into directories, and the storing of files. For example, you use CLI commands to create, initialize, and release directories.
- System Control — Commands that control various functions of the operating system, such as time of day or memory allocation. For example, you use CLI commands to set, display, and collect system statistics.
- System Utilities — Commands that invoke utilities of the operating system. For example, you use CLI commands to run a text editor or to debug a program.

Features

Some important functions of the CLI let the user

- choose among three different data organizations for disk files.
- use a file on disk, magnetic tape, or paper tape; transfer any file between these media.
- build a file of filenames, selecting the files by date, name, or other criteria.
- combine many files into one file.
- restrict access or permit conditional access to a file.
- access a file anywhere on disk by way of a link.
- organize disks into logical subdivisions.
- store sequences of CLI commands in indirect or macro command files.
- record system terminal activity in a log file.

Using this Manual

The remaining chapters show you how to use the CLI for everyday activities and special activities.

Read Chapters 2 and 3 for a solid foundation in using the CLI: learn how to enter and manipulate CLI commands, and how to create and manipulate files and directories. Read Chapters 4 and 5 for more detail on other frequently used functions: automating CLI sequences in indirect or macro command files, or using I/O devices.

Chapters 6 through 9 cover specialized uses of the CLI and system: foreground-background processing, a summary of system utilities, the CLI interface to assembly language programming, and the Batch Monitor.

The Command Dictionary in Chapter 10 documents the use of every CLI command and system utility, and serves as a reference for new and experienced users.

The Appendixes contain tables of CLI commands, error messages, and ASCII character codes.

End of Chapter

Chapter 2
Getting Started With the CLI

This chapter covers topics that are basic to your use of the CLI, including:

- the components of a command line
- entering a command
- how to correct typing mistakes
- the types of input the CLI recognizes
- how the CLI responds to your input
- command line shorthand
- ending a CLI session

Before you can use the CLI, the RDOS operating system must be running on your computer. The first time you bring up RDOS, you follow procedures known as "loading and generating a system". Once you have generated your system, you can start up RDOS by bootstrapping the operating system from the master directory.

The procedures for generating and bootstrapping are beyond the scope of this manual, as they can vary from machine to machine. You can find complete instructions for starting up and shutting down RDOS in *How to Load and Generate RDOS* manual (DGC 069-400013), or *How to Generate Your DOS System* (DGC No. 093-000222), or *Using DG/RDOS on DESKTOP GENERATION™ Systems* (DGC 069-000056).

When you load RDOS, DOS, or DG/RDOS, the operating system runs a program called the Command Line Interpreter — the CLI — that lets you enter commands to the operating system. The CLI is available when the RDOS CLI prompt, *R*, appears at your terminal:

R

If your terminal does not display an R prompt, turn to one of the manuals we just listed for assistance.

Entering Commands to the CLI

Before you begin to use specific CLI commands, you should be familiar with the components of a command line, and the syntax (or format) for entering command lines.

Components of the Command Line

A command line consists of a CLI command name, arguments, switches, and a command line terminator. The order and the manner in which you specify each component on a line is important to the CLI's interpretation of the line. Therefore, all command lines must follow a general syntax or format:

```
COMMAND/global-switches arguments/local-switches
```

The following sections look at each component separately — the command, the command line terminator, arguments, and the two types of switches.

Commands

A CLI command name must appear as the *first word* in a command line. If the first word is not a CLI command, the CLI will look for a macro command file (.MC file), then an executable file (.SV file) of that name. If the CLI does not find a command, a macro, or a save file, it displays a message such as

FILE DOES NOT EXIST: FIRSTWORD.SV

to inform you that the CLI could not recognize your input.

An example of a simple command follows, where all you need enter is the command name followed by either a CR or a NEW LINE, depending on your system:

```
R
LIST ⤸
```

LIST is a CLI command that displays the names of files in the current directory. LIST offers many options and accepts arguments, but does not require an argument. We will be using the LIST command in our examples throughout this chapter.

Chapter 10, the Command Dictionary, describes all CLI commands.

The Command Line Terminator

A *command line terminator* enters a command line to the CLI for execution. Depending on the operating system, computer, or the terminal you're using, the command line terminator is either the key marked NEW LINE, or the key marked CR or RETURN, which stands for carriage return.

Examples in this manual represent command line termination, whether NEW LINE or CR, with the symbol ⤸.

If you are not sure which key to use, try each one at your terminal, without typing anything else on the line. The key that functions as your command line terminator always returns an *R* prompt. The other key will not enter a CLI command (but could move the cursor down a line). If you use this key at the end of a command line by mistake, simply follow it with the correct terminator key.

The following example enters the LIST command, and shows a sample response.

```
R
LIST ⤸
FILE1.      0 D
FILE2.      0 D
R
```

When the CLI processes a command, it displays any information called for by the command (or an error message if it could not execute the command), and then displays an *R* prompt. The *R* prompt indicates that the CLI has finished execution, and is ready to accept another command.

Arguments

An *argument* to a command names an object upon which the command is to take action. Usually, arguments are the names of files or devices.

Some commands require arguments. For example, the CLI can't execute the command that prints a file on the line printer unless you supply the file's name in the command line. For example

```
R
PRINT FILE88 ⤸
```

Some commands accept optional arguments. In an earlier example, we used the LIST command by itself to display a list of files. You can direct LIST to list a particular file by specifying the name of that file as an argument.

```
R
LIST FILE1 ↵
FILE1.    0 D
R
```

Some commands accept multiple arguments on a command line.

```
R
LIST FILE1 FILE2 FILE3 ↵

FILE1.    0 D
FILE2.    0 D
FILE3.    0 D
R
```

The syntax line for each command in the Command Dictionary tells you when a command requires arguments, which arguments are optional, and when you can specify more than one argument. You can specify arguments in any order on the command line, unless the command line syntax for that command specifies otherwise.

You must precede each argument on a command line with an argument delimiter, so that the CLI can recognize it as an argument. The valid argument delimiters are

- one or more spaces
- a comma
- a combination of a comma and spaces

It's best to choose a simple delimiter that you prefer, and to use it consistently. This manual uses a space to separate an argument from a command, and to separate one argument from another:

```
LIST FILE1 FILE2
```

Switches

Switches are options that modify a command or its arguments. A switch is made up of a slash (/) and a single letter or number. For example

```
/E
```

The CLI recognizes two types of switches — global switches and local switches — as discussed separately below.

The Command Dictionary describes the switches available for a particular command. Note that switches have meaning only to the command under which they are listed. Many commands may have an /E switch, but the /E switch will cause a different, specific action for each command.

Global Switches

A *global switch* is appended to a *command name*, and affects the action of the command. Commands can take multiple switches. Note that no spaces may appear between a switch and the command, or between a switch and another switch.

For example, the LIST command has a switch, /E, that instructs LIST to display a variety of information about each file, such as the date and time the file was last modified.

```
R
LIST/E ↵
FILE1.    0 D  01/06/84   16:27   01/06/84   [006564]   0
PROGX.    0 D  01/09/84   13:49   01/09/84   [006565]   0
FILE2.    0 D  01/13/84   10:03   01/13/84   [006566]   0
R
```

The following example appends more than one switch to the LIST command. The /S switch instructs LIST to sort and display the filenames in alphabetic order.

```
R
LIST/E/S ↵
FILE1.    0 D  01/06/84   16:27   01/06/84   [006564]   0
FILE2.    0 D  01/13/84   10:03   01/13/84   [006566]   0
PROGX.    0 D  01/09/84   13:49   01/09/84   [006565]   0
R
```

You can add arguments to this command line, as follows:

```
R
LIST/E/S PROGX FILE2 ↵

FILE2.    0 D  01/13/84   10:03   01/13/84   [006566]   0
PROGX.    0 D  01/09/84   13:49   01/09/84   [006565]   0
R
```

For the two arguments PROGX and FILE2, LIST displays a variety of information about each file (/E) and lists the filenames in alphabetic order (/S).

Local Switches

A *local switch* attaches to an argument, and modifies the command's treatment of that argument.

The Command Dictionary uses a format similar to the following to describe local switch arguments under each command:

name/S or n/S

where:

/S	is the switch.
name	tells you the local argument must be a filename argument.
n	tells you the local argument must be a number.

Local switches often affect the other arguments on the same command line, *unless you explicitly name the other arguments.* An explicitly named argument is one that does *not* include a template. A *template* is a symbol that allows you to specify a number of files, as discussed in Chapter 3. For the purposes of the next few examples, you need to know that the argument

```
FILE-.-
```

means "all files that begin with the letters F I L E, whether or not they have filename extensions". The symbol - (dash) is a template, here used twice in formation of the filename and the extension.

For example, LIST has a /N switch that instructs LIST to exclude a particular argument from the display:

```
R
LIST FILE-.- FILE2/N ↵

FILE1.    0 D
FILE3.    0 D
R
```

This command lists all files that begin with the letters F I L E, not including FILE2.

One of the most useful types of local switches allows you to specify files created before or after a given date. In the special format for these switches, you specify the date as an argument, and append the switch to it.

For example, LIST has an "after" switch, /A, whose format requires a date as an argument, as follows:

```
mm-dd-yy/A
```

This entire date argument applies to every argument in the command line:

```
R
LIST/S/E FILE-.- PROG-.- 1-9-84/A ↵

FILE2.    0 D  01/13/84   10:03   01/13/84   [006566]   0
FILE3.    0 D  01/13/84   11:34   01/13/84   [006567]   0
PROGX.    0 D  01/09/84   13:49   01/09/84   [006565]   0
R
```

Since FILE1 was not created on or after January 9, 1984, the CLI does not include it in the display. The other files satisfy the date condition.

Note that the /A switch does not apply to any arguments that you enter explicitly, without templates. For example, the command

```
LIST/S/E FILE-.- SINGLEOUT 01-09-84/A
```

lists all files that match the template FILE-.- only if they have creation dates that are on or after 01-09-84, and it lists the file SINGLEOUT, regardless of its creation date, because it is named explicitly.

How to Correct Typing Mistakes

Two keys enable you to correct typing mistakes on a line — a character delete key and a line delete key.

The key marked DEL or RUBOUT (depending on your terminal) erases the character immediately preceding the cursor. (A cursor marks your place on a line in the form of an underline (_) (a blinking underline on many terminals) or a highlighted box (#). A cursor is not visible on hard-copy terminals.)

If you type LISY instead of LIST, you could correct the line by pressing the character delete key, and then typing the T. The line

R
LISY<DEL>T

results in LIST.

If you type LISSY instead of LIST, press the character delete key twice, and then type the T. The line

R
LISSY<DEL><DEL>T

results in LIST.

The backslash key (\ ) erases an entire line and resets your cursor to the beginning of the current or next line (depending on your operating system), where you can re-enter your command correctly. Note that \ resets the cursor but does not display an R prompt. For example, the sequence

R
QWERTY\
LIST

results in LIST.

If you enter a command line before you realize it has a typing mistake, the CLI tells you it can't recognize the command by displaying a message such as:

FILE DOES NOT EXIST: LISY.SV

You would then need to retype the command.

What the CLI Recognizes

You need to be aware of all of the characters and symbols that have a special meaning to the CLI. These can be grouped into the following categories:

- alphanumeric characters
- special symbols
- special keys
- control characters

We have encountered some (but not all) uses of the first three of these categories when we examined the components of a command line:

combinations of alphanumeric characters	commands, arguments, and switches
special symbols	the line delete symbol (\ ), and delimiters such as commas, spaces, and the switch delimiter (/)
special keys	DEL or RUBOUT, CR and NEW LINE

Next we discuss control characters, then all the special characters and keys of RDOS.

Control Characters

Control characters execute functions by overriding current CLI processing. You enter a control character by simultaneously pressing the key marked CTRL and the appropriate letter. In this manual we indicate a control character in the form CTRL-character, for example CTRL-A.

The most important control characters you need to know at this point are CTRL-A, CTRL-S, and CTRL-Q. (DG/RDOS users, note that under DG/RDOS, CTRL-A is preceded by CTRL-C to invoke the same function. See Table 2-1.)

CTRL-A immediately interrupts and aborts the execution of the current CLI command, and returns you to the CLI R prompt. You can use CTRL-A to exit from any CLI command or from many programs. CTRL-A displays the message *INT*, meaning "interrupt", before it returns the *R* prompt. At that point, you can enter a CLI command.

CTRL-S and CTRL-Q control screen displays. For example, if you are displaying a long file at your terminal, you can use CTRL-S to halt the display (preventing it from scrolling off the screen) so you can read it. When you are ready to look at more output, you press CTRL-Q to resume the display.

Figure 2-1 shows the use of CTRL-S, CTRL-Q, and CTRL-A.

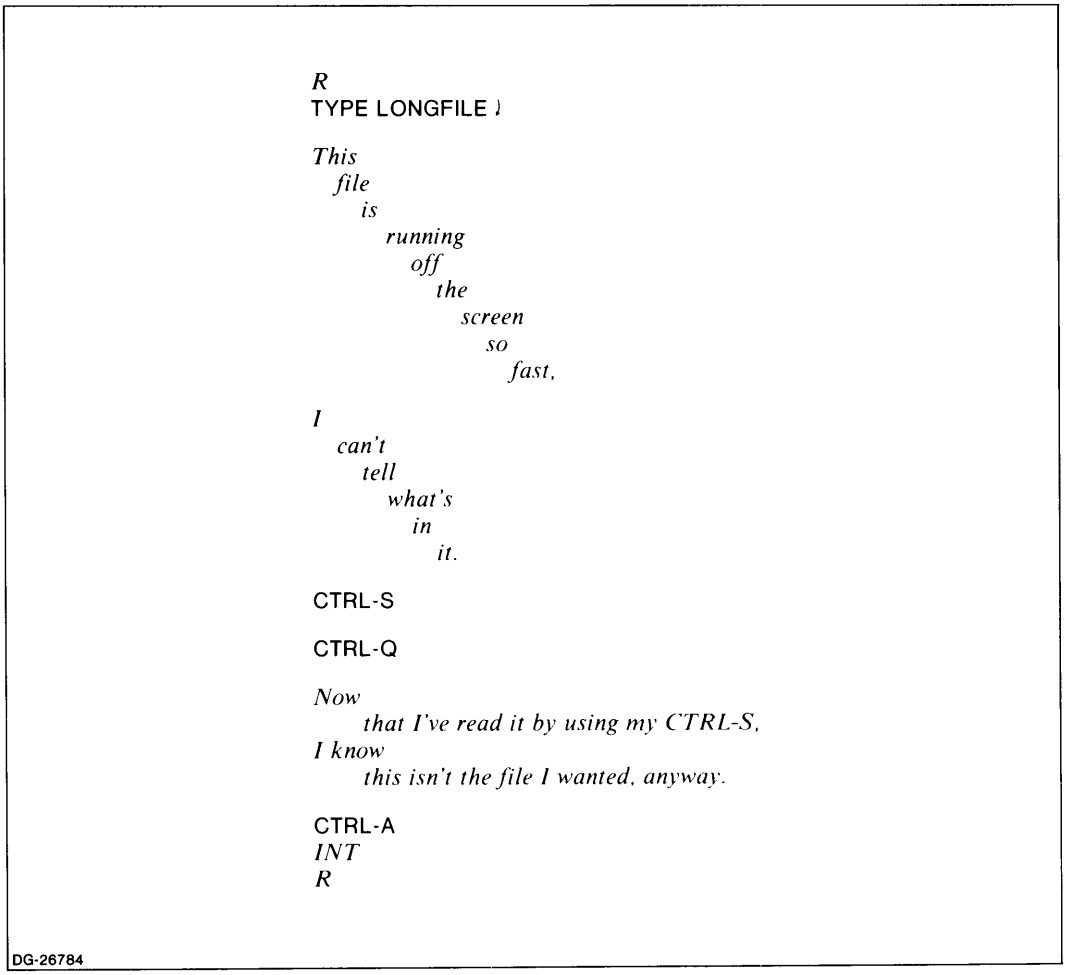


Figure 2-1. Using CTRL-S, CTRL-Q, and CTRL-A

Note that CTRL-S and CTRL-Q do not display any messages, so that if your terminal seems "dead", it could be that you have used a CTRL-S, reached the end of your output, and forgotten to use CTRL-Q to resume CLI output operations.

Table 2-1 describes all of the RDOS control characters and their functions. Note that under DG/RDOS, three of the functions are called by *control character sequences*, that is, two control characters entered sequentially.

Table 2-1. RDOS Control Characters

Control Character		Function
RDOS and DOS	**DG/RDOS**	
CTRL-A	CTRL-C CTRL-A	Immediately terminate current CLI execution from a foreground or background terminal, and return to the next higher-level program (usually the CLI).
CTRL-C	CTRL-C CTRL B	Same as above, but first wait for task I/O to complete, and create a breakfile of the current memory image called BREAK.SV (background) or FBREAK.SV (foreground).
CTRL-F	CTRL-C CTRL F	Terminate a foreground program from the background terminal, releasing foreground devices and directories.
CTRL-L	CTRL-L	Clear a CRT screen; terminate a command line.
CTRL-Q	CTRL-Q	Resume terminal output suspended with CTRL-S.
CTRL-S	CTRL-S	Suspend terminal output. (Resume with CTRL-Q.)
CTRL-Z	CTRL-Z	Close a file being created from the terminal with the XFER/A command. End-of-file character.

Special Characters of the CLI

Table 2-2 describes all symbols and characters that have special meaning to the CLI, and shows examples of how to use them. We will be encountering most of these as we proceed through this manual. You can refer to this table as you need it.

Table 2-2. CLI Special Symbols

Character	Keyboard Symbol	ASCII Code (Octal)	Function	Example
NEW LINE	NEW LINE	012	Terminates command line. (Ignored on systems that recognize CR.)	LIST ⤶
Carriage Return	CR	015	Terminates command line. (Ignored on systems that recognize NEW LINE.)	LIST ⤶
Delete	DEL or RUBOUT	177	Deletes individual characters from right to left on a line.	LISS<DEL>T ⤶
Erase Page	ERASE PAGE	014	Terminates command line and erases display screen.	
Backslash	\	134	Deletes current line.	ASDF\ LIST ⤶
Space		040	Separates arguments in command line. Extra spaces have no effect.	LIST YOU ME ⤶
Comma	,	054	Separates arguments in command line. Multiple commas indicate null arguments.	LIST HER,HIM ⤶
Slash	/	057	Sets off switch in command line.	LIST/E/S ⤶
Semicolon	;	073	Delimits CLI commands, separating multiple commands entered on one line.	LIST IT;TYPE IT ⤶
SHIFT-6	^	136	SHIFT-6 followed by line terminator extends command to following line. Do not confuse with the uparrow cursor control key.	LIST IT;TYPE IT;^ PRINT IT ⤶

(continues)

Table 2-2. CLI Special Symbols

Character	Keyboard Symbol	ASCII Code (Octal)	Function	Example
Period	.	056	Separates filename from extension. Also, adds or removes the time of day to CLI *R* prompt.	↵
Colon	:	072	Separates device names, directory names, and filenames in a pathname.	SYS:IT
Asterisk	*	052	Template character that represents any single character, except a period, in a filename or extension.	D**E.SV
Dash	-	055	Template character that represents any number of characters, except period, in a filename or extension.	D-E.-
Parentheses	( )	050/051	Enclose multiple commands or arguments in one command line for the CLI to expand to multiple lines.	(LIST,PRINT) IT ↵
Angle Brackets	< >	074/076	Enclose multiple arguments in a command line for the CLI to repeat.	LIST IT.<1,2> ↵
Commercial At	@	100	Encloses a filename in command line to invoke the file as an indirect file.	@LEMME@ ↵
Quotation Marks	"	042	Enclose a text string for literal interpretation in MESSAGE command.	MESSAGE "UNCLE"
Percent sign	%	045	Encloses a CLI-designated variable.	%MDIR%

(concluded)

Interpreting CLI Responses and Error Messages

The CLI responds to the command lines you enter in one of the following ways:

- The CLI displays any output the command may have generated, and returns an R prompt.
- The CLI displays an error message, and returns an R prompt.

Some commands do not generate any output, and you receive only the R prompt.

The CLI attempts to execute everything it can. For example, if you listed many filename arguments, and the CLI couldn't locate one of the files, the command would execute for all of the files except for that one file. The CLI would also notify you that it couldn't find that one file by displaying an error message.

Most CLI commands do not execute commands that threaten to overwrite an existing file. The message

FILE ALREADY EXISTS: FILENAME.SV

informs you that the CLI did not execute a command because it would be forced to overwrite an existing file (in this example, the file FILENAME.SV). The exception to this is the BUILD command.

If a command can't execute properly, or if the CLI can't interpret the command line, the CLI displays an error message. The error messages attempt to explain why a command, or a part of a command, did not execute. Table 2-3 describes some of the more common CLI error messages.

Table 2-3. Some Common CLI Error Messages

Message	Meaning
FILE ALREADY EXISTS	You cannot add information to or replace an existing file, unless you include a switch that allows it.
FILE DOES NOT EXIST	File does not exist in the current directory (or the directory you specified).
NO SUCH DIRECTORY	The specified directory is not initialized, or does not exist.

More on CLI Command Lines

The following sections focus on using command line shorthand techniques to cut down on repetitive typing. The chapter closes with instructions on ending CLI sessions.

Wrapping a Command Line

Your display screen or hard-copy display terminal can accept some maximum number of characters across a single line; the maximum is usually 80 characters per line, but could be 132 characters. Many display screens automatically wrap a line that is over 80 characters.

RDOS for its part accepts 132 characters per line. To enter a command line that is greater than 132 characters, use the uparrow character (^) (often the SHIFT-6 key), followed by a command line terminator. Enter the command up to 131 characters, then enter ^ ⤸. This continues or *wraps* the command onto the next line. You then use the command line terminator when you have finished typing everything you want to include in the command line. For example

```
R
LIST FILE1 FILE2 FILE3 PROGX NEWFILE ^ ⤸
OLDFILE ANYFILE ⤸

FILE1.        0      D
FILE2.        0      D
FILE3.        0      D
PROGX.        0      D
OLDFILE.     74      D
NEWFILE.     28      D
ANYFILE.    137      D
R
```

Typing Multiple Commands on a Single Line

So far, we've said that we end every command line with a terminator, and the CLI executes the command, and then returns an *R* prompt when it is ready to accept more input. You can include more than one command on a single line by using the *command line separator*, the semicolon (;) to delimit each command. You then use a command line terminator at the end of the line to enter the commands to the CLI, all at once.

For example

```
R
DIR MYDIR; LIST FILE1; TYPE FILE1 ⤸
FILE1.      50 D
THIS IS FILE1,
WHICH HAS ONLY THESE TWO LINES OF INFORMATION IN IT.
R
```

The CLI executes this line from left to right. First it executes the DIR command, then the LIST command, and then the TYPE command (which displays the contents of a file). Note that the CLI displayed all output, and then returned a single *R* prompt for all three commands.

Command Line Shorthand: Angle Brackets and Parentheses

RDOS provides two shorthand methods for cutting down on repetitive typing in command lines — the use of angle brackets and parentheses.

You may have discovered that most of the repetition in command lines comes from repeating similar filename arguments for one command, or specifying the same sets of files for more than one command. For example,

```
LIST FILE1 FILE2 FILE3
PRINT FILE1 FILE2 FILE3
```

By using angle brackets and parentheses, you can let the CLI perform the repetitions for you. This CLI operation is known as *expansion*. Using these two shorthand techniques, we could combine the previous two lines into the following single command line:

```
(LIST,PRINT) FILE<1,2,3>
```

Using Angle Brackets

Angle brackets, < >, cause the CLI to expand the enclosed portions of an argument into a single command line. For example,

```
R
LIST FILE<1,2,3> ↓

FILE1.      0 D
FILE2.      0 D
FILE3.      0 D
R
```

The shorthand argument FILE<1,2,3> expands to three arguments: FILE1, FILE2, and FILE3.

To test whether you're using angle brackets correctly for a command, you can first use the CLI MESSAGE command. Substitute the MESSAGE command for the command you intend, enter the angle-bracketed arguments, and issue the command to the CLI. MESSAGE will repeat the arguments on your display (returning your "message"), and you can see how the CLI interpreted the angle-bracketed arguments.

Syntax Rules for Angle Brackets

The following rules define how to use angle brackets in command lines.

- *The CLI expands arguments enclosed in angle brackets from left to right.*

  LIST FILE<1,2>

  expands to

  LIST FILE1 FILE2

- *You must separate each argument enclosed by angle brackets with a delimiter (a comma or a space). You can use multiple commas or spaces to indicate a null argument, as follows:*

  LIST FILE<,1,2>

  expands to

  LIST FILE FILE1 FILE2

  The null argument (,) causes the CLI to include the argument FILE without expanding it.

- *You can nest angle brackets within other angle brackets, without limit.*

  In the following examples, our file arguments include directory names in the form directory:filename. Directories contain files, and are discussed in Chapter 3.

  LIST PROJ<83:FILE1,84:FILE<1<A,B>,2>>

  This line expands to

  LIST PROJ83:FILE1 PROJ84:FILE1A PROJ84:FILE1B PROJ84:FILE2

- *When the CLI encounters nested angle brackets, it expands the innermost brackets first, and then proceeds to the outermost brackets.*

  LIST PROJ<83:FILE1,84:FILE<1<A,B>,2>>

  The CLI begins expanding this line with <A,B>:

  LIST PROJ<83:FILE1,84:FILE<1A,1B>,2>

  and then expands the <1A,1B>:

  LIST PROJ<83:FILE1,84:FILE1A,84:FILE1B,84:FILE2>

  and finally expands the outermost angle brackets:

  LIST PROJ83:FILE1 PROJ84:FILE1A PROJ84:FILE1B PROJ84:FILE2

Using Parentheses

Parentheses cause the CLI to expand the enclosed commands or arguments into multiple command lines. For example,

```
R
(LIST,TYPE) FILE1 FILE2 ↵

FILE1.      68D
FILE2.      50D

THIS IS FILE1,
WHICH HAS ONLY THESE TWO LINES OF INFORMATION IN IT.
THIS IS FILE2, WHICH CONTAINS ONLY THIS SENTENCE.
R
```

The command (LIST,TYPE) expands to two commands, LIST and TYPE. Note that these commands execute for each argument specified on the command line: LIST lists both files, and TYPE displays the contents of both files.

Syntax Rules for Parentheses

The following rules define how to use parentheses in command lines.

- *The CLI executes elements in parentheses from left to right.*

  ```
  (LIST,PRINT) FILE1
  ```

  expands to

  ```
  LIST FILE1
  PRINT FILE1
  ```

- *You must use a comma to delimit each element enclosed within parentheses.*

  ```
  (LIST,PRINT,DELETE) FILE1
  ```

- *You cannot nest parentheses within another set of parentheses.*

- *When the CLI encounters more than one set of parentheses in a command line, it extracts one element from each set of parentheses, returns to the beginning of the line, and then extracts the next element from each set.*

  ```
  (LIST,PRINT) (FILE1,FILE2)
  ```

  expands to

  ```
  LIST FILE1
  PRINT FILE2
  ```

Ending a CLI Session

When you are finished with a CLI session, you can do any of the following, according to your requirements:

- leave the operating system and the computer running.
- shut down the operating system, which leaves the computer in a halted state.
- shut down the operating system and power off the computer.

NOTE: Always shut down the operating system before powering off the computer.

If no one else will be using the system, you may want to shut down the operating system for the sake of security. The different procedures for shutdown of RDOS (and DOS), and DG/RDOS, are discussed in the next sections.

Shutting Down RDOS and DOS

The RDOS and DOS operating systems shut down when you make some key system files unavailable to the operating system. These files reside on disk in the *master directory*. The master directory contains system administration files and system bootstrap files.

As an RDOS or DOS user, you can shut down the system by typing the command

```
R
RELEASE %MDIR% ↵
MASTER DEVICE RELEASED
```

%MDIR% is a *variable* that directs the CLI to supply the name of the system master directory. Releasing the master directory shuts down the RDOS operating system, and you receive the message *MASTER DEVICE RELEASED.* You can bring RDOS up again from this point by bootstrapping the system.

Alternatively, you can specify the name of the disk on which the master directory resides. For example, if DP0 is the master directory, you can shut down the system with this command:

```
R
RELEASE DP0 ↵
MASTER DEVICE RELEASED
```

Note that under RDOS and DOS you can release the master directory while you're located in any other system directory. That is, another directory can be the current directory when you issue the RELEASE %MDIR% or RELEASE diskname command. See Chapter 3 for a discussion of the current directory.

You also must use the RELEASE command when preparing to take any disk or diskette offline. For example, before physically removing the cartridge disk DP4, issue the command

```
RELEASE DP4 ↵
```

This ensures that the CLI closes files properly on the disk.

For more information, see the manual *How to Load and Generate RDOS* or *How to Generate Your DOS System.*

Shutting Down DG/RDOS

To shut down DG/RDOS, you must first get to the master directory by typing in the command

```
R
DIR %MDIR% ↵
```

DIR is a command that moves you to different directories; %MDIR% is a *variable* that directs the CLI to supply the name of the system master directory. (You can type in the name of the master directory, if you know it, rather than use %MDIR%; for example DP0 could be the master directory name).

Alternatively, you can specify the name of the disk on which the master directory resides. For example, if DE0 contains master directory, you can move there with the command:

```
R
DIR DE0 ↵
```

Once located in the master directory, you shut down the operating system by typing BYE to the R prompt. This invokes the BYE macro, which performs the shutdown procedure. For example

```
R
BYE ↵
STARTING SYSTEM SHUTDOWN
MASTER DEVICE RELEASED
Filename?
```

You can bring DG/RDOS up again from this point by entering the name of the operating system, or by bootstrapping the system, or you could turn off power to the equipment.

Note that in DG/RDOS, the operating system does not shut down if you simply release the master directory, as it would in RDOS or DOS. The DG/RDOS BYE macro does release the master directory as one of its steps in shutting down the system. Be sure to use BYE when you want to stop running DG/RDOS.

You also *must* use the RELEASE command when preparing to take a diskette offline. For example, before physically removing the diskette in DJ1, issue the command

```
RELEASE DJ1 )
```

This ensures that the CLI closes files properly on the diskette.

For more information, see the manual *Using DG/RDOS on DESKTOP GENERATION™ Systems.*

End of Chapter

Chapter 3
Using Files and Directories

This chapter explains how to create, access, and manage disk files and directories. These actions form the foundation for using your RDOS system.

Storing Information on Disks and Diskettes

Disks and diskettes are devices on which you can store and retrieve data. Disks accommodate data storage and retrieval with a great deal of efficiency and flexibility.

Storing and retrieving data are often referred to as input/output, or I/O, or as writing and reading.

In this manual, we generally use the term *disk* to refer to both disks and diskettes. We specify *diskette* when necessary.

Preparing a Disk for Use

Before you can store files on a disk, the disk must be hardware and software formatted to accept RDOS file and directory structures. Disks supplied by Data General Corporation are hardware formatted. You software format a disk with both the DKINIT utility (DOSINIT for DOS) and the CLI command INIT/F.

During software formatting with INIT/F, RDOS places two files on the disk — MAP.DR and SYS.DR. These files keep track of locations and identifiers for the files you later store on the disk.

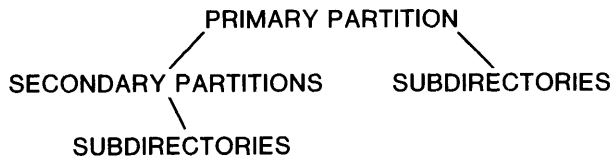
To software format RDOS disks and DOS diskettes, refer to the description of DKINIT in *How to Load and Generate RDOS*, and DOSINIT in *How to Generate Your DOS System.*

To software format DG/RDOS disks and diskettes, refer to the chapter on preparing disks and diskettes in *Using DG/RDOS on DESKTOP GENERATION™ Systems.*

Introduction to RDOS File and Directory Structures

You can divide the storage space on a disk into directories; the directories form a structure in which to keep files.

You can create a directory structure up to three levels deep, as illustrated below. *Primary partition, secondary partition,* and *subdirectory* are the names of the different types of RDOS directories.



Files

A *disk file* is a collection of information that is given a name and stored on a disk. A file can contain a memo, a program, or any collection of information.

Disk files are the primary method of storing information on your system, both because it is easy to store and organize files on a disk, and easy to retrieve files from a disk in order to display, execute, or make changes to them.

You use the CLI to create files, and work with files.

Directories

A *directory* is a special type of disk file that holds groups of other disk files. Directories allow you to arrange the hundreds of files you can store on a disk into functional groups.

You use CLI commands to create and work with directories. You can work within one directory, or easily move to work in another directory. You can also move files between directories.

RDOS has three types of directories — the primary partition, secondary partitions, and subdirectories. When we use the term *directory*, we are referring to any one of these three types.

Two directories have a special status: the directory that acts as your current directory, and the directory designated as your master directory.

The Current Directory

The directory you are presently working in is known as your *current directory*. RDOS assumes that the files you refer to reside in your current directory, unless you explicitly refer to another directory.

There are three ways to gain access to the files in any directory:

1. You can change your current directory to another directory. (DIR command.)
2. From the current directory, you can refer explicitly in a command line to a file in another directory. (Pathname, also known as directory specifier.)
3. You can create a file in your current directory that *links* to a file in another directory. (LINK command.)

RDOS does not recognize a directory until you open it for access, or *initialize* it (INIT command).

The Master Directory

The disk with which you bootstrap RDOS, called the master device, contains a *master directory*. The master directory is the directory that contains RDOS system files necessary to run the operating system and the CLI.

The master directory plays an important role in your directory structure for a number of reasons: You use it to start up and shut down RDOS. When you start up RDOS, the master directory is your current directory.

Creating Files

Before we introduce the commands for creating files, we need to go over the rules for naming files, and explain the differences between the three RDOS file constructions — random, contiguous, and sequential.

Naming Files

RDOS *filenames* can be up to 10 characters long, plus an extension. The *filename extension* consists of 1 or two characters, always preceded by a period (.). (The period is known as a *delimiter*.) MYFILENAME.XT is an example of a legal RDOS filename.

You can use the following characters in a filename:

- alphanumeric characters

  a through z
  A through Z
  0 through 9 (not in initial position)

- and the character $ (dollar sign)

Note that filenames cannot begin with a number.

RDOS truncates filenames having more than 10 characters in the main body of the filename, or more than 2 characters in the extension. For example, RDOS would recognize the filename TREMENDOUSFILE.EXTENSION as TREMENDOUS.EX.

RDOS stores only the uppercase version of a filename. When you type a filename in lowercase letters (e.g., file1), RDOS automatically converts the letters to uppercase (e.g., FILE1).

You can assign a file any name of your choosing, but should avoid using RDOS reserved filenames, discussed shortly. Generally, you should try to assign a name that describes the contents of the file.

Filename extensions classify files of the same type. RDOS uses predefined extensions for particular types of files, discussed below. You can choose your own extensions for all other types of files, or you can choose not to use extensions.

Note that *no two files can have the same name within the same directory*. Even when files exist in different directories, you should avoid assigning duplicate names to files.

The following is a list of some valid RDOS filenames:

```
FILE1
File1.SV
Longername.1
$$FILENAME
Z123abc.$$
```

Because the period is a delimiter, not a character in the filename or extension, then the names FILE1 and FILE1. are identical.

Reserved Filenames

RDOS reserved filenames include the names of RDOS commands and utilities, input/output devices, and RDOS reserved files. You should avoid using these names for one of your own files to ensure that RDOS can always locate the correct file.

The names of all RDOS commands and utilities are listed in Appendix A.

RDOS device names are usually three to five characters long, and many of them begin with the character $. For example, the device name for a line printer is $LPT. You can find a list of RDOS device names in Table 5-1.

Table 3-1 describes RDOS reserved files.

Table 3-1. RDOS Reserved Files

Filename	Function
CLI.OL CLI.ER	CLI overlay file and error interpretation file.
(F)CLI.Tn * (F)CLI.Cn (F)CLI.Sn	CLI virtual buffer files. n represents current system level.
(F)CLI.CM (F)COM.CM	CLI command files.
(F)LOG.CM	Log file created by LOG command.
(F)TML.TM	Temporary sort file of LIST/S command.
MAP.DR SYS.DR	Operatins system files that track locations and identifiers of disk files.

*The letter F prefixes RDOS files generated by a foreground process.

Filename Extensions

RDOS assigns predefined filename extensions to files having special properties. Other extensions, not required but accepted as standards, are recommended for certain types of files. Consistency in filename extensions is useful for identifying similar types of files by their extensions.

Table 3-2 lists required filename extensions. For most of these file types, RDOS or the creating utility automatically assigns the extensions. For a macro command file (.MC), you must append the extension when you create the file. Table 3-3 lists the extensions that we recommend you use. These file types are explained in more detail later, as we discuss their uses.

Table 3-2. Required Filename Extensions

Extension	File Type
.BU	Backup copy of a file, created by a text editor.
.CM	CLI command file, created by RDOS.
.DR	Directory, created by the CPART or CDIR command.
.LB	Library file, created by the LFE utility.
.LS	Program assembly listing file, created by the assembler.
.MC	Macro command file, created by the user.
.OL	Program overlay ilfe, created by the RLDR utility from .RB file.
.RB	Relocatable binary file, created by a program compiler or assembler.
.SV	Executable program file, created by RLDR from .RB file.
.TU	Tuning report file, created by the TUON command.
.VF	VFU line printer format file, created by the VFU utility.

Table 3-3. Suggested Filename Extensions

Extension	File Type
.FR	FORTRAN source file.
.JB	Batch job command file.
.SR	Assembly language source file.

Random, Contiguous, and Sequential Files

RDOS supports several types of file constructions — random, contiguous, and sequential for RDOS and DG/RDOS, and random and contiguous for DOS.

Random file construction is most common because it combines fast access time and efficient use of disk space, and it offers you the most flexibility for modifying file contents.

If you have an application that requires faster access time, RDOS, DOS, and DG/RDOS offer the alternative of contiguous files construction. For sequential access, you can create sequential files under RDOS and DG/RDOS.

File Storage and Retrieval

This section presents an overview of file storage and retrieval to provide a background for exploring the differences between random, contiguous, and sequential file constructions.

The storage space on a disk is divided into sections of 512 bytes each, called *blocks*. Each block has a physical address associated with its physical location on the disk.

RDOS divides a file into blocks, and stores the file using as many blocks as necessary to hold the entire file. RDOS assigns a *logical address* to each part of the file it stores in a separate block. A logical address *points to* (or, contains references to) the corresponding physical address of a block. Every file has its own set of logical addresses.

For example, we can represent the logical addresses for a file that takes up three blocks as BLOCK1, BLOCK2, and BLOCK3. BLOCK1 contains the actual physical address of the first block used to store the file. BLOCK2 contains the physical address of the next block used to store the file, which may be located wherever RDOS found a free block. BLOCK3 contains the physical address of the last block used to store the file.

Because our logical addresses BLOCK1, BLOCK2, and BLOCK3 contain the physical addresses of the blocks used to store the file, we don't have to concern ourselves with what those actual physical addresses are.

Random Files

RDOS stores a random file in blocks as the blocks become available on a disk. It assigns a logical address to each physical block location, and in addition, processes each logical block through an index.

Through use of this index, RDOS can jump from any block in a file to any other block, without having to access any of the blocks that may logically lie between them.

For example, for RDOS to move from the first block to the last block of a random file, it locates the address of the last block through the index, and then accesses only that block.

Random files represent the most flexible organization for the operating system and for the user. Use of disk space is efficient, because random files can use free blocks as they occur; and access time is efficient for file reads and writes, because random files allow for flexible movement from one block to any other block in the file.

Contiguous Files

Contiguous files are stored in disk blocks that are physically located right next to one another on a disk. When you create a contiguous file, you must assign to it a fixed number of blocks; you cannot expand the block count later. RDOS reserves the amount of space you specify for use by the contiguous file.

You can include only as much information in a contiguous file as its size can hold. If you do not use all of the space reserved for the file, you are left with some "wasted" space. However, contiguous files allow for fast access response time, because RDOS needs only to locate the first logical block of the file. RDOS can access the rest of the file quickly because the rest of the file exists on adjoining physical blocks.

Sequential Files

RDOS stores sequential files in physical blocks as the blocks become available on a disk. For each physical block used, RDOS assigns it a logical address. This logical addess also keeps track of the addresses of the last block used to store the file, and the next block that will be used to store the file.

From any given block in a sequential file, RDOS can move only to the addresses kept for that block, which are the addresses of the next or the previous block used to store the file.

For example, for RDOS to move from the first block to the last block of a sequential file, it finds the address of the second block of the file, accesses that block, finds the address of the third block of the file, accesses that block, and so on until it reaches the end of the file.

Sequential files are not as flexible as random files, because to move from one block to any other block, RDOS must access every block of the file that lies between the two. The use of disk space for a sequential file is slightly more efficient than for a random file, but this benefit is offset by its less flexible access capabilities. We recommend sequential organization only for very small files, such as 512 bytes or under.

Creating Files With CLI Commands

Three separate CLI commands control the creation of random, contiguous, and sequential files. Once you've created the files, you can insert data into them with the CLI XFER command, or by using a text editor. You can also create and modify (or edit) files using a text editor.

Creating Random Files

The CLI command you are likely to use most often to create a file is CRAND. CRAND creates a random file and assigns it the name you supply.

```
R
CRAND MYFILE ↵
R
```

This command creates a random file in your current directory called MYFILE.

```
R
LIST CRAND ↵
MYFILE.     0   D
R
```

When we list this file, LIST tells us that the file has a length of zero bytes (indicating that the file is empty), and the characteristic D (which indicates a random file). The length of the file will expand as we add data to it.

File characteristics are discussed in more detail at the end of the chapter under "File Characteristics and Attributes".

Creating Contiguous Files

To create a contiguous file, use the CCONT command, supplying a filename as the first argument, and the size of the file, in blocks, as the second argument.

```
R
CCONT CTGFILE 144 ↵
R
```

This command creates a contiguous file in your current directory called CTGFILE, and assigns it a length of 144 blocks.

```
R
LIST CTGFILE ↵
CTGFILE. 73728 C
R
```

The LIST command tells us that the file CTGFILE has a length of 73728 bytes (144 blocks), and the characteristic C (which indicates a contiguous file). The 144 blocks represent the space allocated to the file, and not the actual length of the information it contains.

Creating Sequential Files

The CLI command for creating sequential files is CREATE (sequential file organization is unavailable under DOS).

```
R
CREATE SEQFILE ↵
R
```

This command creates a sequential file in your current directory called SEQFILE.

```
R
LIST SEQFILE ↵
SEQFILE.     0
R
```

When we LIST SEQFILE, we can see that it has a length of zero. Sequential files have no file characteristic.

Inserting Data into Files With the XFER Command

The CLI XFER command lets you add data to the files you create. The format for using the XFER command to add text into a random file is

```
XFER/A/B $TTI filename
```

where:

XFER	is the command name.
/A	specifies an ASCII transfer (i.e., you want to type in some text, not binary code).
/B	appends the information into the file you have created.
$TTI	tells XFER you'll be typing in text from our keyboard (device name $TTI).
filename	tells XFER to add the information to this file.

When you enter the XFER command, XFER transfers everything you type at your keyboard into the specified file. To stop the transfer, enter a CTRL-Z.

The following example adds some information into the random file FILE1.

```
R
XFER/A/B $TTI FILE1 ⤸
This is my first RDOS file. ⤸
XFER accepts input from my keyboard. ⤸
That's enough for now. ⤸
CTRL-Z
R
```

FILE1 now contains the information we typed in, up to the CTRL-Z. Pressing CTRL-Z removes us from XFER and returns us to the CLI.

The XFER command has many more capabilities than described above. You can create files with XFER, and use XFER to transfer files to and from devices such as a line printer or a tape. See the XFER command description in the Command Dictionary.

Creating Files With a Text Editor

The most common way to create a file is by using a text editor. Your system comes with two text editors — EDIT and SPEED. The advantages of using a text editor are

- You can enter information into the file when you create it.
- You can move anywhere in the file to examine or make changes to the file.
- You can make changes to the information in the file by adding and deleting the information.

Text editors by default create random files; you can edit other types of files with a text editor.

Text editors come with their own commands for manipulating files. For this reason, a separate manual covers each editor. EDIT is described in the *RDOS/DOS Text Editor* manual. SPEED is described in the *RDOS/DOS Superedit Text Editor* manual. A tutorial introduction to each of these editors can be found in the *Introduction to RDOS* manual.

Displaying Files

This section introduces two commands that display the contents of files. The TYPE command displays the contents of a file at your terminal, and the PRINT command prints the contents of a file on the line printer.

To use TYPE, supply the filenames of one or more files with the TYPE command, for example

```
R
TYPE FILE1 ⤸
THIS IS MY FIRST RDOS FILE.
XFER ACCEPTS INPUT FROM MY KEYBOARD.
THAT'S ENOUGH FOR NOW.
R
```

TYPE displays the entire contents of a file. If your terminal is a display screen, a long file may scroll off the screen too fast for you to read it. Type a CTRL-S to pause the display, and when you're ready to see more, use CTRL-Q to resume it.

The PRINT command displays the contents of a file on the line printer (device name $LPT).

```
R
PRINT FILE<1,2,3> ⤸
R
```

This command prints the files FILE1, FILE2, and FILE3 on the line printer, as soon as the printer becomes available (i.e., when it completes any print requests submitted before yours). The PRINT command displays no confirmation message.

Specifying Groups of Files With Templates

Templates allow you to group together files that have similar elements in their filenames. When you use template symbols to represent any part of a filename argument, RDOS replaces the argument with all filenames that apply. RDOS template symbols are

- The dash (also known as hyphen), which can represent any group (or string) of legal filename characters, except the period character (.).

  For example, the command

  ```
  LIST C-
  ```

  lists all files that begin with C and don't have filename extensions. The command

  ```
  LIST E-.-
  ```

  lists all files that begin with E, whether or not they have filename extensions.

* The asterisk, which represents any single character, except the period character (.).

  For example, the command

  ```
  LIST FILE*
  ```

  lists any files with 5-character filenames, where the first 4 characters are FILE.

You can use templates anywhere within a filename. For example,

```
LIST -.SV
```

lists all executable program files (i.e., any files that have a .SV extension).

You can mix template symbols in filename arguments. For example,

```
LIST -.**
```

lists all files that have a two-character filename extension.

You can use templates only with the following CLI commands:

BUILD
DELETE
DUMP and LOAD
LIST
MOVE
PRINT
UNLINK

You can use templates only in filename arguments that refer to files in the current directory. You cannot use templates in pathnames to refer to files in another directory.

The ability to use templates can influence the way you name your files. For example, if you keep progress files for each month of the year, you could organize their filenames so that you can access them in a number of ways.

For example, you could give each progress file the same first two letters (say, PR for progress), followed by three characters for the month, and an extension that indicates the year. In this way you could access these files

- by specifying all progress files with the argument PR-.-
- by specifying all progress files for a particular month with the argument -APR.- or -JAN.-, for instance.
- by specifying all progress files for a particular year with the argument -.83 or -.84.

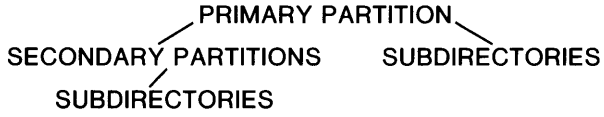
Templates can also be helpful when you aren't sure of the exact spelling of a filename.

Setting Up a Directory Structure

A *directory* is a special type of disk file that holds other files, much like a drawer in a file cabinet holds files. RDOS recognizes three kinds of directories, which can provide you with up to three levels in your *directory structure.* These are

- the primary partition
- secondary partitions
- subdirectories

The following diagram illustrates the possible levels in the directory structure.



The primary partition is the top level in the directory structure, and embodies the entire disk or diskette.

Secondary partitions are a subset of the primary partition, forming a second level in the structure. Subdirectories can be a subset of a primary partition, also forming a second level in the directory structure; or subdirectories can be a subset of a secondary partition, thus forming a third level.

Simple data files (as compared to directory files) can reside in any of these directories.

The Primary Partition

Before you create any directories on a disk, that disk has one directory, the *primary partition.* The primary partition comprises the total amount of file storage space on a given disk or diskette.

The name of a primary partition is the device name of the disk itself, such as DZ0 or DJ0. Often these terms are used interchangeably. For example, disk DZ0 is the same as primary partition DZ0; diskette DJ0 is the same as primary partition DJ0.

A primary partition forms the first level in your directory structure for a given disk or diskette. A primary partition can contain files, subdirectories, and secondary partitions.

Creating Directories

You can set up your own directory structure on a disk or diskette by creating directories. There are two types of directories you can create — secondary partitions and subdirectories. Each has its own characteristics, which we will explore shortly.

Secondary Partitions

A *secondary partition* is a type of directory that contains a fixed amount of contiguous disk space for file storage. You specify the amount of disk space, in blocks, when you create the secondary partition.

Secondary partitions are constructed as contiguous files. The space RDOS allocates to a secondary partition is made up of disk blocks located right next to each other on the disk. When you store files in a secondary partition, the files are stored within the constraints of this space. You cannot exceed the partition size that you specified when creating the partition.

RDOS forms a secondary partition by assigning it space from the primary partition. A secondary partition is a subset of the primary partition, and forms a second level in the directory structure. For example,

You can further break down a secondary partition into subdirectories.

Creating Secondary Partitions

To create a secondary partition, use the CPART command, and supply a name and a *block count* for the partition, as follows:

```
R
CPART PART1 256 ↵
R
```

In this example, we created a secondary partition called PART1 that has 256 contiguous blocks.

The block count allocates a fixed amount of disk storage space to the partition. The information contained in a secondary partition cannot exceed the amount of space allocated. (The section in this chapter, "Managing Files and Directories", discusses managing your storage space.)

Since directories are actually special types of files, we can list them with the LIST command:

```
R
LIST
PART1.DR    131072 CTY
R
```

The LIST command tells us that the command CPART PART1 256 created a file called PART1.DR, with 131072 bytes of storage space (equivalent to 256 blocks), and characteristics CTY (which indicate a secondary partition).

Characteristics of a file are explained further at the end of the chapter under "File Characteristics and Attributes".

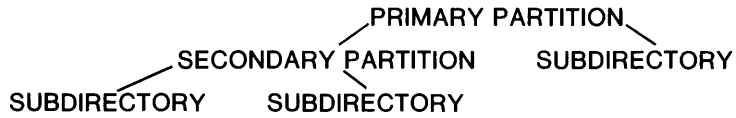
The CPART command automatically assigns the directory a .DR filename extension. You do not need to specify the .DR as part of the directory name, unless you use it as a filename argument.

Subdirectories

A *subdirectory* is a variable length directory that can be a subset of the primary partition or a subset of a secondary partition.

When a subdirectory is a subset of the primary partition, it forms a second level of directory structure:

When a subdirectory is created within a secondary partition, it forms a third level of directory structure:



Since a subdirectory cannot be further divided into other types of directories, we can see that the *RDOS directory structure can be a maximum of three levels deep.*

Subdirectories are constructed as random files that contain filenames and location information about each filename. Subdirectories can expand as you add more files or data. Because subdirectories are of variable length, RDOS doesn't set aside any physical disk space when you create a subdirectory, as it does for secondary partitions.

You can expand a subdirectory to occupy as many disk blocks as the subdirectory's parent partition can accommodate.

Creating Subdirectories

To create a subdirectory, use the CDIR command and supply a name for the directory.

```
R
CDIR MYDIR ↵
R
```

This command creates a subdirectory, MYDIR, which is a subset of the current partition.

When you create a subdirectory from a primary partition, it is a subset of the primary partition. When you create it from a secondary partition, it is a subset of that secondary partition.

We can list this file with the LIST command:

```
R
LIST ↵
MYDIR.DR      512 DY
R
```

LIST tells us that the command CDIR MYDIR created a file called MYDIR.DR with a length of 512 bytes and characteristics DY (which indicate a subdirectory).

The CDIR command automatically assigns the directory a .DR filename extension. You do not need to specify the .DR as part of the directory name, unless you use it as a filename argument.

Examples of a Directory Structure

Figures 3-1 and 3-2 illustrate RDOS directory structures. Figure 3-1 does this by means of a tree structure, so called because it resembles an inverted tree. Figure 3-2 shows the division of a primary partition into various directories and files.

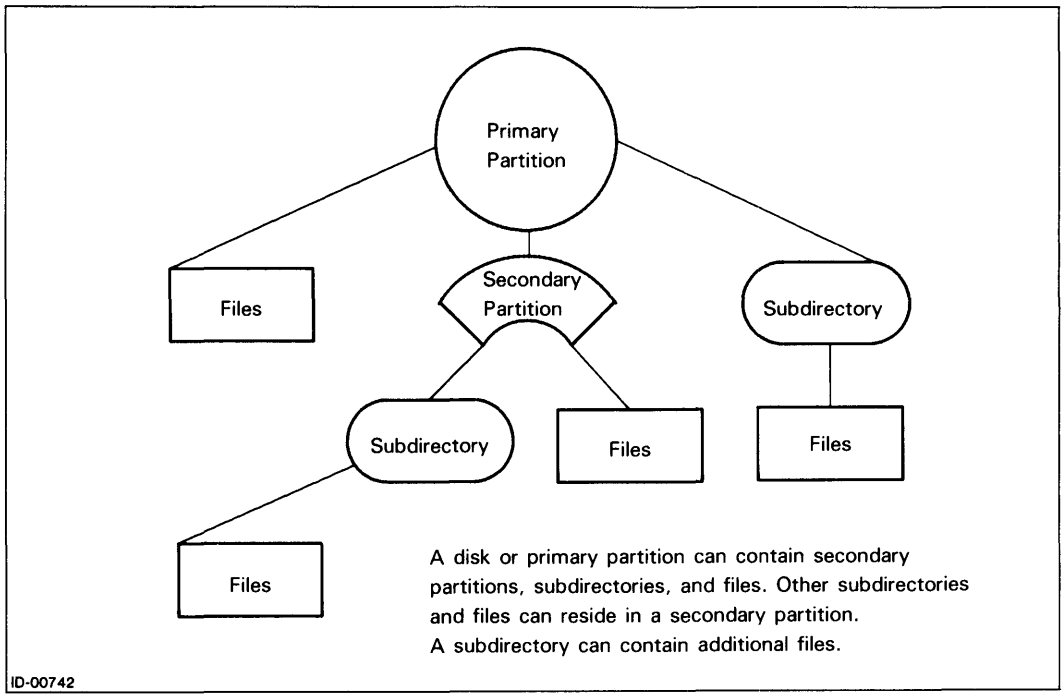


Figure 3-1. RDOS Directory Tree Structure

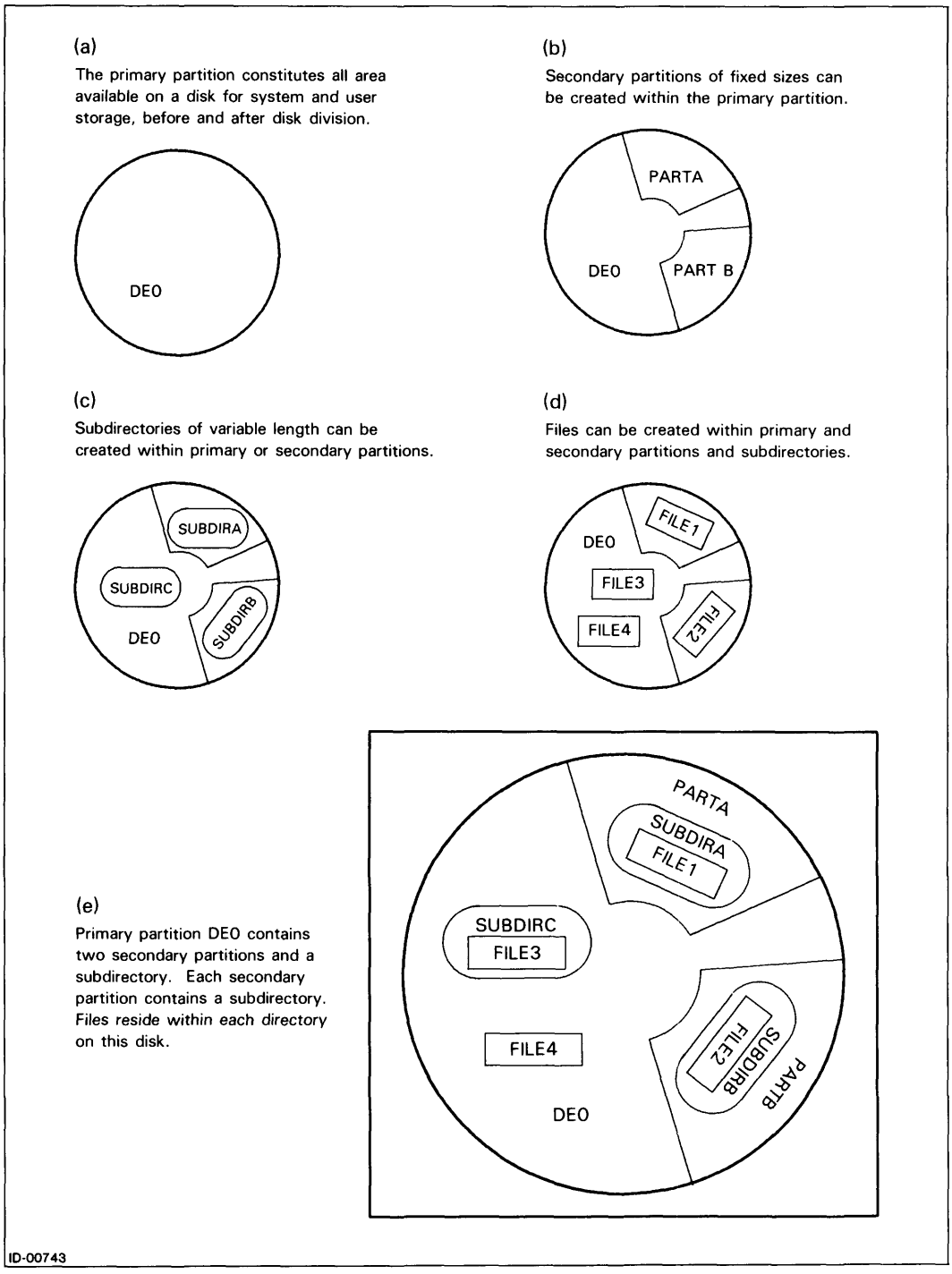


Figure 3-2. Formation of an RDOS Directory Structure

Summary of Directory Types

The following lists summarize the most important characteristics that differentiate directories.

Primary Partition

- fixed length (the size of the disk or diskette)
- constitutes the first level of a directory structure
- can contain secondary partitions, subdirectories, and files

Secondary Partition

- fixed length (size set by user at creation)
- constitutes the second level in a directory structure, being a subset of the primary partition
- can contain subdirectories or files

Subdirectory

- variable length
- constitutes either the second or third level of a directory structure, depending on whether it's a subset of the primary partition or a secondary partition
- can contain files

Accessing Directories

When you *access* a directory, the directory and its files must be available to you and to the operating system. RDOS does not recognize a directory until you open it, or *initialize* it, for access.

You can also designate any directory as your *current directory*; that is, the directory from which you are presently working. You have direct access to any files in your current directory.

Releasing a directory reverses the initialization, and designates a directory as unavailable for access.

You initialize and release directories, whether they are partitions or subdirectories, with the same commands. The only exception is the master directory. When you start up RDOS, the master directory becomes your current directory automatically. Releasing the master directory causes RDOS to release all other open directories, and shuts down an RDOS or DOS system.

Initializing Directories

Before you can access a file within a directory, the directory must be *initialized*; that is, made available to the operating system. The term "accessing a file" refers to any file operation, including displaying a list of files, displaying the contents of files, modifying files, or executing program files.

There are two ways to initialize a directory:

- by using the INIT command
- by changing the current directory with the DIR command

The INIT command initializes the directory you specify in the command line.

```
R
INIT MYDIR ↵
R
```

This command initializes directory MYDIR.

INIT opens a directory for access, without affecting the current directory. You can refer to files in an initialized directory but not current directory by naming the path to that directory in filename arguments. This is explained shortly in the section "Using Pathnames."

You can have many directories initialized at one time. The system generation program SYSGEN allows you to set a limit on the number of directories that can be initialized at once.

Changing the Current Directory

When you first start up your RDOS system, your current directory is the master directory. You can change your current directory with the DIR command.

```
R
DIR MYDIR ↵
R
```

The command DIR MYDIR places you in the subdirectory (or secondary partition) MYDIR. All file commands, such as LIST and TYPE, will now apply to the files in MYDIR (unless you specify otherwise in a pathname). *DIR automatically initializes your new current directory, and your previous current directory remains initialized.*

Releasing Directories

Releasing a directory closes it to input and output, and prohibits access to the files it contains. The RELEASE command releases a directory from the state of being initialized.

```
R
RELEASE MYDIR ↵
R
```

The command RELEASE MYDIR releases directory MYDIR.

A RELEASE command will undo both the INIT command and the DIR command. If you release your current directory, RELEASE also changes the current directory to the master directory. For example, if your current directory is MYDIR, a subdirectory of DP0, RELEASE places you in directory DP0.

You can use DIR or INIT to reinitialize a directory that has been released, with the exception of your master directory, as explained in the next section.

Releasing the Master Directory

When you release the master directory, RDOS (or DOS)

- releases all other currently open directories
- updates the disk with any new file information
- and shuts down the RDOS (or DOS) operating system

In the following example, the name of the master directory is DP0. When you release the master directory, it displays the message MASTER DEVICE RELEASED.

```
R
RELEASE DP0 ↵
MASTER DEVICE RELEASED
```

Under DG/RDOS, releasing the master directory does not shut down the system.

Releasing Disks and Diskettes

You *must* release a disk or diskette before physically removing it from its drive. Otherwise, the appropriate file information can't be updated, which leaves the disk or diskette in an inconsistent format.

To release a disk or diskette, you cite its device name as the primary partiton name, for example DP0 or DJ1.

```
R
RELEASE DJ1 ↵
R
```

Displaying Information About a Directory Structure

RDOS keeps information about the current state of your directory structure. The commands MDIR, GDIR, and LDIR display this information.

The MDIR command displays the name of the master directory.

```
R
MDIR ⤸
DP0
R
```

In this example, the master directory is DP0.

The GDIR (Get DIRectory) command displays the name of your current directory.

```
R
GDIR ⤸
DP0
R
```

DP0 is the current directory. If we change the current directory with the DIR command, GDIR displays the new current directory.

```
R
DIR SOLUTIONS ⤸
R
GDIR ⤸
SOLUTIONS
R
```

The current directory is now SOLUTIONS.

The LDIR command displays the name of the previous current directory.

```
R
LDIR ⤸
DP0
R
```

The previous current directory was DP0.

The MDIR, GDIR, and LDIR commands can assist you as move through your directory structure.

Using Pathnames

A *pathname* or directory specifier allows you to access a file or directory that is not in your current directory. A pathname comprises an argument string of directory names and a filename, listing all the directory names in the path from the current directory to the destination. The format of a pathname requires that each directory be separated by a colon (:), and that a colon precede the final destination, the filename. For example

```
DP3:MAURA:HERFILE
```

names two directories in the path to the destination, the file HERFILE.

Note that each directory you list in a pathname must be initialized to make it available to the operating system.

Example of Using Pathnames

Figure 3-3 depicts a directory structure with which we can experiment with pathnames.

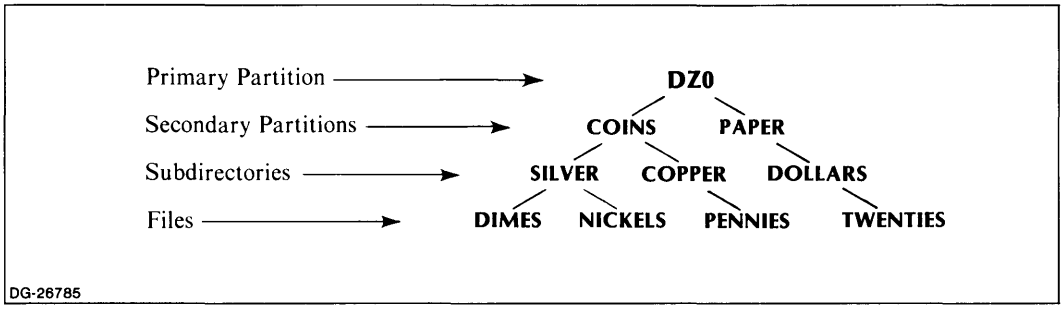


Figure 3-3. Directory Structure

Master directory DZ0 is our current directory; no other directories are initialized. The following commands initialize all of the directories in this directory structure; note that, by being named in the path, PAPER and COINS are initialized.

```
R
INIT PAPER:DOLLARS ↵
R
INIT COINS:SILVER ↵
R
INIT COINS:COPPER ↵
R
```

Our current directory is DZ0. We can list the file NICKELS without changing the current directory by naming them explicitly, as follows:

```
LIST COINS:SILVER:NICKELS
```

Assume our current directory is SILVER. We can list the file PENNIES as follows:

```
LIST COINS:COPPER:PENNIES
```

Assume the current directory is DOLLARS. We can change the current directory to COPPER as follows:

```
DIR COPPER
```

We did not have to specify all of the directories in the path from DOLLARS to COPPER, because COPPER has already been initialized.

Managing Files and Directories

The remainder of this chapter explains how to

- display the amount of space available on a disk
- rename, delete, and move files between directories
- link to a file in another directory
- set access privileges to files using file attributes

Determining the Amount of Free Disk Space

The DISK command displays the allocation of space for the current partition.

For a primary partition, DISK calculates the total amount of space on the disk. Blocks allocated to secondary partitons are listed as USED; add the number of USED and LEFT (unused) blocks for total number of disk blocks.

For a secondary partition, DISK calculates the amount of space allocated to it. For a subdirectory, DISK calculates the amount of space for its parent partition.

DISK reports the allocation of disk blocks (where one block equals 512 bytes) as described in Figure 3-4.

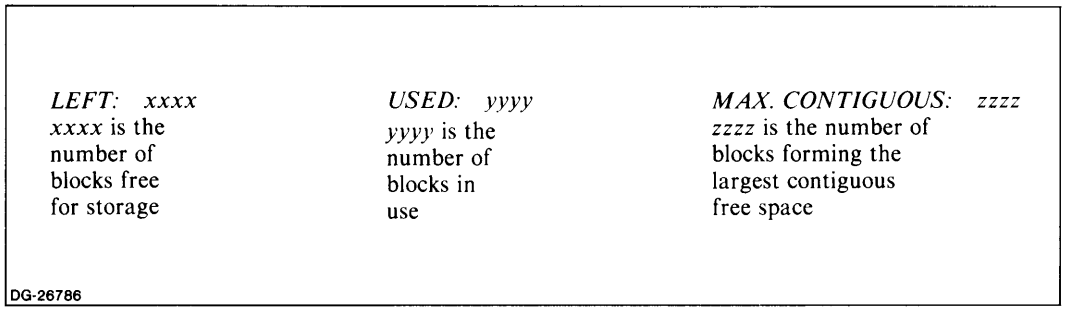


Figure 3-4. DISK Command Interpretation

If you add the number of blocks LEFT to the number of blocks USED, the result is the total amount of space allocated to the partition. If you want to create a secondary partition, or other contiguous file, you would check the number given for MAX. CONTIGUOUS to see how much contiguous space is available. For example

```
R
DISK ↵
LEFT: 1499   USED: 23161   MAX. CONTIGUOUS: 910
R
```

In this example, the current partition has 1499 free blocks available for use, 23161 are blocks currently in use, and 910 blocks form the largest amount of contiguous blocks available for use.

When you start to exhaust your file storage space on a disk, you need to decide which files you use most often, and which files (or perhaps entire directories) you no longer need daily or less frequently. You can then store old information on another device, such as a tape, and erase outdated files, thus freeing up space.

The CLI provides commands for moving files and directories to a variety of devices, and for transferring those files back to disk when they're needed. Chapter 5 discusses the alternatives at length. See also *RDOS, DOS, DG/RDOS Backup and Move Utilities* for more information.

Renaming and Deleting Files and Directories

The RENAME command lets you change the names of files and directories. You specify the new name as the first argument, and give the old name as the second argument.

```
R
RENAME GRAPHICS.DR PART1.DR ↵
R
```

This command assigns a new name of GRAPHICS.DR to secondary partition PART1.DR. This has no effect on the files in this directory — we only changed the directory's name.

The DELETE command erases a file or directory. When you specify its /C switch, DELETE will request *confirmation* from you before it deletes the file. Consider the following example.

```
R
DELETE/C FILEX.-↵
FILEX? n
FILEX.1? ↵*
FILEX.2? ↵*
R
```

The CLI displays each filename that matches the template FILEX.- and pauses for confirmation after each name. If you type a command line terminator, the CLI displays a asterisk (*) and deletes the file. If you type any other key, the CLI does not delete the file.

In this example, we deleted FILEX.1 and FILEX.2, but not FILEX. If we had not used the /C switch, the CLI would have erased all three files immediately, without requesting confirmation.

If you supply the filename of a directory to the DELETE command, DELETE erases the directory and all of the files and subdirectories contained in that directory.

Never use the DELETE command to erase a link file, because DELETE will erase the wrong file — it will delete the resolution file, not the link file. Use the UNLINK command instead. Link files are discussed at the end of this chapter under "Linking Files".

When you delete a file, you can't retrieve it unless you have made a backup copy of the file on another medium. Creating backup copies of your files regularly can save time, effort, and frustration if you inadvertently delete a needed file. (More on this in Chapter 5.)

You cannot delete a permanent file without first removing the attribute P from the file. The P attribute thus provides another method of protecting your files. See the last section in this chapter, "File Characteristics and Attributes" for more information.

Moving Files

The MOVE command allows you to move files from the current directory to another directory (the destination directory). The destination directory must be initialized for MOVE to be able to find it.

You supply the name of the destination directory as the first argument, and then specify the file or files you want to move. Including a /V (verify) switch with the MOVE command causes the CLI to list each file moved.

```
R
MOVE/V GRAPHICS XCHART.SV, CHART-.-↵
XCHART.SV
CHART1
CHART1A
CHARTFIN
R
```

This command moves the file XCHART.SV and all of the files matching the template CHART-.- from the current directory into the directory GRAPHICS.

Note that MOVE does not recognize permanent files (i.e., files with the attribute P), unless you include its /A switch. For example, the command

```
MOVE/A/V GRAPHICS C-.-
```

moves all files in the current directory that begin with a C to directory GRAPHICS, whether or not they are permanent files.

Often, you create a new directory when you see that a group of files you've been working on deserve their own directory. Let's look at an extended example of how you would do this by using a combination of the commands we've discussed so far.

The session in Figure 3-5 creates and initializes a directory called GRAPHICS, and moves four files from the secondary partition, USR$APPL, to the subdirectory GRAPHICS.

Once the files exist in GRAPHICS (which we verify by listing the files), we no longer need them in USR$APPL, so we delete them.

```
R
DIR USR$APPL ↵                          Get into dir USR$APPL
R
CDIR GRAPHICS ↵                         Create GRAPHICS directory
R
INIT GRAPHICS ↵                         Initialize GRAPHICS
R
MOVE/V GRAPHICS XCHART.SV CHART-.- ↵    Move desired files into directory GRAPHICS
XCHART.SV
CHART1
CHART1A
CHARTFIN
R
DIR GRAPHICS ↵                          Get into GRAPHICS
R
LIST ↵                                  List its files

XCHART.SV   54376   D
CHART1        945   D
CHART1A      1085   D
CHARTFIN    26386   D

R
DIR USR$APPL ↵                          Get into USR$APPL
R
DELETE/V XCHART.SV CHART-.- ↵           Delete the files that were moved to GRAPHICS
DELETED XCHART.SV
DELETED CHART1
DELETED CHART1A
DELETED CHARTFIN
R
```

DG-26787

Figure 3-5. Moving Files into a New Directory

Linking Files

Creating links to files in another directory allows you access to files without using pathnames, changing your current directory, or moving duplicate copies of the files into your current directory.

Links are often used to access CLI utilities residing in the master directory. Let's assume the master directory is the primary partition DP0. We want to be able to use the SPEED editor from our current directory (subdirectory GRAPHICS), and have RDOS access the file SPEED.SV that resides in the master directory.

The LINK command takes two arguments. The first argument assigns the link a name, and the second argument specifies the *resolution file*, the name of the file you want to reach. You enter the resolutiion file argument in the form of a pathname. For example

```
R
LINK SPEED.SV DP0:SPEED.SV ↵
R
```

This creates a link in GRAPHICS to the SPEED program file in DP0.

It often makes sense to give the link file the same name as the file you want to access, so that you don't have to remember two different names. This is the only case in which assigning duplicate names to files is good practice.

When you list a link file, the LIST command displays the name of the link and its resolution:

```
R
LIST SPEED.SV ↵
SPEED.SV @:SPEED.SV
R
```

LIST displays the name of the link file, SPEED.SV, and its resolution file, @:SPEED.SV. The symbol @ denotes the parent directory of the current directory. If the resolution file does not reside in the parent directory of the current directory, LIST replaces the @ symbol with the name of the directory; for example, DE0:SPEED.SV.

To remove a link, use the UNLINK command with the link filename. For example,

```
R
UNLINK SPEED.SV ↵
R
```

This removes the link file SPEED.SV from your current directory.

NOTE: It is very important that you remove link files with UNLINK, and not DELETE. DELETE will erase the resolution file.

File Characteristics and Attributes

RDOS keeps information about all files on a disk in a system file directory called SYS.DR. The information kept about your files is called a *user file definition* (UFD). Part of the UFD for a file contains the file's characteristics and attributes.

File Characteristics

RDOS uses *file characteristics* to identify particular properties of files — for example, whether a file has random data block organization, or is a link or directory file. The characteristics are codes that RDOS assigns when creating a file, according to the type of file you're creating. You cannot later change a file's characteristics. Table 3-4 describes RDOS file characteristics.

Table 3-4. File Characteristics

Characteristic	Meaning
C	Contiguous file.
D	Random file.
L	Link file.
T	Partition file. (Partitions also have characteristics C and Y.)
Y	Directory file. (Partitions, subdirectories.

The LIST command displays the characteristics for each file listed.

```
R
LIST ↵

FILE1.          68  D
FILE2.          50  D
GRAPHICS.DR     512 DY
R
```

In this example, FILE1 and FILE2 have the characteristic D, indicating a random file, and file GRAPHICS.DR has the characteristics DY — D for a random file, and Y for a directory.

File Attributes

File *attributes* are codes assigned by you or by RDOS that restrict or permit file manipulation or access. The CHATR (CHange ATtRibute) command allows you to assign and remove a file's attributes. Table 3-5 describes RDOS file attributes.

Table 3-5. File Attributes

Attribute	Type of File	Comment
A	Attribute-protected file	You cannot change the file's attributes, including attribute A.
N	Resolution-protected file	You cannot access the file by way of a link.
P	Permanent file	You can't delete or rename files with P set. Note that the P attribute is not recognized by the BUILD, DUMP, LIST, LOAD, or MOVE commands unless you include the /A switch.
R	Read-protected file	You cannot display or read a file with R set.
S	Executable program file (.SV)	Allows file execution. The RLDR utility assigns S when creating a file.
W	Write-protected file	You cannot edit or modify a file with W set.
?	User-definable attribute	You can use this to mark files for your own record-keeping.
&	User-definable attribute	You can use this to mark files for your own record-keeping.

Note that you must use attribute A with caution. For example, if you combine attributes P (permanent — you cannot delete or rename) and A (attribute protected — you cannot change its attributes), you would not be able to DELETE that file, unless you erase the entire disk with a full initialization.

The LIST command displays file characteristics and attributes, as follows:

```
R
LIST FILE10 ↵
FILE10.     0 DN
R
```

The letter D is the file characteristic for a random file, and the letter N is the file attribute that prevents links to the file FILE10.

The CHATR (change attribute) command takes two or more arguments. The first is a filename; the second and subsequent arguments specify the desired attributes. You precede each attribute with a plus sign (+) to assign the attribute, or with a minus sign (−) to remove the attribute.

```
R
CHATR FILE1 +W ↵
R
LIST FILE1 ↵
FILE1.      214 DNW
R
```

This example assigns the write-protect attribute, W, to FILE1, so that you cannot modify the file without first changing the attribute. The LIST command shows that FILE1 has the characteristic D, and attributes N and W. The command

```
CHATR FILE1 −W
```

removes the write-protect attribute from FILE1.

You can write a program to give values to user-definable attributes ? and &, and then assign these attributes to a file with the CHATR command.

The CHLAT (CHange Link ATtribute) command changes a file's link access attributes. Link access attributes are the same attributes described in Table 3-5, but they take effect only when a file is the target, or resolution file, of a link. Link access attributes have no effect on direct access of a file (i.e., access from within the file's own directory or through use of a pathname).

You use CHLAT in the same way you use CHATR. The LIST command displays link access attributes by appending them to any existing attributes and characteristics with a slash (/).

```
R
CHLAT DISPLAY.SV +PW ↵
R
LIST DISP-.- ↵
DISPLAY.SV      634 D/PW
R
```

This example assigns the attributes P and W as link access attributes to file DISPLAY.SV. The LIST command shows that DISPLAY.SV has the random file characteristic, D, and link access attributes P (permanent) and W (write-protect). DISPLAY.SV cannot be deleted or modified through a link file, but you can modify or delete it from within its own directory, or through use of a pathname.

End of Chapter

Chapter 4
Grouping CLI Commands: Macro and Indirect Files

Macro command files and indirect command files enable you to use a single command to execute a set of CLI commands. By storing a set of CLI commands in a macro or indirect file, you can run involved or often-used procedures quickly and simply. You can modify the commands in these files, as needed, with a text editor.

Macro files are the simpler of the two types of files. A macro file can contain only valid command lines. To execute a macro file, you type in the name of the macro file, as you would a CLI command.

Indirect files have further capabilities. An indirect file can contain complete command lines, or just the command arguments. You specify an indirect file in a command line by enclosing its filename in @ symbols. You can invoke an indirect file as a command, or as an argument to a command.

When you invoke a macro or indirect file, the CLI reads in and executes each line in the file sequentially. The CLI responds to the commands in the file exactly as if you had typed each command to the CLI interactively.

See also the DO utility in the Command Dictionary (Chapter 10) for executing a macro command file with variable arguments that you specify in the DO command line.

Macro and indirect files can call other macro or indirect files. The section "Improving Macro and Indirect Files" discusses this.

Creating a Macro File

You can assign a macro file a name of your choice, but you must include the filename extension .MC. The .MC enables the CLI to recognize the file as a macro file when you invoke it.

We will be using the XFER command to create files for our examples; a text editor such as EDIT or SPEED may be more appropriate for you.

The following example creates a macro file called ALLDONE.MC to execute a number of commands that you might normally use when you have finished using the CLI for the day.

```
R
XFER/A $TTI ALLDONE.MC ⤸
DELETE/V -.BU ⤸
LIST/S/E ⤸
DISK ⤸
DIR %MDIR% ⤸
^Z
R
```

The command line XFER/A $TTI ALLDONE.MC creates a file called ALLDONE.MC from the input we subsequently type in at the keyboard (device $TTI). XFER closes the file when we enter a CTRL-Z.

The commands in macro file ALLDONE.MC

- delete any backup files you have created
- list all the files in your current directory
- display information about the amount of available file storage space
- make the master directory the current directory

Invoking a Macro File

To *invoke* (or execute) a macro file, simply enter its macro name, as you would a CLI command. A *macro name* is the filename of a macro file, minus the .MC extension.

The example below invokes the ALLDONE.MC macro file by typing in ALLDONE as a command.

```
R
ALLDONE ⤸
DELETED FILE3.BU

FILE1.      68      D    01/06/84      16:37   01/06/84    [006564]    0
FILE2.      50      D    01/06/84      13:24   01/09/84    [006565]    0
PROGX.       0      D    01/13/84      10:42   01/13/84    [006566]    0
LEFT: 1499 USED: 23161 MAX. CONTIGUOUS: 910
R
```

The CLI executes each command in macro file ALLDONE.MC.

You do not need to specify a macro file's .MC extension because the CLI assumes that the first word in a command line is either a CLI command, a macro (.MC) file, or an executable program (.SV) file. (The "Command Interpretation Order" section ahead in this chapter explains this hierarchy.)

However, if you do not include the .MC when you create the file, the filename you type would be meaningless to the CLI, and you would receive a message such as

FILE NOT FOUND: ALLDONE.SV

Creating an Indirect File

An indirect file may have any filename of your choosing. It does not require an extension, because the CLI recognizes an indirect file by the way in which you invoke it.

The following example creates an indirect file called PROJFILES.

```
R
XFER/A $TTI PROJFILES ⤸
PROJ83:1QTR83 PROJ84:<1QTR,JAN,FEB,MAR>84 ^Z
R
```

The file PROJFILES contains two filename arguments. The first argument refers to the file 1QTR83, in the directory PROJ83. The second argument refers to four files in the directory PROJ84 (files 1QTR84, JAN84, FEB84, and MAR84). (The use of angle brackets as a shorthand method of specifying arguments is introduced in Chapter 2.)

This file does not contain complete command lines; we will invoke it as an argument to another command.

Invoking an Indirect File

You invoke an indirect file by enclosing its filename with two @ symbols. Since our indirect file, PROJFILES, contains only arguments, we can invoke the file only as an *argument* to a command.

```
R
LIST @PROJFILES@↵

PROJ83:1QTR83     142          D
PROJ84:1QTR84     161          D
PROJ84:JAN84      234          D
PROJ84:FEB84      306          D
PROJ84:MAR84      271          D
R
```

In this example, the @ symbols cause the CLI to search for the file PROJFILES, and use its contents as arguments to the LIST command. We could use @PROJFILES@ as an argument to a number of commands, such as PRINT, MOVE, and DELETE.

Note that the CLI recognizes *all* characters in macro and indirect files, including the command line terminators we used to enter each line into the file.

We can also invoke the file ALLDONE.MC as an indirect file, since it has all the components of an indirect file. All we need do is enclose the filename ALLDONE.MC with @ symbols.

```
R
@ALLDONE.MC@ ↵
```

This command has the exact effect of typing in ALLDONE as a macro file. However, we needed to include the .MC extension, because the CLI searches for a filename when it encounters an @ symbol. In this example, the @ symbols caused the CLI to search for and execute the contents of the file ALLDONE.MC.

Improving Macro and Indirect Files

You can improve your macro and indirect files by making use of the following features:

- CLI variables
- the MESSAGE command
- calling macro and indirect files from within other macro and indirect files

CLI Variables

The CLI stores some information about your system in a set of eight predefined variables, described in Table 4-1. When you include a variable in a command line, RDOS replaces the variable with the current value of the item that the variable represents.

Table 4-1. CLI Variables

Variable	Value
%DATE%	Today's date, in the form mm-dd-yy.
%GCIN%	The device name for your keyboard (terminal input), e.g. $TTI.
%GCOUT%	The device name for your display screen or terminal output, e.g. $TTO.
%GDIR%	The name of the current directory.
%LDIR%	The name of the previously current directory.
%MDIR%	The name of the master directory.
%FGND%	The character "F" if the CLI is running in the foreground; a null value if it is not.
%TIME%	The time of day, in the form hh:mm:ss.

The following command line uses the variable %DATE% to list all files in the current directory that were modified today:

```
LIST/S/E %DATE%/A
```

This command line is general enough to include in a macro or indirect file. It will list all files modified on the current date, even as the actual current date changes.

Displaying Messages in Command Files

The CLI MESSAGE command is especially useful in macro files and indirect files for displaying text during the execution of the file. The MESSAGE command accepts text strings of up to 72 characters as arguments. For example

```
R
MESSAGE MESSAGE displays whatever you type.
MESSAGE displays whatever you type.
R
```

When you enclose your text argument in quotation marks ("), the CLI displays every character you type literally, except for the quotation marks. When you do not use quotation marks, the CLI interprets and executes any special command line symbols, including angle brackets, commas, and all characters listed in Table 2-2. This can work to your advantage. For example, the following line reports the value of a CLI variable.

```
R
MESSAGE The device name for your keybord is %GCIN%
The device name for your keyboard is $TTI
R
```

Note that the CLI replaced the variable %GCIN% with its real value. If you enclose %GCIN% in quotation marks, the CLI treats it as a character string, as it does any other string in quotation marks. For example

```
R
MESSAGE "The value of %GCIN% is" %GCIN%
The value of %GCIN% is $TTI
R
```

The command MESSAGE/P pauses the current execution and displays the following message:

STRIKE ANY KEY TO CONTINUE

This can be useful if you write a macro that requires the user to perform some procedure, such as inserting a diskette or mounting a tape, before the macro continues.

For example, we include the following lines in a macro file:

```
MESSAGE Transfer complete.
MESSAGE Insert next diskette.
MESSAGE/P
```

When we invoke the macro file, these lines display the following:

Tranfer complete.
Insert next diskette.
STRIKE ANY KEY TO CONTINUE

Calling Files From Within a File

You can call other macro and indirect files from within a macro or indirect file. This allows you to write command or argument files in small, flexible modules, and then put them together as needed. In addition, you can make changes to one file without modifying any of the files that use that one file.

To call another macro or indirect file, you need only call it in the file exactly as you would call it interactively. For example

```
R
XFER/A $TTI MASTERFILE.MC ↓
PRINT @PROJFILES@
PRINT @MOREFILES@
ALLDONE
^Z
R
```

This file contains commands to print the files contained in the indirect files PROJFILES and MOREFILES, and then to execute the ALLDONE macro.

Command Interpretation Order

The CLI can recognize four types of items as the first word in a command line:

- indirect files (enclosed in @ symbols)
- CLI commands
- macro files (with a .MC filename extension)
- program files and CLI utilities (with a .SV filename extension)

Macro, indirect, and program files are types of files that the CLI can *execute*. The command interpretation order determines what the CLI will execute in a case where files or commands share the same name.

When the CLI interprets the first word in a command line, it searches for these items in the following order:

1. When the CLI comes upon the indirect file symbol, @, it searches for the filename enclosed by the @ symbols, and executes that file as an indirect file. If no files in the current directory match the specified filename, the CLI returns a *FILE NOT FOUND* message.
2. If there are no @ symbols, the CLI attempts to match the name with one of its own commands.
3. If the name is not a CLI command, the CLI attempts to match the name with *name.MC* from the current directory. (That is, the CLI tries to execute *name* as a macro file.)
4. If the CLI cannot find *name.MC*, it searches for *name.SV* in the current directory. (That is, the CLI tries to execute *name* as a program file.) .SV files include user programs and CLI utilities.

This means, for example, that if a program (.SV) file has the same name as a macro (.MC) file in the same directory, the CLI will always find the macro file first and execute it. You can eliminate the problem by renaming one of the files, or you can direct the CLI to search specifically for the program file by including a .SV extension when you type in the program name.

If the CLI cannot match the first word of a command line with any of the items listed above, the CLI sends you a FILE NOT FOUND message. For example,

```
R
FIRSTWORD ↵
FILE NOT FOUND: FIRSTWORD.SV
R
```

The CLI checked for a command, then for a macro file (.MC), and finally for a program file (.SV), and could not match FIRSTWORD with any of these.

Command Line Execution Order

The CLI executes the components of a command line in the following order:

1. Text strings in quotation marks. (MESSAGE command).
2. Indirect files and variables (as they appear from left to right on the line).
3. Angle brackets.
4. Parentheses.
5. Numeric switches.
6. Text strings outside of quotation marks (including the first word of the command line, arguments, and global and local alphabetic switches, in the order they appear on the command line).

Numbers 1 through 5 are all items the CLI will need to *expand* the command line. Once the CLI takes care of the expansions, it can begin executing the entire command line from left to right.

End of Chapter

Chapter 5
Using Input/Output Devices

Input/output devices, called *I/O devices* or *peripheral devices* are auxiliary components of a computer system. All I/O devices perform some combination of the following functions:

- provide storage for information
- send (*output*) information to computer memory or to another device
- receive (*input*) information from memory or from another device

When you move information between two I/O devices, the act is known as a file transfer operation.

About RDOS Peripherals

RDOS supports a number of different types of peripheral devices, described in the next few sections.

Each device is assigned a device name. To direct input or output to a device from the CLI, you need only specify its name as an argument on the command line. RDOS devices are set up as files that automatically access the corresponding device in the format required by that device.

In addition to the devices that RDOS supports by default, you can define your own devices and incorporate them as part of the operating system.

Types of I/O Devices

Most I/O devices fall under two general categories: magnetic peripherals and hard-copy peripherals.

Magnetic peripherals write data to and read data from magnetic storage media such as disks, diskettes, and magnetic tape. The section "Magnetic Storage Media" in this chapter discusses magnetic peripherals, and commands and issues that apply to their use.

Hard-copy peripherals write data to or read data from paper storage media such as paper tape and punched cards, or print files or graphics on paper ("hard-copy"). The "Hard-copy Peripherals" section in this chapter discusses hard-copy peripherals and related CLI commands.

Two I/O devices do not fit neatly into a category: terminals and multiplexors. This chapter discusses them under their own headings.

RDOS Input/Output Device Names

Table 5-1 lists the RDOS device names, shows which device is attached to which *controller*, and describes each device. Each device has a unique name that can include elements denoting the device's controller, and unit number on that controller.

RDOS provides two I/O controllers for most types of devices: a primary and a secondary controller. A *controller* is hardware within the computer that is set up to handle input and output between the CPU (Central Processing Unit) and a device or group of devices.

In Table 5-1, a 1 in the Controller column indicates that the named device or devices are attached to the primary controller; a 2 indicates that the device or devices are attached to the secondary controller.

For disks and tapes, more than one unit (or device) may be attached to a single controller. In such cases, the device name contains an *n* to represent the range of unit numbers available — the Description column gives the ranges. When a range is a number from 0 to 3, up to 4 of that type of device can be supported on that controller (i.e., device unit numbers 0, 1, 2, and 3).

Table 5-1. RDOS Device Names

Device Name	Controller	Description
$CDR $CDR1	1 2	Punched or mark sense card reader.
CTn CT1n	1 2	Cassette tape drive; n can be 0 to 7.
DAn	1 2	Disk drive; n can be 0 to 3. n can be 4 to 7.
DEn	1 2	Hard disk or diskette drive; n can be 0 to 3. n can be 4 to 7.
DH0 DH1	1 2	Hard disk drive (removable disk).
DH0F DH1F	1 2	Hard disk drive (fixed disk).
DJn	1 2	Diskette drive; n can be 0 to 3. n can be 4 to 7.
DPn	1 2	Disk drive (removable disk); n can be 0 to 3. n can be 4 to 7.
DPnF	1 2	Disk drive (fixed disk); n can be 0 to 3. n can be 4 to 7.
DM0 DM1	1 2	Disk drive.
DSn	1 2	Disk drive; n can be 0 to 3. n can be 4 to 7.

(continues)

Table 5-1. RDOS Device Names

Device Name	Controller	Description
DZn	1	Disk drive; n can be 0 to 3.
	2	n can be 4 to 7.
$DPI		Input dual processor link.
$DPO		Output dual processor link.
$LPT $LPT1	1 2	Line printer.
MCAR MCAR1	1 2	Multiprocessor communications adapter receiver.
MCAT MCAT1	1 2	Multiprocessor communications adapter transmitter.
MTn MT1n	1 2	Magnetic (reel-to-reel) tape drive; n can be 0 to 7.
$PLT $PLT1	1 2	Plotter.
$PTP $PTP1	1 2	Paper tape punch.
$PTR $PTR1	1 2	Paper tape reader.
QTY:n		Multiplexor. Supports ALM, ASLM, QTY, ULM, and USAM multiplexors. *
$TTI $TTI1	1 2	Terminal keyboard (input).
$TTO $TTO1	1 2	Terminal printer or display screen (output).
$TTP $TTP1	1 2	Teletypewriter punch.
$TTR $TTR1	1	Teletypewriter reader.

*Multiplexor acronyms:

ALM	Asynchronous Line Multiplexor
ASLM	Asynchronous/Synchronous Line Multiplexor
QTY	Asynchronous Data Communications Multiplexor
ULM	Universal Line Multiplexor
USAM	Universal Synchronous/Asynchronous Multiplexor

(concluded)

Terminal Input and Output

Most of the information on terminal input and output has been covered in Chapter 2 of this manual. Your terminal serves as your primary means of communicating with the system by entering CLI commands to execute commands, utilities, and user programs which may in turn call on other peripherals. You can control terminal input and output operations with the control characters discussed in Chapter 2.

There are two types of terminals: display screen terminals and typewriter terminals.

The device names for the input component (i.e., the keyboard) of a terminal are $TTI and $TTI1. The device names for the output component (display screen or typewriter-like carriage) of a terminal are $TTO and $TTO1.

$TTI and $TTO compose primary terminal, attached to the primary terminal controller. The primary terminal is often referred to as the system or operator *console*. $TTI1 and $TTO1 compose the secondary terminal, attached to the second controller, and it serves as a foreground terminal. (Foreground-background processing is introduced in Chapter 6.)

You may attach other terminals via multiplexor lines (discusssed next), but these terminals cannot run the CLI.

Note that for most devices, RDOS supplies an end-of-file. However, for $TTI input, you must indicate an end-of-file with a CTRL-Z.

Mulitplexors

RDOS multiplexors can accommodate up to 64 half duplex lines for supporting terminals and modems. RDOS supports a number of multiplexor types, including:

ALM Asynchronous Line Multiplexor

ASLM Asynchronous/Synchronous Line Multiplexor

QTY Asynchronous Data Communications Multiplexor

ULM Universal Line Multiplexor

USAM Universal Synchronous/Asynchronous Multiplexor

The device name for a multiplexor of any type is QTY. Each multiplexed line of a QTY device corresponds to a filename in the form

QTY:xx

where xx is a number from 0 to 64. Numbers 0 through 63 represent a physical QTY line. QTY:64 is a software setup that allows you to communicate with the multiplexor drive.

Note that RDOS does not supply an end-of-file for QTY input; you must indicate an end-of-file with a CTRL-Z.

Refer to the *RDOS System Reference* manual and the *How to Load and Generate RDOS* manual for more information on multiplexors.

Magnetic Storage Media

Magnetic storage media — disks, diskettes, and magnetic tapes — have magnetic surfaces that allow compact storage and rapid retrieval of data. Each type of magnetic media has a corresponding I/O device that writes and reads its information, as illustrated in Table 5-2.

Table 5-2. Storage Media and Their I/O Devices

Storage Media	I/O Devices
Disks	Disk drives
Diskettes	Diskette drives
Magnetic tapes	Tape drives
Cassette tapes	Cassette tape drives

Magnetic media are the most versatile of available methods of storage. Other methods of storing data include paper tape and punched cards, discussed in this chapter under the section "Hard-Copy Peripherals."

Each command or utility we discuss in this chapter moves files in some way, and has its own requirements about what can be moved and to where. The command line syntax for each command or utility specifies all argument requirements; the following sections examine some of the possible combinations of arguments for disk and for tape.

Using Disks and Diskettes

Disks and diskettes store information efficiently for interactive use, or for backup. Disk and diskette drives are designed to access areas of a disk for either read or write operations very rapidly.

The basics of using disks and diskettes are covered in Chapter 3, and we'll review some of that information here before explaining how to transfer information between disks and diskettes.

This chapter uses the terms disks and diskettes interchangeably, except when referring directly to a particular device name. Because it helps to clarify which files are being moved where, we often use a disk to contain the original files, and diskettes to contain backup copies of files.

Initializing Disks, Diskettes, and Directories

Before you can access the information on a disk, the disk must be initialized, or recognized by the system and thus open for I/O operations. To initialize a disk, supply the disk's device name as an argument to the INIT command. For example

```
R
INIT DJ0 ↵
R
```

initializes the primary partition of the diskette in drive DJ0.

To initialize a directory, use its name as the argument, or use a pathname if the directory's parent directory is not already initialized. For example

```
R
INIT PRODUCTS:GRAPHICS ↵
R
```

initializes secondary partition PRODUCTS (if it has not already been initialized), and subdirectory GRAPHICS.

The DIR command, in contrast to the INIT command, initializes a directory *and selects it as the current directory*. Use a pathname argument if the directory's parent directory is not already initialized. For example

```
R
DIR PRODUCTS:GRAPHICS ↵
R
GDIR ↵
GRAPHICS
R
```

initializes secondary partition PRODUCTS, initializes subdirectory GRAPHICS, and makes GRAPHICS your current directory. The GDIR command displays the name of the current directory.

Many of the commands and utilities we discuss in this chapter involve file movement using the current directory as a reference point. Often, before issuing any of these commands, you must select the appropriate current directory.

It is important to release disks — diskettes too — before removing them from the disk drive. The RELEASE command updates all file directories, ensuring that all blocks on the disk are accounted for. The command

```
RELEASE DJ1
```

releases diskette DJ1 so that you can remove it from its drive. If you inadvertently remove a disk from the drive before releasing it, reinsert the disk in the drive and release it, as RDOS may still have the information it needs to perform the release.

Specifying Disk Arguments in Command Lines

When you need to refer to a disk in a command line, you must name the *device* (or drive) that holds that disk.

For example, argument DJ1 refers to any diskette in diskette *drive* DJ1. Device name DJ1 temporarily serves as the primary partition name for that diskette. A primary partition name is not a fixed filename like other directory names; it serves to tell RDOS where to locate the disk or diskette.

When you want to refer to files or directories on a disk, you use a pathname argument. For example, the argument

```
DJ0:MYDIR:FILE1
```

is a pathname argument that refers to the file FILE1, in directory MYDIR.DR, on the diskette in drive DJ0.

The argument

```
DJ0:MYDIR
```

refers to directory MYDIR.DR on the diskette in drive DJ0.

When you need to refer to a group of filenames, you can specify each one in your command line, or you can build an indirect file to contain the filenames, and supply the indirect file as a filename argument. Chapter 4 covers using indirect files.

Using Magnetic Tape

Magnetic tapes are used to create backup copies of disk files. The advantages of using magnetic tape for backup are:

- A single tape holds a large amount of information.
- Tape read and write operations take a small amount of time to complete.

You can store any number of disk files on a single tape file. For example, one tape file could contain the entire contents of a disk (i.e., all files and directories on the disk). The maximum number of tape files allowed on any one tape is 100, but in practice, the number of tape files you use on any one tape is one to six or eight.

There are two kinds of magnetic tape — reel-to-reel and cassette. From a CLI standpoint, reel-to-reel and cassette tape are identical, except for their device names. They use the same format for reading and writing data, and have the same file organizations.

The following are the device names for magnetic tape (note that the f represents a file number):

MTn:f	Reel-to-reel tape drive attached to the first tape I/O controller.
MT1n:f	Reel-to-reel tape drive attached to the second tape I/O controller.
CTn:f	Cassette tape drive attached to the first tape I/O controller.
CT1n:f	Cassette tape drive attached to the second tape I/O controller.

n is a number from 0 to 7 that represents the unit number of the tape drive. You can attach up to 8 tape drives to a single tape I/O controller.

f is a number from 0 to 99 that specifies a file number on the tape. You can store up to 100 tape files on a single tape.

Magnetic Tape File Organization

Tape files vary in length according to how much information you store in each one. You can store any number of disk files on a single tape file.

You place tape files on a tape in numerical order, starting with file number 0. File number 99 is the last permissible file.

When the CLI creates a tape file, it writes the data to tape, and places two end-of-file (EOF) marks at the end of the tape file. For each additional tape file you write, the CLI backs up and erases one of the EOF marks from the end of the preceding tape file, writes the data to tape, and again places two EOF marks at the end.

In this way, the CLI recognizes a single EOF mark as a separator between tape files, and a double EOF mark as the end of the last tape file. The double EOF mark is not static; as you tack additional tape files onto the last tape file, the double EOF marks become single EOF separators, and the last file written is followed by a double EOF mark.

Initializing and Releasing a Tape Drive

Before you can access a tape, you must initialize the tape by supplying the tape's device name to the INIT command. INIT rewinds the tape to its starting point.

```
R
INIT MT0 ↵
R
```

This command initializes tape drive MT0, and rewinds the tape on drive MT0.

```
R
INIT CT0 ↵
R
```

This command initializes tape drive CT0, and rewinds the cassette tape on drive CT0.

When you are finished with a tape, use the RELEASE command to rewind the tape and release the drive.

Using INIT and RELEASE while the tape is on the tape drive is important because both commands rewind the tape, resetting the system tape file pointer to 0.

If you are using a brand new tape, use INIT with the /F switch to prepare it for use. INIT/F writes two EOF tape marks to the beginning of the tape, which allows you to overwrite any previous data (of any format).

Accessing Magnetic Tape Files

To specify a tape file as an argument in a CLI command line, supply the tape device name, a colon, and the number of the tape file you wish to read from or write to. For example, the argument

```
MT0:0
```

specifies the first file (file 0) of the tape on tape drive MT0.

Once a file is written, RDOS assigns the number of the next file, and that file is considered empty. If you put some data into file 0, RDOS knows where to find file 1. However, if you put data into file 0 and issue a command to store data into file 2 with an argument such as MT0:2, RDOS responds with the message *FILE DOES NOT EXIST, FILE: MT0:2.*

Tape files are written to tape sequentially. When you overwrite a tape file, all subsequent tape files on the tape are lost. For example, if you have three tape files on a tape, and you overwrite the second tape file (file 1), RDOS places a double EOF mark at the end of that file, signifying the end of data on the tape. Hence, the contents of the third tape file are lost.

In this way, it is easy to reuse tapes. By taking a tape and writing data to file 0, RDOS can no longer know the existence of any previously existing information.

Some commands and utilities allow you to retrieve selected disk files from a single tape file. This requires a combination of arguments on the command line, including the name of the tape file and the disk filename arguments. For example, the command line

```
LOAD MT0:0 XCHART.SV FCR.SV
```

Loads to disk the files XCHART.SV and FCR.SV, which are stored on tape file MT0:0.

Copying Files Disk to Disk: The MOVE Command

Chapter 3 introduced the MOVE command as a way to move copies of current directory files to another directory, called the *destination directory*. This destination directory can be another disk or diskette, as illustrated in the following example.

```
R
DIR BASKETBALL ↵
R
INIT DJ0:TEAMS ↵
R
MOVE/V/R DJ0:TEAMS ↵
CELTS.
76ERS.
LAKERS.
R
RELEASE DJ0 ↵
R
```

This example selects a current directory, BASKETBALL, and initializes the destination directory, TEAMS, on diskette DJ0.

Because the MOVE command line does not include any filename arguments, MOVE moves all files in the current directory that apply to the specified switches to destination directory DJ0:TEAMS.

The /V (verify) switch displays the names of each file moved at the terminal. The /R (recent) switch compares the creation dates of files that have the same name in the current and destination directories, and then replaces destination directory files with those files with the same name in the current directory that have more recent creation dates.

The RELEASE command releases diskette DJ0 so we can remove it from the diskette drive.

When MOVE encounters a directory (.DR) file in the current directory, it ignores the file. MOVE does not move directories subordinate to the current directory.

The new files MOVE created in directory DJ0:TEAMS are duplicates of the original files in the current directory BASKETBALL.

Creating Backup Copies of Files and Directories

This section discusses commands and utilities that use a *dump format* to create backup copies of disk files and directories.

The *dump format* stores all file and directory information in a very compact form in order to fit as much information as possible on the backup disk or tape.

When you need the backup files, you *load* the files from the backup medium to a disk. *Load operations* interpret the dump format on the backup medium and use it to reconstruct (or restore) the original file and directory structures on disk.

Each command or utility that performs a dump operation has a complementary load command or utility. To restore backup copies, you must use the command that corresponds to the command you used to create the backup.

It is good practice to mark all backup disks and tapes with information about the dump session, and then store the backup medium in a safe place.

Commands and utilities that fall under the dump/load category are summarized in Table 5-3.

Table 5-3. Dump and Load Software

Commands	Description
DUMP and LOAD	CLI commands for single volume disk and tape backups.
DDUMP and DLOAD	CLI utilities for fast backups to multiple diskettes.
FDUMP and FLOAD	CLI utilities for fast backups to multiple tapes.
IMOVE	DG/RDOS utility for disk and tape backups. (Global switches specify whether the operation is a dump or a load.)

Using the DUMP and LOAD Commands

The DUMP command stores files and directories subordinate to your current directory, in dump format, on a disk, diskette, magnetic tape, or paper tape. (Paper tape is discussed in a later section entitled "Hard-Copy Peripherals.")

The LOAD command restores files and directories, previously dumped with the DUMP command, from disk, diskette, or magnetic tape to the current directory.

DUMP and LOAD are versatile in that they allow you to select files to be moved based on template specifications and file creation dates.

Before entering a DUMP or LOAD command, you must select the appropriate current directory, and initialize the disk or tape you plan to use.

The syntax for the DUMP command line is

DUMP*[/global-sw] [dest:]*dumpfilename *[filename...] [arg/local-sw]*

DUMP's global switches allow you to specify options such as /A to include files that have the attribute P (permanent), and /V to display the names of the dumped files at your terminal.

The *[dest:]*dumpfilename argument specifies a destination device or directory, and a dumpfilename, which assigns a name to the single file that DUMP creates to contain all the dumped files.

An example of a *[dest:]*dumpfilename argument for a disk or diskette is

DJ0:GRAPHICS.BU

where DUMP creates file GRAPHICS.BU on diskette DJ0 to contain all the files dumped. You need to label the outside of the diskette with this dumpfilename, as you will need to specify the dumpfilename to load the dump files back to disk.

An example of a *[dest:]*dumpfilename argument for a magnetic tape is

MT0:0

where DUMP creates tape file 0 on tape MT0 to contain all the files dumped.

DUMP's *[filename...]* arguments are optional, and may include templates when they refer to files in the current directory.

If you specify no filename arguments, DUMP dumps all files and directories subordinate to the current directory. (That is, for each file that is a directory (.DR) file, DUMP dumps that directory and all of its files.)

The *[arg/local-sw]* local switch arguments enable you to select files based on their creation dates, or to exclude files that match a specified template.

The following example dumps all files created after a specified date from the current directory to a diskette.

```
R
DIR PRODUCTS:GRAPHICS ↵
R
INIT DJ0 ↵
R
DUMP/A/V DJ0:GRAPHICS.BU 3-1-84/A ↵
  CHART1.
  CHART2.
  CHART4.
  PLOT4SYM.
  XCHART.SV
  XCHTHELP.SV
R
```

First, we select GRAPHICS as our current directory, and initialize the destination diskette. Then we issue the DUMP command. Global switch /A includes files with attribute P (permanent) in the dump, and /V displays the names of each dumped file at the terminal.

DJ0 is the destination diskette, and GRAPHICS.BU is the dumpfilename. Because we did not specify any filename arguments, DUMP dumps all files from the current directory that fit the criteria set up by global and local switches.

The local switch argument 3-1-84/A specifies that only files whose creation dates are on or after March 1, 1984 should be dumped.

The files listed in the output produced by the /V switch reside in dump format in file GRAPHICS.BU on diskette DJ0.

The following example dumps specified files from the current directory and files from a subordinate directory to a magnetic tape.

```
R
DIR PRODUCTS ↵
R
INIT MT0 ↵
R
DUMP/A/V MT0:0 -.SV GRAPHICS.DR ↵
  CALCWKLY.SV
  CALCDLY.SV
* GRAPHICS.DR
    CHART1.
    CHART2.
    CHART4.
    PLOT.SV
    PLOTSYM.SV
    PLOT4SYM.
    XCHART.SV
    XCHTHELP.SV
R
RELEASE MT0 ↵
R
```

First, we select PRODUCTS as our current directory, and initialize tape drive MT0.

All files in the current directory that match the template -.SV are dumped to file 0 of tape MT0. Because GRAPHICS.DR is a directory file, all the files it contains are also dumped. Note that the output produced by the /V switch reflects the directory levels of the dumped files.

The LOAD command line is similar to the DUMP command line, except the first argument is called the *source*, because you are loading files from the device (source) to your current directory:

LOAD*[/global-sw] [source:]*dumpfilename *[filename...] [arg/local-sw]*

The LOAD command has its own global switches that help you to load files selectively. For example, the /O (overwrite) switch overwrites any file in the current directory with the same filename as a file to be loaded. The /R (recent) switch compares creation dates between files that have the same name, and loads the backup files only if the backup files have creation dates more recent than those of the files in the current directory.

In addition, LOAD has an /N (no load) global switch, which executes the command line without performing the actual load, so that you can check the effect of your command line.

The following example loads the files we dumped to tape in the previous DUMP example.

```
R
DIR PRODUCTS ↵
R
INIT MT0 ↵
R
LOAD/A/V MT0:0 -.SV GRAPHICS.DR ↵
  CALCWKLY.SV
  CALCDLY.SV
* GRAPHICS
    CHART1.
    CHART2.
    CHART4.
    PLOT.SV
    PLOTSYM.SV
    PLOT4SYM.
    XCHART.SV
    XCHTHELP.SV
R
RELEASE MT0 ↵
R
```

This example loads files and directories from tape MT0, file 0, to our current directory, PRODUCTS. Note that PRODUCTS must be a secondary partition; otherwise it would not be able to accept the directory structure we loaded and we would have received the message *DIRECTORY DEPTH EXCEEDED.*

Fast Dump and Fast Load Utilities for Tape and Disk

There are two sets of fast dump and fast load utilities — DDUMP and DLOAD, for diskette storage, and FDUMP and FLOAD, for tape storage.

These utilities share the following features:

- You do not specify filename arguments for the dump or load. By default, all files in your current directory are dumped, or you can specify a global switch to include directories subordinate to the current directory in the dump.
- If the dump fills a single diskette or tape, the utility prompts you to insert another diskette or to mount another tape, and continues the dump. The utility continues in this fashion until all files are dumped.

  You must load from mutiple volume diskettes or tapes in the same order in which you dumped to them. The utilities prompt you to set up the next diskette or tape in the sequence, as needed. Therefore, it is important to mark the outside of backup diskettes and tapes with the appropriate dump information, such as dates and sequence numbers so you can load them back in order.
- These utilities are faster than using the DUMP and LOAD commands. They do not have to spend time selecting individual files — everything in the current directory is dumped, and everything must be loaded back.
- The utilities initialize the devices you specify automatically.

See the DDUMP and DLOAD and FDUMP and FLOAD descriptions in this manual, or refer to the *RDOS, DOS, DG/RDOS Backup and Move Utilities* manual for details.

Other I/O Utilities

IMOVE is a DG/RDOS-only utility that performs dumps and loads between disk, diskette, and magnetic tape. IMOVE is a versatile utility, compatible with other MOVE utilities on AOS-based operating systems. For more information about IMOVE, check the IMOVE command description in this manual, and refer to *Using DG/RDOS on DESKTOP GENERATION™ Systems* or *RDOS, DOS, DG/RDOS Backup and Move Utilities.*

FCOPY is a DG/RDOS-only utility that allows you to duplicate diskettes. See the FCOPY command description in this manual, and refer to *Using DG/RDOS on DESKTOP GENERATION™ Systems.*

COPY is a DOS-only utility that duplicates DOS diskettes. For COPY information, see the COPY command description in this manual.

FCOPY and COPY do not use dump formats; you can use the new duplicate diskette just as you use the original.

A group of stand-alone utilities (i.e. running without the CLI or the operating system) called the BURST utilities facilitate backup. The BURST utilities take a disk image of an entire disk and transfer it to either disk, tape, or diskette, depending on the utility. The BURST utilities accompany one or more RDOS-based operating systems, depending on the utility. Because the BURST utilities are not CLI utilities, we don't document them here. They're documented in *RDOS, DOS, DG/RDOS Backup and Move Utilities.*

Hard-Copy Peripherals

Hard-copy peripherals use some form of paper to store or display information, in contrast to magnetic media, which store information on a magnetic surface.

There are two types of hard-copy peripherals:

1. Devices that produce output on paper, such as line printers and plotters.
2. Devices that perform input, output, and storage, such as paper tape and card devices.

Hard-copy peripherals produce output with great speed, yet they are very slow compared to the rate of output of the CPU. To prevent bottlenecks — a situation where the CPU must wait for a peripheral to finish processing before it can continue to send data to that device — hard-copy peripherals *spool* information.

Spooling temporarily stores information sent to a hard-copy peripheral on disk, if the peripheral is busy. Each file sent to the peripheral is put in a queue, and is processed when the peripheral becomes available, in the order in which it was sent (i.e., first in, first out). Spooling is discussed further on in the section "Spooling Output to Slower Devices."

Using a Line Printer

Line printers produce output on paper, printing one line at a time. The device name for a line printer is $LPT or $LPT1, where $LPT is the first line printer (attached to the first line printer controller), and $LPT1 is the second line printer (attached to the second line printer controller).

Printing Disk Files: PRINT and FPRINT

There are two CLI commands for printing the contents of disk files on a line printer: PRINT and FPRINT.

The PRINT command, introduced in Chapter 3, prints the contents of text files (i.e., nonbinary files) on the line printer. For example, the command line

```
PRINT FASTCAR SPORTSCAR MODELT ↵
```

displays the contents of files FASTCAR, SPORTSCAR, and MODELT on the line printer.

If you attempt to use PRINT to display compiled program files, the output shows an seemingly unorganized group of characters, as the file is in machine (computer) readable form. To print nontext files, use the FPRINT command.

The FPRINT command prints the contents of either text or nontext files. By default, FPRINT translates the contents of a file to octal integers and displays the file at the terminal.

By using FPRINT command switches, you can print the file at the line printer (/L), and change the default octal number base to decimal, binary, or hexadecimal. The command line

```
FPRINT/L PROGRAM.RB ↵
```

prints the contents of file PROGRAM.RB at the line printer.

Producing Line Printer Output With /L Global Switches

Many RDOS commands and utilities support a /L switch that sends to the line printer any output the command or utility produces. The /L switch is useful for keeping records on paper, especially when the output is too long to fit on one terminal screen, or when you create backup files and wish to store a written record of the dump with the dump medium. Output produced by a /L switch is often referred to as a *listing file.*

Creating Line Printer Forms with the VFU Utility

The VFU Vertical Format Utility allows you to set up the line printer to produce forms according to your specifications. By invoking the utility with different switches, you can

- create a file in which you specify settings for the form, such as columns, blank lines, and vertical length.
- read the VFU file into line printer memory.
- execute or disable the VFU file formats.

When you print files with VFU file formats in place, the files are printed according to the formats you specify. VFU enables you to produce forms for a variety of business and office needs.

For a complete description of VFU, see the VFU utility description in the Command Dictionary in this manual, or refer to *RDOS/DOS Sort/Merge and Vertical Format Unit Utilities.*

Using a Plotter

Plotters are devices that produce plots, graphs, and charts on paper. The device name for a plotter is $PLT or $PLT1, where $PLT is the first plotter, attached to the first controller, and $PLT1 is the second plotter, attached to the second controller.

Your graphics application provides instructions on using the plotter. There are no specific CLI commands that direct output to a plotter. Use the CLI spooling control commands SPEBL, SPDIS, and SPKILL to manage spooling to a plotter.

Using Paper Tape Devices

Paper tape is a long strip of paper on which data is stored in the form of punched holes. There are two devices for using paper tape, as shown in Table 5-4.

Table 5-4. Paper Tape Devices

Device Name	Controller	Description
$PTP $PTP1	1 2	Paper tape punch — punches data to paper tape.
$PTR $PTR1	1 2	Paper tape reader — reads paper tape data to a file on another device.

The CLI commands PUNCH and BPUNCH create paper tape copies of files. The files may reside on any device (e.g., disk files or magnetic tape files). Use PUNCH for text files, and BPUNCH (binary punch) for nontext files.

PUNCH and BPUNCH accept filename arguments for the files you wish to transfer. They automatically send the output to $PTP.

You can use the CLI XFER command to write files to, or read files from, a paper tape, by specifying the $PTP or $PTR device names in the command line.

Using Card Devices

Punched or mark sense cards are similar to paper tape in that they store information on cards in the form of punched or marked holes. A card punch writes data to cards; a card reader (device names $CDR and $CDR1) reads data from cards to a file on another device.

RDOS has reserved device names for card readers, but not for card punches. The section "User-Defined Devices," below, gives some information on how to go about incorporating a device such as a card punch as part of your RDOS system.

You can use the XFER command to transfer card files to another device by specifying the $CDR device name in the XFER command line.

Spooling Output to Slower Devices

Three CLI commands control spooling operations for output devices:

- SPEBL (enables spooling operations).
- SPDIS (disables spooling operations).
- SPKILL (cancels any spool files currently in the queue).

For all hard-copy peripherals and also for terminals ($TTO and $TTO1) and any user-defined devices defined as spoolable, RDOS automatically enables spooling.

To disable spooling, which forces the CPU to send information directly to the device, use the SPDIS command supplying the device name as an argument. For example,

```
SPDIS $LPT
```

disables spooling to the line printer. You must explicitly re-enable spooling with the SPEBL command. For example,

```
SPEBL $LPT
```

resumes spooling operations for the line printer.

The SPKILL command deletes all spool files currently in the spool queue. SPKILL provides a way to cancel all pending spooled output jobs. Spooling resumes automatically with the next command that sends output to the named device.

For example, the command

```
SPKILL $LPT
```

cancels all output currently in queue to be printed. A command such as PRINT MYFILE automatically resumes spooling operations to the line printer.

User-Defined Devices

RDOS can support up to 64 user-defined I/O devices. You can incorporate the device into the operating system at the source level, or you can identify it to the operating system at runtime.

Refer to the *RDOS System Reference* manual and to the *How to Load and Generate RDOS* manual for information.

End of Chapter

Chapter 6
Executing Foreground and Background Programs

RDOS architecture allows you to separate memory into two areas or *grounds*, called *foreground* and *background*. Each ground can run a separate set of programs. More efficient use of system resources can result from running programs in two grounds.

Foreground and background programs run independently of one another. Each ground has its own user memory area, and has its own task scheduler for controlling program tasks.

The two grounds share system resources such as CPU time and I/O devices, so that while one ground is idle, the other can use the system. The grounds can have equal priority, or you can assign higher priority to the foreground. This allows the foreground to run programs requiring fast system response, while the background runs less urgent programs, making use of extra system time that would otherwise go unused.

Setting Up the Foreground and Background

When you first start up RDOS, no foreground or background areas exist. All user memory is available to any program you wish to run. You create a foreground and a background by specifying a portion of total available user memory to the background; RDOS allocates the remainder to the foreground.

Setting up the foreground and background is different for mapped and unmapped systems. Mapped systems are those that have the Memory Management and Protection Unit, which provides hardware protection to separate foreground and background memory areas.

On mapped systems, you can set up foreground and background areas with CLI commands. On unmapped systems, you specify the appropriate addresses for a foreground memory area when you load your program(s).

Setting Up Foreground and Background on Mapped Systems

Mapped systems provide an absolute hardware boundary between foreground and background, providing each ground with complete ZREL (page zero relocatable) and NREL (normal relocatable) memory areas.

The SMEM command creates the two grounds. SMEM allocates the amount of memory you specify to the background, and allocates the remainder of available user memory pages to the foreground. You specify the amount of memory in pages, where one page of memory equals 1024 words. The GMEM command displays the current memory allocations for each ground. For example

```
R
GMEM ↵
BG: 341    FG: 0
R
```

Before we set up foreground and background, we check the total number of available memory pages with the GMEM command. GMEM reports the total, 341, as belonging to the background (BG: 341). However, when the total for foreground is zero (FG: 0), no foreground exists, and therefore no background/foreground areas exist.

```
R
SMEM 121 ↵
R
```

We use SMEM to allocate 121 of the 341 available pages to the background. Foreground and background areas now exist; RDOS created the foreground with the 220 remaining available pages.

```
R
GMEM ↵
BG: 121    FG: 220
R
```

When we check the memory allocations with GMEM, this time the background has 121 pages and the foreground has 220 available pages.

Executing Foreground and Background Programs on Mapped Systems

Once you allocate the desired amount of memory to the background with the SMEM command, you can execute programs in either background or foreground.

The background runs the CLI, and you execute programs as you normally do. The following line executes a program called SPROING.SV in the background:

```
R
SPROING ↵
```

The CLI command EXFG (execute foreground) executes a program in the foreground. For example, to execute the program SPROING.SV in the foreground, we use the following command line:

```
R
EXFG SPROING ↵
```

We type this line from the CLI running in the background to execute program SPROING in the foreground. We can also execute utilities in the foreground by preceding the utility name and any needed arguments with the EXFG command.

Setting Up Foreground and Background on Unmapped Systems

Unmapped systems support foreground and background areas, but the user achieves this separation by specifying ZREL and NREL foreground boundary addresses when loading the program to be run in the foreground. This is a software separation, and the system may fail if the program in one ground tries to use more space than is allocated to it.

You prepare an assembled source program for foreground execution by specifying the starting ZREL and NREL addresses in the RLDR command line. For example, the line

```
R
RLDR 13000/F 250/Z PORGIE ↵
```

loads program PORGIE. RLDR loads the ZREL portion of PORGIE starting at location 250 (octal), and the NREL portion of PORGIE starting at 13000 (octal).

Executing Foreground and Background Programs on Unmapped Systems

Once you create the foreground boundaries for your program with RLDR, you execute the program in the foreground with the EXFG (execute foreground) command. You issue the EXFG command from the background. For example,

R
EXFG PORGIE ↵

This command line executes program PORGIE within the foreground boundaries previously set up by RLDR. Without the EXFG command, RDOS could not set up the seperate ZREL and NREL (page zero relocatable and normal relocatable) areas specified in program PORGIE.

The background area runs the CLI. To execute programs in the background, type in their names as you normally would. Keep in mind that your background program has less space available to it when you are executing a foreground program.

See Figure 6-1 for a representation of a foreground/background memory configuration. See Chapter 7 or the RLDR command description in this manual for information on using the RLDR utility.

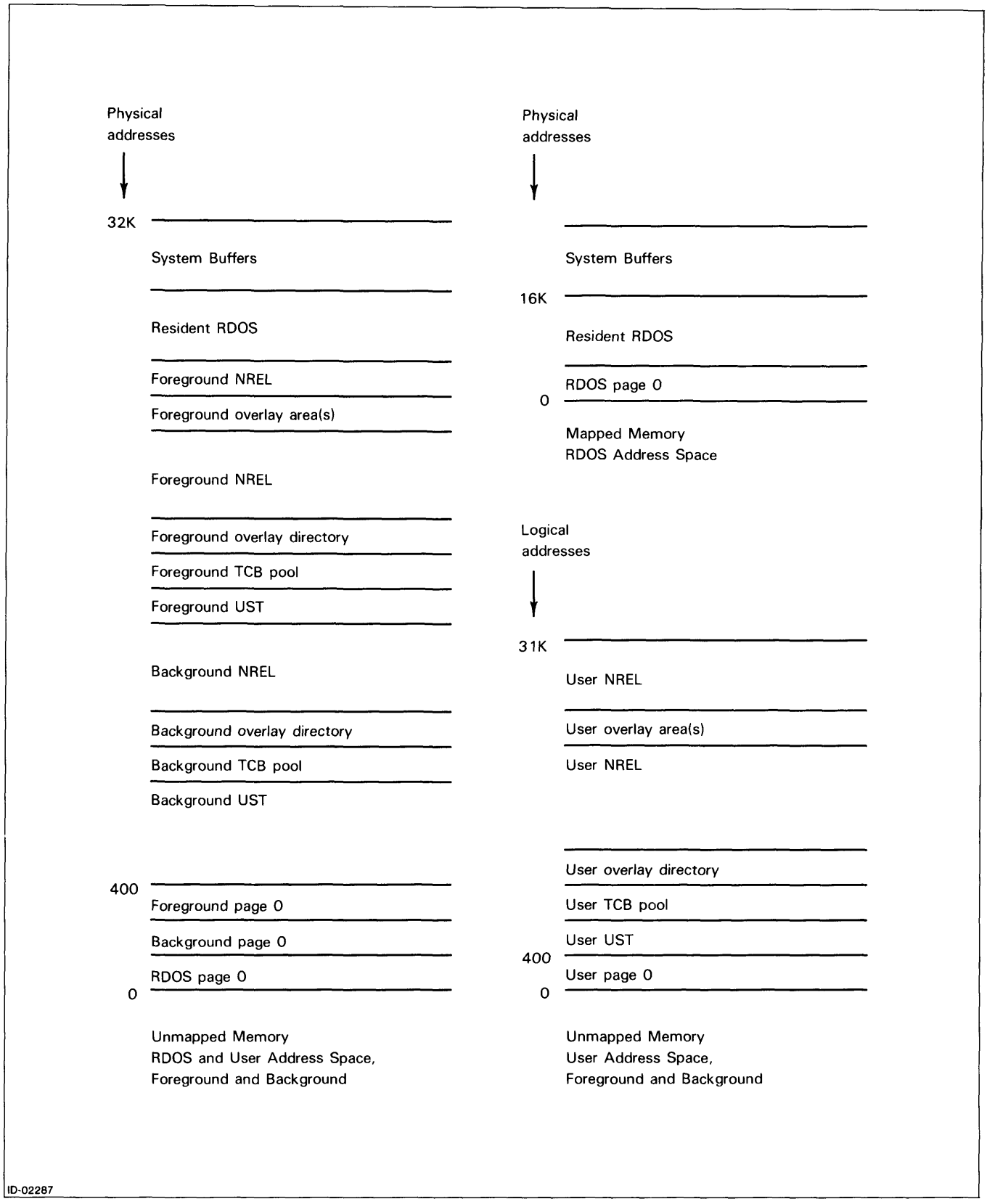


Figure 6-1. Foreground-background Memory Use

Communication Between Foreground and Background Programs

Foreground and background programs run in separate portions of memory, but share system resources. RDOS allows communication and resource sharing between foreground and background as described below.

When running two grounds, the system console ($TTI and $TTO) interacts with the background, and another terminal (usually $TTI1 and $TTO1) interacts with the foreground program.

To find out if a program is currently executing in the foreground, type the FGND command from the background. You receive one of two responses: *NO FOREGROUND PROGRAM RUNNING* or *FOREGROUND PROGRAM RUNNING.*

Programs running in foreground and background may pass information to each other through system calls, or may communicate over the data channel with a Multiprocessor Communications Adapter (MCA).

You can use system calls to *checkpoint* a background program — to stop temporarily the background program, run something else, then recall the background program. For instance, if your background and foreground programs are functionally dependent upon each other, you can issue a call from the foreground that causes the background to process information that the foreground program needs. Refer to *RDOS System Reference* for more information.

Foreground and background programs can initialize the same devices, but cannot simultaneously read or write data to the same device. The first program to read or write to a device will be successful.

Foreground and background programs can open and read the same file, but writing requires an exclusive open to the file; thus only one program can have write privileges to a file at one time.

Foreground and background programs can initialize the same directory. Either program can release a directory without affecting the other's use of that directory.

If one ground releases a directory that the other ground has initialized, the message

DIRECTORY SHARED: directoryname

appears at the system console. This is an informational message only, not an indication of error. This directory remains initialized for one ground, and becomes released for the other.

If you attempt to release the *master* directory from the background while a foreground program is running, you receive a message such as

FOREGROUND ALREADY RUNNING: DZ0

To release the master directory from the background, you must first terminate any foreground program still running.

If the foreground program releases the master directory, RDOS releases it for the foreground, and the master directory remains initialized for the background.

Terminating the Foreground Program

Your foreground program terminates and returns control to the CLI (which is running in the background). You can force the foreground program to terminate by issuing a CTRL-F from the background CLI.

For mapped systems, you need to use SMEM to restore all memory to a single ground. For example,

```
R
FGND ↵
NO FOREGROUND PROGRAM RUNNING
R
GMEM ↵
BG: 121   FG: 220
R
SMEM 341 ↵
R
GMEM ↵
BG: 341   FG: 0
R
```

The previous example finds that no foreground program is executing with FGND, and checks the memory allocation for foreground and background with GMEM.

The SMEM 341 command allocates all available user memory pages (121 plus 220) to the background, and RDOS allocates the remainder of available memory to the foreground. Since the amount of foreground memory pages is equal to 0, foreground ZREL and NREL no longer exist, and all memory is restored to a single ground.

End of Chapter

Chapter 7
Invoking System and Program Utilities

The CLI is made up of CLI commands and CLI utilities. A utiltity is a program that performs a complex set of functions.

While you execute CLI commands, you invoke CLI utilities. There is no apparent difference between entering a command line for a CLI command and entering a command line for a CLI utility, although the CLI treats the two differently.

The differences between CLI commands and utilities can be summarized as follows:

- Utility programs run a specialized set of functions. For example, when you invoke a text editor utility such as EDIT, you enter the text editor environment and use the utility's editing commands to work with a file.
- While CLI commands are immediately available to RDOS memory, CLI utilities are programs that are called in from disk, as needed.

Invoking a CLI Utility

Utilities employ the same command line syntax rules as CLI commands. They take arguments, global switches, and local switch arguments. Utility switches allow you to change defaults, customizing the utility's operation to meet your current requirements.

To invoke a utility, type the utility name as a CLI command, and complete the command line with any needed arguments and switches. The utility then finds and executes the utility files it needs to complete its operation.

Utility Program Files

Utilities are programs that exist on disk as executable program files (.SV files).

A single utility may be packaged with a number of files. For example, a utility that is very large may come with a program overlay (.OL) file in addition to the .SV file. A utility that displays interactive error messages may come with an error (.ER) file. These files will share the same filename — usually the name of the utility. All the needed files should exist together in the same system directory.

Most users store CLI utilities in the master directory, but you have the option of storing them in another directory when you generate your system. Wherever they are kept, CLI utilities should be readily available to you at all times. You can use the LINK command to create links to utilities in their directory.

Documentation for CLI Utilities

This chapter summarizes all CLI utilities, and groups them in the following categories:

- High-Level Language Utilities
- Program Development Utilities
- Debugging Utilities
- File Backup Utilities
- System Utilities

The chapter also includes a final section summarizing CLI commands that are often used in context with the program development and system functions described here.

Most utilities are documented in separate manuals. These manuals are listed under the appropriate utility in the command dictionary (Chapter 10) in this manual. The command dictionary description for each utility contains

- the command line syntax
- an explanation of arguments, switches, and options
- a description of functions, detailed according to the requirements of the utility
- one or more examples showing how to invoke the utility

In general, the command dictionary descriptions assume that you have read or will read about the utility's full functionality in the appropriate manual.

High-Level Language Utilities

RDOS supports a number of high-level languages, including Business BASIC and Extended BASIC, FORTRAN IV and FORTRAN 5, ALGOL, and Interactive COBOL.

The high-level language utilities summarized in Table 7-1 either invoke a language environment, as is the case with BASIC and Interactive COBOL, or compile source programs written in a high-level language.

You create source programs with a text editor such as EDIT or SPEED. There are two steps in preparing a finished source program for execution: *compiling* the program, and *loading* the program.

A program compiler translates your source code into machine-readable (assembly) code, and assigns relocatable (or nonfixed) memory addresses. The compiler flags any errors it encounters while compiling, and reports errors and program statistics to the terminal, a line printer, or a file. The compiler produces a relocatable binary file that has the filename of your source program and the .RB filename extension.

You use the .RB file produced by the compiler to load your program with the RLDR utility. RLDR produces an executable program from your .RB file that has the same filename and a .SV extension. RLDR also flags any errors it finds and produces listing information. (See the section "Program Development Utilities" for more about RLDR.)

You execute your program by typing in the filename of your .SV file as a CLI command.

Table 7-1. CLI Commands for High-Level Languages

Command	Description
ALGOL	Compiles an ALGOL source program.
BASIC	Invokes the BASIC language interpreter.
CLG	Compiles, loads, and executes a FORTRAN IV source program.
FORT	Compiles a FORTRAN IV source program.
FORTRAN	Compiles a FORTRAN 5 source program.
ICOBOL	Compiles an Interactive COBOL source program.
ICX	Invokes the Interactive COBOL runtime environment.

Program Development Utilities

Program development utilities include text editors, assemblers, the Relocatable Loader, the Library File Editor, and the Overlay Loader, as listed in Table 7-2.

Text Editors

You create source program files, whether in a high-level language or assembly language, with a text editor. RDOS supports two text editors — EDIT and SPEED (or NSPEED for NOVA computers). When you invoke a text editor, you enter an editing environment in which you use editing commands to enter and modify text. The editing commands for EDIT and SPEED are very similar; of the two editors, SPEED is the more powerful.

Assemblers

Once you have completed your source program, you need to prepare your program for execution. As a first step, you must translate your symbolic instructions into machine-readable code. Assemblers provide this function for assembly language source code (language compilers translate high-level languages).

RDOS supports two assemblers — the Extended Assembler, ASM, and the Macroassembler, MAC. Of the two assemblers, ASM is faster, and MAC is more powerful. MAC allows you to define sets of instructions as macros, and call those instructions as you need them.

The assembler translates your symbolic instruction code and symbolic addresses into machine-readable numeric codes and numeric addresses, producing a relocatable binary file. The relocatable binary file has the filename of your source file plus an .RB filename extension. The assembler flags any errors it encounters during assembly, and produces a report, or *listing*, of assembler statistics.

The Relocatable Loader: RLDR

You use the .RB file produced by an assembler or compiler for the RLDR (Relocatable Loader) utility. RLDR takes the relative addresses set up by the assembler or compiler and replaces them with absolute memory addresses.

RLDR produces an executable program (.SV file), and, if requested, an overlay program (.OL file). RLDR uses the filename of the first .RB file specified on the command line and appends the .SV and .OL extensions. Executable programs are sometimes referred to as save files.

RLDR flags any errors it encounters during loading, and produces a listing of program load statistics.

RLDR can accept a number of files, also known as modules. These can include .RB files and library (.LB) files, which are binary files created by the assembler and managed with the Library File Editor (LFE).

You execute your program by typing in the name of your .SV file as a CLI command.

The Library File Editor: LFE

The Library File Editor, LFE, allows you to build and maintain library files. A library file contains a set of .RB files that you group for later use. You assemble library files with the assembler and group them into library files with LFE. RLDR can extract binary modules from your libraries for loading into an executable program.

The Overlay Loader Utility and the REPLACE Command

The overlay loader, OVLDR, creates an overlay replacement file containing individual overlay modules. You can replace the individual overlays in an overlay (.OL) file with the overlays contained in an overlay replacement (.OR) file. OVLDR creates the .OR file; you use the CLI command REPLACE to replace the contents of the existing .OL file with the contents of the .OR file.

Table 7-2. Program Development Utilities

Utility	Description
ASM	Extended assembler; assembles source files to produce a relocatable binary (.RB) file.
EDIT	Invokes the EDIT text editor.
LFE	Library File Editor; creates, edits, and analyzes library files.
MAC	Macroassembler; assembles source files to produce a relocatable binary (.RB) file.
NSPEED	Invokes the NOVA Superedit text editor.
OVLDR	Overlay Loader; creates an overlay replacement (.OR) file.
REPLACE*	CLI command; replaces the overlays in an overlay (OL) file with new overlays from an overlay replacement (.OR) file created with OVLDR.
RLDR	Relocatable Loader; loads relocatable binary (.RB) files to produce an executable program (.SV) file.
SPEED	Invokes the SPEED (Superedit) text editor.

*CLI command

Debugging Utilities

The debugging utilities summarized in Table 7-3 include a debugger, two file patch utilities, and two disk file editors.

The DEB CLI command invokes one of two RDOS debuggers, if you have requested that RLDR bind a debugger into your program. You can request the default debugger, DEBUG, or the interrupt disable debugger, IDEB. Then you invoke your program with the CLI command DEB; this causes the system to start executing the program at its debugger address, not the starting address. With the debugger, you can make modifications and continue to test your program.

The file patch utilities work in concert to install patches into a .SV file. ENPAT creates the replacement code (or patches), and places it into a patch file. PATCH installs the patches from the ENPAT patch file into the .SV file.

The disk file editors, SEDIT and OEDIT, allow you to modify a .SV file, location by location. The editors let you and work with the values in more than one number base (the default number base is octal).

Table 7-3. Debugging Utilities

Utility	Description
DEB *	CLI command; loads program at debugger address for debugger loaded with RLDR.
ENPAT	Creates a patch file containing program fixes.
OEDIT	Octal editor; examines and modifies disk file locations.
PATCH	Installs patches created with ENPAT into a program or overlay file.
SEDIT	Symbolic editor; analyzes and modifies the contents of disk file locations.
*CLI command	

File Backup Utilities

Table 7-4 summarizes the file backup utilities discussed in Chapter 5. These utilities create compact files in dump format; the compacted files serve as backup copies of files on an active disk. The *load* counterpart restores the files in their original form on disk.

Table 7-4. File Backup Utilities

Utility	Description
DDUMP	Dumps files from the current directory to one or more diskettes, in dump format.
DLOAD	Loads (restores) files previously dumped with DDUMP from one or more diskettes into the current directory.
FDUMP	Dumps files from the current directory to one or more magnetic tapes.
FLOAD	Loads files previously dumped with FDUMP from one or more magnetic tapes to the current directory.

System Utilities

The system utilities include a Batch Monitor, a System Generation utility, Sort/Merge utilities, the Vertical Format utility, and the DO macro utility, as listed in Table 7-5.

The Batch Monitor

The Batch Monitor utility executes a *batch job*. A batch job consists of series of batch commands that can execute without user intervention. Batch commands are batch processing counterparts to CLI commands, and are documented in Chapter 9 of this manual.

The System Generation Utility: SYSGEN

SYSGEN generates an operating system according to user specifications. SYSGEN's interactive dialog prompts you for specifications, and then builds an operating system program file based on those specifications.

The Sort/Merge Utility

The CSSORT Sort/Merge utility allows you to define records and fields in a file to be manipulated and reordered according to your specifications. (Use the RDOSSORT utility on NOVA computers.)

The Vertical Format Utility: VFU

The VFU utility lets you set up a data channel line printer to produce forms according to the specifications in your VFU control file.

The DO Utility

The DO utility offers an alternate method to invoke CLI macro (.MC) files. DO adds more versatility to macro files through its ability to pass arguments to dummy argument variables in the macro file at execution time.

Table 7-5. System Utilities

Utility	Description
BATCH	Batch Monitor; executes a batch job stream made up of batch commands, which are closely related to CLI commands. A batch job stream executes without user intervention.
CSSORT	CSSORT Sort/Merge Utility; reorganizes records in a file or group of files, according to user specifications.
DO	Executes a CLI macro (.MC) file, and passes the arguments you supply on the command line to dummy argument variables in the macro file.
RDOSSORT	RDOSSORT Sort/Merge Utility; reorganizes records in a file or group of files, according to user specifications.
SYSGEN	Generates a new RDOS, DOS, or DG/RDOS system.
VFU	Vertical Format Utility; creates, edits, and loads a format control file for producing forms on a line printer.

Related System and Programming CLI Commands

The CLI commands summarized in Table 7-6 relate to program development and system functions. You can use these commands in conjunction with the utilities listed in this chapter. See the description in the Command Dictionary for more information on a particular command.

Table 7-6. Related System and Programming Commands

Command	Description
BOOT	Executes a different operating system or a stand-alone program.
CHAIN	Overwrites the CLI with another program.
ENDLOG	Closes the log file opened by the LOG command.
FILCOM	Compares the contents of two files, word by word, and displays words that differ.
GTOD	Displays the current time and date.
LOG	Opens a log file to record the current CLI session.
MCABOOT	Executes an operating system or a program on another CPU via an MCA line.
MKABS	Creates an absolute binary file from an executable program (.SV) file.
MKSAVE	Creates an executable program (.SV file) from an absolute binary file.
POP	Returns to the next higher level program.
REV	Displays the revision level of an executable program (.SV file).
SAVE	Assigns a filename to a breakfile created by a debugger, and saves the file on disk.
SDAY	Sets the system calendar date.
STOD	Sets the system clock.
TUOFF	Closes the tuning file started by the TUON command.
TUON	Starts recording system tuning information into a tuning file, compiling statistics on system efficiency.
TPRINT	Prints the system tuning file created with the TUON command.

End of Chapter

Chapter 8
The CLI — Assembly Language Interface: CLI.CM and COM.CM

Assembly language programs can use the CLI through the CLI files CLI.CM and COM.CM. An assembly language program can execute CLI commands and utilities, and pass arguments from the CLI to the program. This allows a program to call on the CLI for its ability to handle operations such as file management.

Using the CLI from an assembly language program requires understanding of the CLI's interpretation mechanisms. The CLI uses two files, CLI.CM and COM.CM, to communicate with user and utility programs.

Note that when a foreground process generates .CM files, the CLI prefixes the filenames with an F, as in FCLI.CM and FCOM.CM; the foreground files are functionally identical to their background counterparts. This chapter refers to the type of file without the F prefix.

CLI.CM Files

CLI.CM is a file that utilities and user programs can use to store certain commands for the CLI to execute. For example, the SYSGEN utility, which generates RDOS operating systems, stores an RLDR command line in CLI.CM in order to use RLDR to build executable programs.

User programs can use system calls to store CLI command lines in CLI.CM, and then swap or chain to the CLI to execute CLI.CM's contents, with an .EXEC system call. A *swap* saves the program's memory image, allowing the CLI to execute temporarily; a *chain* overwrites the executing program with the CLI.

When a program issues the .EXEC system call to the CLI while having a nonzero value in AC2 (accumulator 2), the CLI searches for and executes file CLI.CM. The CLI executes the commands in CLI.CM as if you had entered the commands interactively from a terminal.

To return to your program from a swap, include a CLI POP command as the last command in CLI.CM, or issue a POP command from the CLI. POP returns processing to the next higher level program. To return to a user program from a chain, include a command to execute the program as the last line in CLI.CM, or execute the program from the CLI.

Figure 8-1 shows a listing of an assembled program that stores command lines in CLI.CM, invokes the CLI on level two, and uses POP to return from the CLI.

```
                        .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
DG-09503
```

Figure 8-1. Program Using CLI.CM Example

COM.CM Files

The CLI uses COM.CM files to store command lines and switches for utility programs and user programs.

When the CLI reads a command line, and the first word is not a CLI command, the CLI assumes the first word is the name of a utility program or user program. The CLI builds a file called COM.CM (or FCOM.CM in the foreground), in which it stores the complete command line, including filenames, switches, and arguments, in an edited format.

When the CLI then invokes a utility, it examines COM.CM. When the CLI invokes a user program, you can request that it examine COM.CM. In this way, you can use COM.CM to pass instructions and arguments to your program.

The CLI places command lines in COM.CM in a specific format. For example, you can enter a utility command line in a number of ways, but COM.CM will always hold the command line in a format that the utility expects. The same is true for command lines that invoke user programs.

In the COM.CM format for a user command, each character of the user command and filename arguments occupies a byte. Each filename is terminated with a null byte.

For example, if you enter the command line

TEST ⤸

the CLI builds the command line into COM.CM with the word and byte organization shown in Figure 8-2.

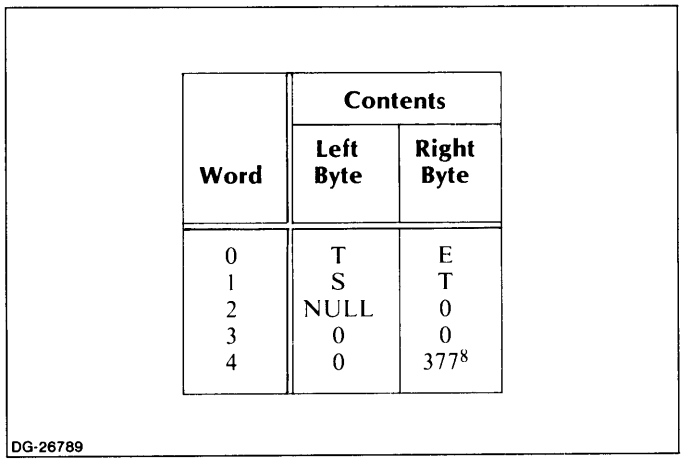


Figure 8-2. COM.CM File Example

After each argument in the command line, the CLI reserves four bytes for switches. Global switches are placed after the first filename argument, and local switches are placed after each argument. The CLI does not interpret switches when building COM.CM, but sets an appropriate bit for each one.

Figure 8-3 illustrates the bits set for letter switches in COM.CM files.

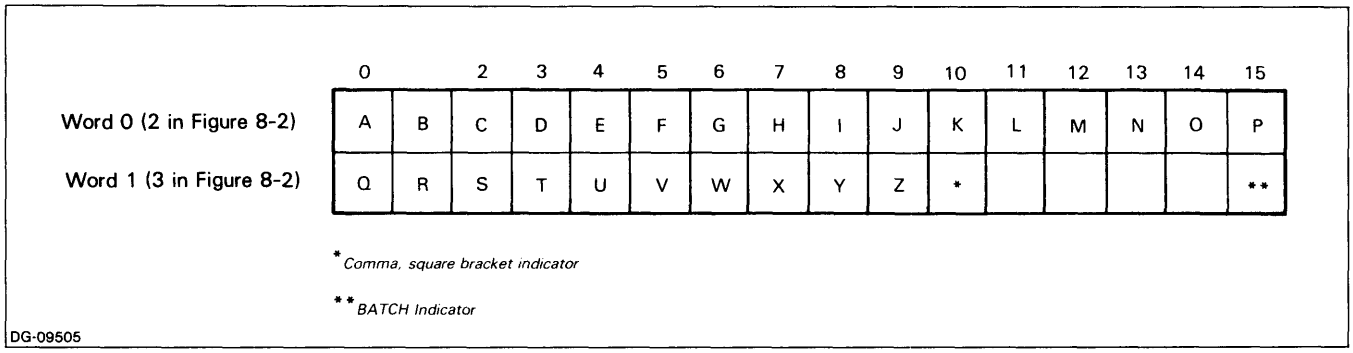


Figure 8-3. Bits Set for Letter Switches in COM.CM

Although the CLI does not interpret switches when it builds COM.CM, your program can interpret the switches and act on what it reads. To read *arguments* in COM.CM, use the system call .RDL, which reads a line from a disk file and terminates on encountering a null byte. To read *switches* in COM.CM, use the .RDS system call, which reads a specified number of bytes.

You can examine the contents of a COM.CM file with the command

```
FPRINT/Z/B COM.CM ↵
```

This command prints the file in byte format, starting with location 0.

COM.CM Examples

Figure 8-4 illustrates how the CLI builds a COM.CM file for the sample command line

TEST/B A ZZZ/X MUMBL ↵

Word	Contents		
	Left Byte	Right Byte	
0	T	E	Command filename TEST, terminated by null byte.
1	S	T	
2	NULL	1B9	Global switch field of TEST. 1 in bit position 9 for /B switch.
3	0	0	
4	0	A	Argument A, terminated by null byte.
5	NULL	0	4 bytes reserved for local switches of argument A. None set.
6	0	0	
7	0	Z	Argument ZZZ, terminated by null byte.
8	Z	Z	
9	NULL	0	
10	0	1B15	Local switches of ZZZ. 1 in bit position 15 for /X switch.
11	0	M	
12	U	M	Argument MUMBL, terminated by null byte.
13	B	L	
14	NULL	0	Local switches of MUMBL. None set.
15	0	0	
16	0	377_8	Terminating byte.

DG-26790

Figure 8-4. Sample COM.CM File

Figure 8-5 is an example of a background program that reads the first argument and global switches from a COM.CM file. (Location data has been removed from the example.)

```
                .
000001          .TXTM 1
                .
                .
                .
020420 READ:    LDA 0, COMCM    ;Pointer to COM.CM
006017          .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).
006017          .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.
006017          .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:
006017 ER:      .SYSTM
006400          .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
```

DG-09507

Figure 8-5. Sample Program That Reads Portions of COM.CM

Using the CLI to Interpret Program Errors

The .ERTN system call, which returns from a swap with error status, enables you to instruct the CLI to interpret errors. When the program encounters an error that prevents it from proceeding, the CLI interprets the error code found in AC2 and displays an error message on the terminal.

The CLI error message names the program that takes the error return instead of the program that causes the error, as the CLI is not active while the program executes. However, the context of the message will be accurate.

Alternatively, your program can use .ERTN with mnemonic EREXQ (code 17(octal)) in AC2. When the program manifests an error and the CLI examines AC2, EREXQ instructs it to examine CLI.CM. The CLI then executes the contents of CLI.CM. In this way, you can specify an action on a program error condition.

Utility Program Formats in COM.CM

Figure 8-6 depicts the COM.CM structures for the following CLI system utilities: ALGOL, ASM, Batch Monitor, FORT or FORTRAN, FLE, MAC, OVLDR, and RLDR. The CLI always builds COM.CM with this structure, regardless of the order you specify in a command line.

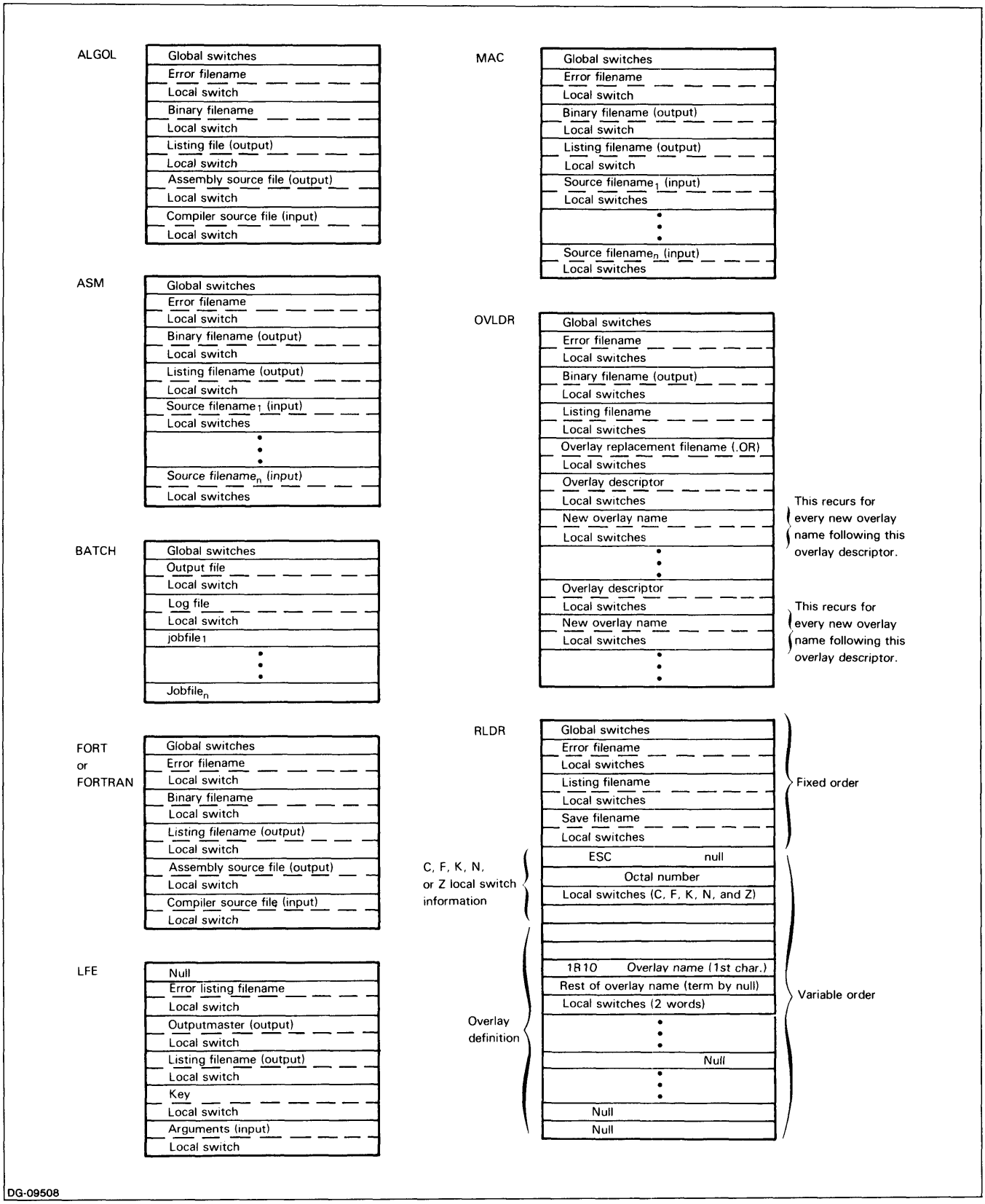


Figure 8-6. COM.CM Structure for Utility

For the compiler utilities, source filenames are listed in their order of appearance.

Figure 8-7 completes the COM.CM structure given for RLDR in Figure 8-6. Figure 8-7 shows the overlay definition field for overlay definition areas A, B, C.

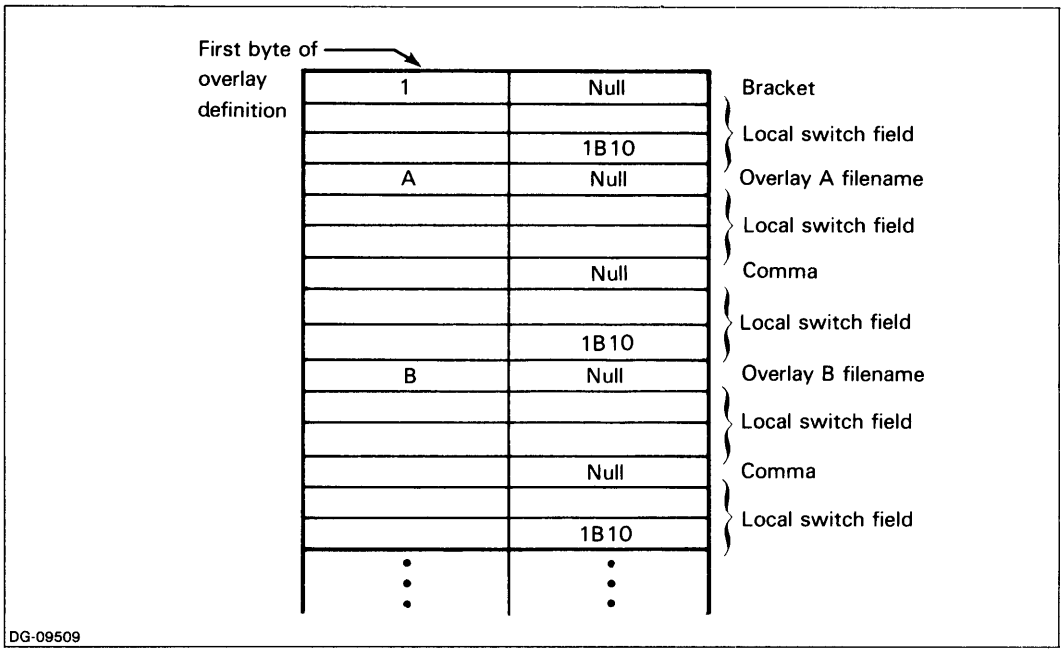


Figure 8-7. RLDR Overlay Definition Field

Under RLDR, local switch information fields (C, F, K, N, or Z) and overlay definition fields can appear in any order following the fixed portion of the COM.CM file.

For the CLG utility, which compiles, loads, and executes FORTRAN IV source files, the following local switch operations are available:

/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.

Figure 8-8 depicts the COM.CM structure for the CLG utility.

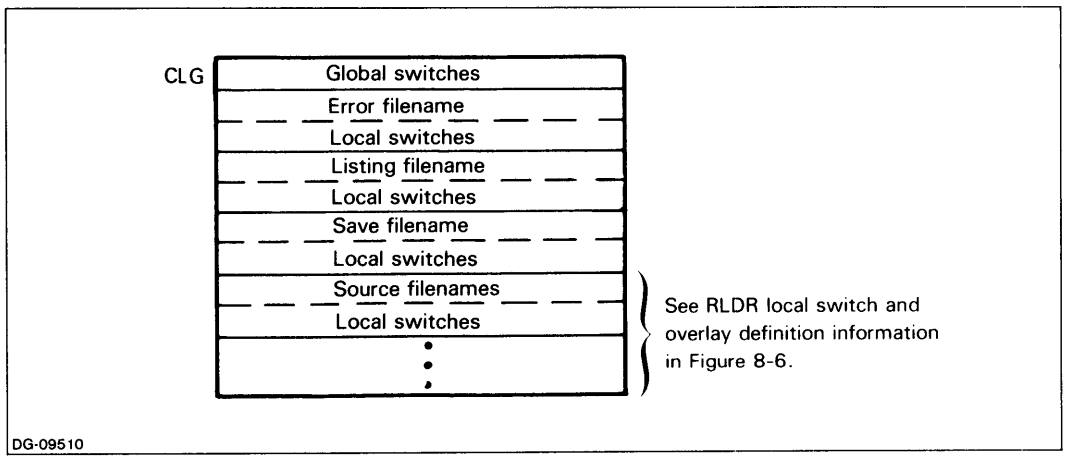


Figure 8-8. COM.CM Structure for the CLG Utility

End of Chapter

Chapter 9
The Batch Monitor

Batch processing, like indirect and macro file processing, allows you to simulate a CLI session with a predetermined set of commands, which you invoke at your convenience. Using the CLI command BATCH, you submit the command sequence in a job stream (specifying one or more jobs) to the Batch Monitor, a utility of CLI. The Batch Monitor has its own set of commands, most of which are very similar to CLI commands.

In addition, the Batch Monitor

- maintains a log of system usage for each job
- captures program output and error listing information
- defers actual device assignments until job execution time
- prefixes a title page to each job
- sends messages to the user while it is executing

The Batch Monitor is released with RDOS.

Invoking the Batch Monitor

To invoke the Batch Monitor, use the CLI command BATCH in the format

BATCH *[jobfile1 ... jobfilen] [outputfile/O] [logfile/G]*

where:

BATCH	invokes the Batch Monitor.
[jobfile1... jobfilen]	are optional arguments, each specifying a device or disk file from which the Batch Monitor draws its input.
[outputfile/O]	specifies a device or file to receive the Batch Monitor's program output and error information. If this optional argument is used, the /O switch is required.
[logfile/G]	names a device or file to receive log information about each file. If this optional argument is used, the /G switch is required.

The effects of each of the arguments to the BATCH command are described in the following sections.

Batch Monitor Input

The Batch Monitor processes input in a job stream in the order specified by the BATCH command. A job stream consists of one or more jobs and a job consists of a sequence of the Batch Monitor commands. Jobs are specified as job files, which are input via a device or disk file.

NOTE: You can defer or change assignment of an input device by including the Batch Monitor commands !CTA, !DKP, and !MTA in the Batch command file.

By default, input to the Batch Monitor is from the card reader ($CDR). Use job file argument(s) to the BATCH command to specify input from a disk or any of the devices listed in Table 9-1.

Table 9-1. System Input Devices

Name	Device
$CDR	Punched card reader
$CDR1	Secondary punched card reader
CTn:m	Data General cassette unit n, file m, first controller
CT1n:m	Data General cassette unit n, file m, second controller
MTn:m	7- or 9-track magnetic tape transport n, file number m (e.g., MT0:2)
MT1n:m	7- or 9-track magnetic tape transport n, file number m, second controller
$PTR	High-speed paper tape reader
$PTR1	Secondary high-speed paper tape reader
$TTR	Console teletype reader
$TTR1	Secondary teletype reader

Disk files are especially convenient when a series of jobs is performed repetitively. When the job input is a disk file, the extension .JB is recommended, since the CLI searches for jobfile.JB first and then for jobfile. Thus the commands

```
BATCH BCD.JB ⤸
```

and

```
BATCH BCD ⤸
```

both execute the job commands in file BCD.JB.

If the job file name has an explicit extension (e.g. .XX), the Batch Monitor searches for the job file with that extension and no other.

For the CLI BATCH command to execute successfully, the disk file's directory must be included in the pathname (directory specifier). Some valid disk names are, for example

DP3 Moving-head disk, unit 3

DK1 Fixed-head disk, controller 1

Thus, to initiate a Batch job stream called MONTHEND.JB, the command could be

```
BATCH DP1:MONTHEND ⤸
```

Batch Job Output

When you submit a job stream with the CLI command BATCH, the Batch Monitor takes over and executes the job commands. Transactions, results, and processing information about the job stream are recorded by the Monitor in two separate files.

- The special file SYSOUT receives listing (program output) and error information.
- The log file for the job stream receives the name of each job, the date and time each job was begun, and the date and time the job stream was completed.

You may use all output devices as well as disk files as either system output devices or as log devices. The valid names for system output devices are listed in Table 9-2.

Table 9-2. System Output Devices

Device	Description
CTn:m	Data General cassette unit n, file m in the first controller
CT1n:m	Data General cassette unit n, file m in the second controller
Dln:filename	Disk or diskette, unit name l, number n (e.g., DZ4:ZFILE)
$LPT	80- or 132-column line printer
$LPT1	Second line printer, either 80- or 132-column
MTn:m	7- to 9-track magnetic tape unit n, file number m, first controller
MT1n:m	7- to 9-track magnetic tape unit n, file number m second controller
$PLT	First digital plotter
$PLT1	Second digital plotter
$PTP	High-speed paper tape punch
$PTP1	Second high-speed paper tape punch
$TTO	System console
$TTO1	Second teletype printer or video display
$TTP	System teletype punch
$TTP1	Second teletype punch

Output File

RDOS reserves the file SYSOUT for program output and error information. By default, the Batch Monitor writes SYSOUT to the line printer ($LPT), but you can change the SYSOUT destination to any output device or to a disk file with an argument to the BATCH command. This argument takes the form outfile/O.

If the output file is a disk file in the system directory, the device mnemonic prefix is optional; otherwise the mnemonic is required.

All system utilities (ASM, FORT, LFE, MAC, OVLDR, and RLDR) use SYSOUT as the default error and listing file. When the listing and error files are selected by default, the listing file precedes the error file on SYSOUT.

Specifying a Title File (TITLE.JB)

You can specify a disk file, called a title file, whose contents will be printed as the first page of output on the system output device. Each title file must have the reserved name TITLE.JB, and no job or job stream may have this name.

In addition to printing the contents of the title file, the Batch Monitor can add to the title files the current job command, date, and time. To obtain this information, use the following reserved character sequence in the first two character positions of any title file line in which they are to appear.

^J Print the current job command and identification on this line.

^D Print the current time and date on this line.

You can create a title file with a text editor before executing a job stream. You can also create different title information files, each replacing the current contents of file TITLE.JB by including the appropriate CLI commands in the job stream.

Log File

The Batch Monitor maintains a log file to record transaction information on each job — its name, the time and date of its initiation, and the time and date of its completion.

The default output device for the log file is the operator terminal: $TTO for a background program or $TTO1 for a foreground program. You can change the log destination to any output device or to a disk file with an argument to the BATCH command. This argument takes the form logfile/G.

You cannot organize disk log files contiguously. If you specify a Batch disk log file and then respecify it with the same name for several runs of the Batch Monitor, each run will append log information to the original file.

Terminating the Batch Monitor

Control characters (CTRL-C and CTRL-A) typed at the system terminal end an executing user job or the entire Batch operation.

Enter CTRL-C to terminate the currently executing user job, This control character releases all current job dependent devices, deletes temporary files used by the job, and sends the message

JOB TERMINATED BY OPERATOR

on system output devices and in the log file, along with the date and time termination occurs. The Batch Monitor then starts the next job in the stream.

Enter CTRL-A to terminate an entire BATCH operation. This control character releases all current job dependent devices, deletes temporary files used by the job, and outputs the following messages in the log file.

****JOB TERMINATED BY OPERATOR****

****BATCH TERMINATED BY OPERATOR****

BATCH TERMINATED: date and time

Batch Monitor Commands

Batch Monitor commands are used to process one or more serial job streams without operator intervention at the completion of each job step.

NOTE: The Batch Monitor does issue messages to the operator indicating that various devices need to be loaded or dismounted, or that operator responses are incorrect. These messages are discussed later in this chapter.

Most Batch Monitor commands are similar to the CLI commands, except that all Batch commands begin with an exclamation point(!). See Chapter 2 for information about command line components. Table 9-3 lists all the Batch Monitor commands, and the Batch Monitor Command Dictionary at the end of this chapter discusses each command individually. Note that in Table 9-3, the commands marked * have no corresonding command in the CLI.

Table 9-3. Batch Monitor Commands

Command	Function
!ALGOL	Compile an ALGOL source file.
!APPEND	Copy one or more files to a new file.
!ASM	Assemble a source file.
!BPUNCH	Punch a binary file.
!CCONT	Create a contiguous file.
!CHATR	Change the attributes of a file.
!CHLAT	Change link access attributes of a file.
!COMMENT *	Output an operator message.
!CRAND	Create a random file.
!CREATE	Create a sequential file.
!CTA *	Assign a physical cassette unit and define its name.
!DELETE	Delete a file.
!DIR	Change the current directory.
!DISK	List the number of blocks used and available on the current partition.
!DKP *	Assign a physical moving-head disk unit and define its name.
!DUMP	Copy files from the current directory to file or device in dump format.
!EOF *	End user job stream.
!EXEC *	Execute save file.
!FILCOM	Compare two files.
!FORT	Compile a FORTRAN IV source file.
!FORTRAN	Compile a FORTRAN 5 source file.
!FPRINT	Print a disk file in byte, decimal, hexadecimal, or octal format.
!GDIR	Print in SYSOUT the name of the current default directory.
!GMEM	Get the current memory allocations in a mapped system.
!GSYS	Get the name of the operating system under which Batch is currently executing.
!GTOD	Print the current time and date in SYSOUT.
!JOB *	Define the beginning of a job.
!LFE	Edit a library file.
!LINK	Create a link entry to a file.
!LIST	List file information.

(continues)

Table 9-3. Batch Monitor Commands

Command	Function
!LOAD	Reload dumped files.
!MAC	Assemble source files using the Macroassembler.
!MDIR	Get the name of the system master directory.
!MKABS	Make an absolute binary file from a save file.
!MKSAVE	Make a save file from an absolute binary file.
!MOVE	Move files from one directory to another.
!MTA *	Assign a physical magnetic tape unit and define its name.
!OVLDR	Create an overlay replacement file.
!PAUSE	Output an operator message and pause.
!PRINT	Print on SYSOUT a file or files.
!PUNCH	Output ASCII files on paper tape punch.
!RDOSSORT	Invoke RDOS Sort/Merge program.
!RELEASE	Release directory or device from system initialization.
!RENAME	Change the name of a file.
!REPLACE	Replace overlays in an overlay file.
!REV	Display the revision level of a save file.
!RLDR	Load relocatable binary file to produce a save file.
!SAVE	Rename TMP.SV.
!TPRINT	Print the tuning file.
!TUOFF	Stop recording in the tuning file.
!TUON	Start recording in the tuning file.
!UNLINK	Delete a link entry.
!XFER	Transfer contents of a file to another file.

*Command has no CLI counterpart.

(concluded)

This section discusses how to use filename arguments, and two Batch Monitor commands, !JOB and !EOF. An example of how to use Batch commands is presented later in the chapter.

Filename Arguments and Searches

Many Batch Monitor commands accept filename arguments. However, the file to be processed by a Batch command often simply follows the command on the current input job stream device. For example, if the Batch command to assemble a source file, !ASM, has no argument, the Batch Monitor assumes that the source file itself follows the command in the job stream.

When you provide an argument, include the argument's extension. A directory may contain a number of entries having the same basic filename but different extensions, for example, ABEL.SR and ABEL.SV.

When you type

```
!RLDR ABEL ⤸
```

the Batch Monitor searches first for ABEL.RB. If the monitor does not find ABEL.RB, it creates an output file ABEL.SV for the relocatable loader. The .SV extension signifies a save file. The commands !SAVE and !MKSAVE also output save files. In both cases, the Monitor adds the extension .SV to the name of the output file. When you attempt to specify an extension, it is ignored. For example,

```
!SAVE ABEL.XX ⤸
```

stores as the save file ABEL.SV, not ABEL.XX, on disk. For most commands, a filename argument must specify the appropriate extension; if the Monitor cannot find the extended filename, it then searches for the simple filename. For example:

```
!ASM ABEL ⤸
```

directs the Monitor to search first for the file named ABEL.SR. If the search succeeds, ABEL.SR becomes the source file for the assembly. Otherwise, the Monitor searches for ABEL. When ABEL is found, the Monitor uses it to assemble a relocatable binary output file, ABEL.RB. Batch automatically adds the extension .RB to the source file's name.

The !JOB Command

The first command in each job is always the !JOB command; it has the format:

```
!JOB identification
```

Where

!JOB	is a Batch command identifying the start of a new job;
identification	labels the job and enables the Batch Monitor to produce a log of system use information for all jobs in each job stream. May contain up to 132 characters (alphanumeric characters and the dollar sign symbol — $).

The !JOB command delimits each job within a job stream. All commands and files following !JOB are used to process the same job, until the Monitor encounters another !JOB command.

The !EOF Command

The !EOF (end-of-file) command terminates a job stream and returns control to the CLI. If the input device is a card reader, the end of file character can be produced by multipunching +, −, and 0 through 9.

Batch Monitor Operations

This section is divided into two parts. The first illustrates how the CLI BATCH commands are used to process a job stream. The second lists and explains the messages the Batch Monitor sends to the operator as it processes input.

Processing a Job Stream

In the example that follows, two jobs are to be input from the second card reader. The second line printer is selected as the system output device, and log information is to be stored in a disk file called LOG. The following CLI command line defines the job stream:

```
BATCH $CDR1 $LPT1/0 LOG/G ⤸
```

If the jobs are an assembly and a FORTRAN compilation, the card deck input as a job stream to the Batch Monitor would be composed of the cards shown in Figure 9-1.

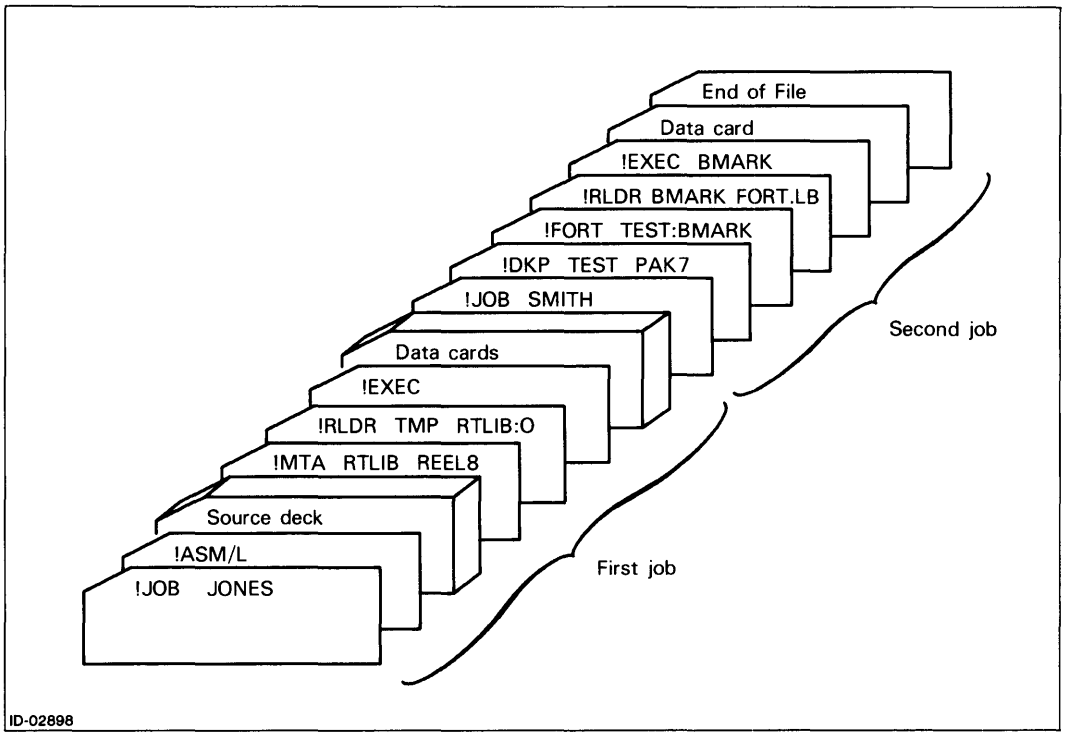


Figure 9-1. Job Stream Input to Batch Monitor

As a result of the commands in the figure,

1. The Batch Monitor begins the first job, JONES.
2. JONES assembles a program, outputs the listing on SYSOUT, assigns the logical name RTLIB to a magnetic transport, and identifies a needed reel as REEL8.
3. Batch loads and executes the assembled program.
5. The Batch Monitor begins the second job, SMITH.
6. SMITH assigns the logical name TEST to the disk named PAK7.
7. Batch directs the system to compile, load, and execute the FORTRAN program BMARK, on TEST.
8. The end-of-file character indicates that the job is complete; control returns to the CLI.

Device Operation Messages

As the Batch Monitor processes jobs, it issues a series of operator messages indicating that various devices need to be loaded or dismounted with appropriate reels, packs, etc., or that operator responses have been improper. The following devices issue such messages:

- Punch card readers
- Cassette units
- Moving head disks
- 7- and 9-track magnetic tape units
- High-speed paper tape readers
- Teletype readers

The user job also sends two types of messages to the console: pause and comment messages.

Punched Card Reader Message

The system issues only one message associated with the card reader:

LOAD $CDR, STRIKE ANY KEY

or

LOAD $CDR1, STRIKE ANY KEY

When you receive this message, load the card deck into the card reader (face down, 9-edge to the back), and place the deck weight on the top of the card stack. To prepare the card reader turn it on by pressing the keys marked POWER and MOTOR, and then depress the key marked START. To indicate to the Batch Monitor that the reader is ready, strike any key on the console keyboard.

Cassette Unit Messages

Whenever BATCH requires a cassette tape, the message:

MOUNT CASSETTExxx

appears. In this message *xxx* is the number of the cassette label. When you receive this message, select a free unit, mount the cassette, remove the file protect tab (unless instructed to do otherwise by a user), close the cassette tape door, and place the unit on line by turning the power switch to REMOTE.

To inform the system which cassette unit the cassette is mounted in, type

CTx ↵

where x is the number of the cassette unit.

When the job releases this cassette unit, you receive the message:

DISMOUNT CTx

where *CTx* is the unit number entered in response to the previous mount message.

The system outputs operator warning messages for certain error conditions. The first of these messages

xxx?

indicates that an incorrect response *xxx* has been input to and received by the system. Often this message is output because the response was misspelled or incomplete. Type the correct response. If the device is not online (the power switch is not in the REMOTE position), the message

UNIT OK? UNIT:xxx

is output; *xxx* is the unit name and number. Place the power switch in the REMOTE position and enter the response

CTx or CT1x ⤶

where x is the unit number to be used.

Moving Head Disk Messages

Whenever the Batch monitor requires a disk pack or cartridge, it sends the message:

MOUNT DISKxxx

where *xxx* is the identification number of the disk pack or cartridge. When you receive this message, select and specify a free unit. When the cartridge or pack drive is ready, type via the system console the unit name and number, followed by a carriage return. For example

DPn ⤶

When the job releases this disk unit, a message appears on the system console, as follows:

DISMOUNT DPn

As with cassettes, the system can issue three warning messages to the operator. The first, *xxx*, indicates that an unrecognizable operator response has been received by the system. Repeat the response correctly. If the moving-head disk unit selected is not ready for system operation, the system sends the message:

UNIT OK? UNIT:xxx

Place the rocker switch of the appropriate unit in the LOAD position (if it is not already there), and when the unit READY lamp is lit, type, for example

DPn ⤶

where n is the number of the disk unit.

Magnetic Tape Unit Messages

Whenever Batch requires a magnetic tape, the message

MOUNT MAG TAPExxx

appears, where *xxx* is the identification number on the tape label. When you receive this message, select a free magnetic tape unit and mount the tape (with a write-enable ring, unless instructed otherwise by the user). Put the unit on line by pressing the REMOTE or ONLINE or other appropriate switch.

After you mounted the tape, give the system the number of the transport in which the tape is loaded, by typing

MTn ⤶

where n is the transport number.

When the job releases this transport, the system outputs a message such as

DISMOUNT MTn

where MTn is the unit number entered in response to the previous mount message.

Two warning messages may be issued. One of these messages,

xxx?

is identical in meaning to the message described for cassettes. The other message,

UNIT OK? UNIT:xxx

indicates that a magnetic tape unit selected was not on line. Put the unit on line by pressing the REMOTE switch. Enter

MTx or MT1x ↵

where x is the unit number.

Paper Tape Messages

Only one message is issued for either the teleptype reader or high speed paper tape reader:

LOAD $TTR1, STRIKE ANY KEY.

LOAD $TTR, STRIKE ANY KEY.

LOAD $PTR, STRIKE ANY KEY.

LOAD $PTR1, STRIKE ANY KEY.

When you receive one of these messages, load the paper tape in the appropriate reader. If this is a high-speed reader, be certain that the power switch (a three-position switch) is in the uppermost position, and that the load lever is down. Also, for the high-speed reader, ensure that the fanfold stack is oriented so that the front end of the tape is nearer the read head and the feed holes are away from you. The teletype reader is keyed to accept the feed holes in only one way when the preprinted "TOP" marking is showing on top.

Pause and Comment Messages

There are two kinds of messages that the user job sends to the operator console:

!COMMENT xxx

and

!PAUSE xxx

Where *xxx* is the message proper. Comment messages do not suspend the system's activity, whereas pause messages suspend program activity until the response:

GO ↵

is typed on the operator console. If you respond any other way, BATCH outputs the query:

xx?

where *xx* are the first two characters of the incorrect response. Batch then awaits the proper response.

Batch Job Example

The following pages list all information sent to SYSOUT during the execution of job EXAMPLE. As each Batch command is input to the monitor, it is output on SYSOUT. Thus the first two lines output on SYSOUT are:

```
!JOB EXAMPLE
!FORT/B/L
```

As the Batch command to perform a FORTRAN IV compilation with a brief listing is executed, the following listing is output to SYSOUT:

```
; COMPILER DOUBLE PRECISION
; WRITE(10) "TYPE PRINCIPAL, RATE AS A DECIMAL, INTEGER NUMBER OF
; YEARS F"
; WRITE (10) "AND 0 FOR BRIEF SUMMARY, OR 1 FOR DETAILED SCHEDULE."
; WRITE (10) "SEPARATE EACH PARAMETER BY A COMMA
  AND TERMINATE"
; WRITE (10) "INPUT WITH CARRIAGE RETURN."
; READ (11) PRIN,R,1YEAR,1FULL
; WRITE (12,6)
; 6 FORMAT (1H1,' ***** HOME MORTGAGE INTEREST CALCULAT
; WRITE (12,5) PRIN,R,1YEAR
; IF( (1FULL) 11,11,21
; 5 FORMAT (1HO,'TOTAL PRINCIPAL = $',F9.2,', ANNUAL INTEREST RATE=',
; XF7.4,/,'T0TAL LIFE OF THE LOAD IS',14,'YEARS.')
; 21 WRITE (12,300)
; 300 FORMAT (1HO,/,'NUM','INTEREST','PRIN.PAY',
; X'PRIN.BAL','INTEREST PAID TO DATE',/,/)
; 11 TINT = PRIN
' N = 12*YEAR
; SUM = 0
; TOINT = 0
; X = 1.
; DO 20 =1,N
; SUM = SUM + X
; 20X = X*(1.+R/12)
; PO =PRIN/SUM
; POO = PO
; DO 30 =1,N
; RINT = PRIN*(R/12.)
; TOINT = RINT + TOINT
; PRIN = PRIN-PO
; IF (FULL) 30,30,32
; 32 WRITE (12,200) ,RINT,PO,PRIN,TOINT
; 30 PO = PO*(1.+R/12.)
; 200 FORMAT (1HO,3,'','$',F9.2,'$',F9.2,
; X'$',F9.2)
; RINT = TINT*(R/12.)
; PAY = POO+RINT
; WRITE (12,100)PAY
; 100 FORMAT (1HO,'MONTHLY PAYMENT = $',F10.2/,/)
; WRITE (12,110) TOINT
; 110 FORMAT (1HO,'TOTAL INTEREST PAID OVER LIFE OF LOAN
= $',F9.2/,
; X/)
; END
```

During the assembly and relocatable load phases of this job, the following lines are output on SYSOUT

PROGRAM IS RELOCATABLE
.TITL.MAIN
!RLDR TMP FORT.LB SYSOUT/L

As relocatable loading is performed, the following load message and load map are output on SYSOUT:

TMP.SV LOADED BY RLDR REV 03 AT 09:43:37 12/11/73
.MAIN

FREAD			
THREA	*.FRG1*	*FFLD2*	*RSTR*
	000114	*006115*	*010675*
RDFLD	*XD.E*	*FFST2*	*QRSTR*
	000115	*006115*	*010711*
READL	*DF.OT*	*FAD2*	*.OFLO*
	000123	*006117*	*010713*
OPEN	*.MVBC*	*FSB2*	*RTER*
	000134	*006120*	*010755*
FOPEN	*.MVBT*	*FML2*	*RTESP*
	000135	*006121*	*010757*
ARYSZ	*.LDO*	*FDV2*	*RTEO*
	000136	*006122*	*010771*
FSBR	*.LD1*	*FXFL2*	*WRCH*
	000137	*006123*	*011171*
SMPY	*LD2*	*FLFX2*	*.BDAS*
	000140	*006124*	*011206*
ARGUM	*.STO*	*FSGN2*	*.BASC*
	000141	*006125*	*011270*
FRGLO	*.ST1*	*FNEG2*	*COUT*
	000142	*006126*	*011325*
FARGO	*.ST2*	*FCLE2*	*CIN*
	000143	*006127*	*011337*
DFL	*.STOP*	*FCLT2*	*LDB*
	000144	*006130*	*011346*
MVBT	*.PAUS*	*FCGE1*	*STB*
	000145	*006131*	*011360*
LDO	*.FCAL*	*FCGT2*	*MOVEF*
	000146	*006132*	*011376*
STOP	*.FSAV*	*FCEQ2*	*CPYAR*
	000150	*006133*	*011423*
FLINK	*.FRET*	*FCALL*	*CPYLS*
	000151	*006146*	*011454*
RTER	*.RTER*	*FRCAL*	*MAD*
	000153	*006147*	*011463*
WRCH	*.RTEO*	*REDS*	*MADD*
	000154	*006200*	*011464*

```
BDASC   .RTES   MPY     .FLSP
        000155  006201  011510
BASC    .WRCH   MPYO    TMIN
        000156  006202  011545
COUT    .COUT   DVD     FHMA
        000157  006203  177777
LDSTB   .CIN    WRITS   FRTSK
        000160  006211  177777
MOVEF   .LDBT   MTDPD   .FLSZ
        000162  006275  177777
MAD     .MOVE   OVOPN   .DSI
        000163  006300  177777
FPZER   .CPYA   OPEN
0002    000164  006303
FPTRS   .CPYL   FOPEN
        000165  006524
DUMMY   .MAD    ARYSZ
        00166   006673
ARDUM   .MADO   FSUBA
        000167  006711
MULT    .IOCA   FSBR
        000170  007054
TMIN    SUCOM   SMPY
        000171  007106
XN.DSI  .NDSP   FARGU
001627  000172  007132
        TVR     FRGLD
        000172  007166
        AFSE    FRGI
        000173  007176
  NMAX  SP0 00174  FRGO 007203
011641
  ZMAX  .OVFL     DFL 007222
000204  000175
  CSZE  .SVO      DFS 007300
000000  000176
  EST   QSP       DFB 007455
000000  000177
  SST   NSP       DFA OO7456
000000  000200
        FLSP 000200  DFM 007730
```

.FREA	.MAIN	DFXL
000050	000440	010035
.FWRI	.I	DFLX
000051	001453	010074
.BRD	SDSW	DFSG
000052	001626	010146
.BRW	IOPTR	DFNG
000053	001632	010174
.ALLO	FERTO	DFLE
000074	001660	010204
.THRE	FERT1	DFLT
00075	001663	010206
.RDFL	FERTN	DFGE
000076	001667	010210
.RDFC	BRD	DFGT
000077	001706	010212
.WRIT	BWR	DFEQ
000100	001712	010214
.READ	FREAD	DFD
000101	001717	010260
.WRTS	FWRIT	MVBT
000102	001723	010457
.REDS	FSAV	MVBC
000103	002150	010463
.FOPE	FRET	LDO
000104	002151	010514
.ARYS	FQRET	LD1
000105	002152	010516
.FSUB	ALLOC	LD2
000106	005567	010520
.FSBR	THREA	STOP
000107	005616	010535
.SMPY	RDFLD	PAUSE
000110	005633	010542
.FARG	RDFCH	SAVO
000111	005637	010605
.FRGL	READL	SAV2
000112	005750	010655
.FRGO	WRITL	SAV3
000113	005760	01066

The FORTRAN save file is executed by the command !EXEC. This command is output on SYSOUT. The data set input to the save file is not displayed on SYSOUT; instead, the following page, triggered by the data set, is output on SYSOUT.

*****HOME MORTGAGE*
*INTEREST CALCULATOR******
TOTAL PRINCIPAL = $1000.00, ANNUAL INTEREST RATE =0.0800
TOTAL LIFE OF THE LOAN IS 1 YEARS.

NUM	*INTEREST*	*PRIN. PAY*	*PRIN. BAL*	*INTEREST PAID TO DATE*
1	*$6.67*	*$80.32*	*$919.68*	*$ 6.67*
2	*$6.13*	*$80.86*	*$838.82*	*$12.80*
3	*$5.59*	*$81.40*	*$757.42*	*$18.29*
4	*$5.05*	*$81.49*	*$675.49*	*$23.44*
5	*$4.50*	*$82.49*	*$593.00*	*$27.94*
6	*$3.95*	*$83.04*	*$509.97*	*$31.90*
7	*$3.40*	*$83.59*	*$426.38*	*$35.30*
8	*$2.84*	*$84.15*	*$342.23*	*$38.14*
9	*$2.28*	*$84.72*	*$257.52*	*$40.42*
10	*$1.72*	*$85.27*	*$172.25*	*$42.14*
11	*$1.15*	*$85.84*	*$86.41*	*$43.29*
12	*$0.58*	*$86.41*	*$-0.00*	*$43.86*

MONTHLY PAYMENT = $86.99

TOTAL INTEREST PAID OVER LIFE OF LOAD = $43.86

!EOF

Finally, the following Batch log information is output on the console and Batch terminates:

12/11/73 9:48:03
!JOB EXAMPLE

Batch Monitor Command Dictionary

This chapter concludes with a dictionary entry for each of the Batch commands.

!ALGOL

Batch Monitor

Compile an ALGOL source file

Syntax

!ALGOL inputfilename

Description

Compile an ALGOL source file. Output may be a relocatable binary file, an intermediate source file, a listing file, or combinations of all three. The command name, ALGOL, must be used in compiling ALGOL source programs; the name, ALGOL, cannot be changed.

By default, executing the ALGOL command produces an intermediate source file, inputfilename.SR (compiler output), and a relocatable binary file inputfilename.RB (assembler output). However, once assembly has been successfully completed, the intermediate source file is deleted. The default command produces no listing.

Global Switches

/A Suppress assembly.

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

/E Suppress error messages from compiler at SYSOUT. (Assembler error messages are not suppressed.)

/L Produce listing to SYSOUT.

/N Do not produce a relocatable binary file.

/S Save the intermediate source input file.

/U Append user symbols to binary output file.

Local Switches

/B Relocatable binary output directed to a given filename.

/E Error messages are directed to a given filename.

/L Listing output directed to given filename (overrides global /L).

/S Intermediate source output directed to given filename.

Extensions

On input search for inputfilename.AL. If not found, search for inputfilename. By default, search for TMP. On output, produce *TMP.RB* by default, and other files with .LS or .SR extensions as determined by switches.

Examples

1. !ALGOL MAIN ⤸

   Produce relocatable binary file, MAIN.RB. No listing is produced.

2. !ALGOL/E/B SUBR LIST/L ⤸

   Produce relocatable binary file, SUBR.RB with a brief (ALGOL source) listing to disk file LIST. Suppress compiler error messages.

3. !ALGOL/A RAY $PTP/S ⤸

   Do not invoke an assembly phase. Punch intermediate source output on high-speed punch.

!APPEND

Batch Monitor

Copy one or more files to a new file

Syntax

!APPEND newfilename oldfilename$_1$ *[oldfilename$_2$...]*

Description

Create a new file, consisting of a concatenation of one or more old files in the order in which their names are listed as arguments. The old files are not changed by the command. No switches.

Examples

1. !APPEND COM.SR COM1 COM2 COM3 COM4 ⤸

   Create the file COM.SR containing the contents of files COM1, COM2, COM3, and COM4 in that order.

2. !APPEND PACK1:ALL.LB A.LB B.LB PACK0:C.LB ⤸

   Create a file ALL.LB on logical disk pack PACK1, containing the contents of files A.LB and B.LB from the default directory and C.LB from logical disk pack PACK0. PACK0 and PACK1 must have been initialized previously by CLI commands DIR or INIT, or by Batch command !DKP.

!ASM

Batch Monitor

Assemble source files

Syntax

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

Description

Assemble one or more source files using the extended relocatable assembler. Output may be a relocatable binary file, a listing file, or both. The command name, ASM, must be used in assembling programs; the name ASM cannot be changed.

Global Switches

By default, input to an assembly is TMP, output of an assembly is a relocatable binary file named TMP.RB (no listing file).

/E Suppress error printouts on SYSOUT.

/L Produce listing file on SYSOUT.

/N Do not produce a relocatable binary or listing file.

/S Skip pass 2. A BREAK is signaled after pass 1, permitting the user to save a version of the assembler that contains the user's semi-permanent symbols.

/T Do not produce symbol table list as part of the listing. (Used when a listing is requested, which produces a symbol table by default.)

/U Append user symbols to the relocatable binary output.

/X Produce cross referencing of symbol table. Symbol table output contains page number-line number pairs for the symbol definition as well as every reference to the symbol within the assembly.

Local Switches

/B Direct relocatable binary output to the given file name. (Overrides global /N.)

/E Direct error output to given file name.

/L Direct listing output to the given file name. (Overrides global /L.)

/N Do not list this file. (Used, when a listing is requested, to list a selected number of fields to be assembled.)

/S Skip file name on pass 2 of assembly. (This switch should be used only if the file does not assemble any storage words.)

Extensions

On input, search for filename.SR. If not found, and the filename did not have an extension, search for filename. By default, search for TMP.

On output, produce *filename.RB* for relocatable binary and *filename.LS* for listing (global /L switch), where *filename* will be the name portion of the first source file specified without a /S, /L, or /B local switch.

!ASM (continued)

Examples

1. !ASM Z ↵

   Assemble source file Z, producing a relocatable binary file called Z.RB.

2. !ASM/N/L A ↵

   Assemble file A, producing as output a listing on SYSOUT.

!BPUNCH

Batch Monitor

Punch file or files in binary format

Syntax

!BPUNCH filename$_1$ *[...filename$_n$]*

Description

Punch a given file or files in binary format on the paper tape punch. The input files can come from any device. No switches.

The command is the equivalent of a series of XFER commands, as follows:

!XFER filename$_1$ $PTP, !XFER filename$_n$ $PTP

Examples

1. !BPUNCH FEE.SR FIE.SR FOE.RB FUM.RB ↵

   Punch source files FEE and FIE, and relocatable binary files FOE and FUM on the punch.

2. !BPUNCH $PTR ↵

   Punch a paper tape duplicate of the tape in the paper tape reader.

!CCONT

Batch Monitor

Create a contiguous file

Syntax

!CCONT filename$_1$ blockcount$_1$ *[filename$_n$ blockcount$_n$]...*

Description

Create one or more contiguous files. The file has the length given in decimal disk blocks by blockcount, and no attributes. Disk storage is allocated in increments of 256_{10} words each. No switches.

Examples

1. !CCONT ALPHA 20 ⤸

   Create the contiguous file, ALPHA, in the default directory, and allot 20 disk blocks for the file.

2. !CCONT TEST 100 PACK0:TEST1 51 ⤸

   Create two files, TEST in the default directory with 100 blocks and TEST1 in the directory PACK0 with 51 blocks.

!CHATR

Batch Monitor

Change a file's attributes

Syntax

!CHATR filename$_1$ attributes$_1$ *[...filename$_n$ attributes$_n$]*

Description

Change, add, or delete resolution file attributes of a given file. No switches.

Attributes

N Not a resolution entry (cannot be linked to)

P Permanent file (cannot be deleted or renamed)

R Read-protected file (cannot be accessed for reading)

S Save file (having been set, cannot be removed)

W Write-protected file (file cannot be altered)

0 No attributes

* Current file attributes (Dash convention is not used.)

? First user-defined attribute

& Second user-defined attribute

When several attributes are specified for a given file name, they must be a contiguous string. Attributes can be listed in any order in the argument.

Characteristics

The following are file characteristics that the system sets when creating a file. File characteristics can't be set or changed with !CHATR.

A Attribute-protected file

C Contiguously organized file

D Randomly organized file

I Accessible by direct I/O only

T Partition file

Y Directory file

Examples

1. !CHATR ABBY WP ⤸

   Make file ABBY write-protected and permanent.

2. !CHATR ABBY 0 BUDDY R ⤸

   Delete all attributes of ABBY and make BUDDY read-protected.

3. !CHATR AGATE.SV W ⤸

   Make AGATE.SV a write-protected save file. If AGATE.SV had had other previous attributes, these would be deleted.

4. !CHATR AVESTAN.SV *W ⤸

   Make AVESTAN.SV a write-protected file, retaining any existing attributes.

!CHLAT

Batch Monitor

Change a file's link access attributes

Syntax

!CHLAT filename$_1$ attributes$_1$ *[...filename$_n$ attributes$_n$]*

Description

Change, add, or delete file link access attributes. All current link access attributes of the file are replaced by those given in the attributes argument. No switches.

Attributes

N Not a resolution entry, cannot be linked to.

P Permanent file, cannot be deleted or renamed.

R Read-protected file,cannot be accessed for reading.

S Save file; having been set, attribute cannot be removed.

W Write-protected file, cannot be altered.

0 No attributes.

* Current file attributes.

? First user-defined attribute.

& Second user-defined attribute.

When several attributes are specified for a given file name, they must be given as a single argument. Attributes may be listed in any order in the argument.

Characteristics

The following are file characteristics that !CHLAT cannot set or change.

A Attribute-protected file.

C Contiguously organized file.

D Randomly organized file.

I Accessible by direct I/O only.

T Partition file.

Y Directory file.

For examples, see those given for !CHATR.

!COMMENT

Batch Monitor

Output a message to the operator

Syntax

!COMMENT messagestring

Description

Output an operator message during the execution of a job. The message can be any combination of up to 72 printable characters. No switches.

Example

The following series of commands illustrates a portion of a job that the programmer did not want to run over 5 minutes, perhaps because this would indicate that the program was malfunctioning:

```
!JOB HOPKINS ↓
!COMMENT ABORT THIS JOB IF IT RUNS OVER 5 MINUTES. ↓
!ASM MYFILE ↓
!RLDR A B C ↓
```

Prior to assembling the program, the following message is output on the operator console:

!COMMENT ABORT THIS JOB IF IT RUNS OVER 5 MINUTES.

!CRAND

Batch Monitor

Create a random file

Syntax

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

Description

Create a randomly organized file. The entry specifies a file of 0 length and no attributes. No switches.

Example

```
!CRAND RANDFILE )
```

Create a randomly organized file named RANDFILE in the current directory. The file is of 0 length and has no attributes. The size of the file grows as required during use.

!CREATE

Batch Monitor

Create a sequential file

Syntax

!CREATE filename$_1$ *[filename$_2$...]*

Description

Create a sequentially organized file in the default or specified directory. The file has 0 length and no attributes. No switches.

Examples

1. !CREATE ALPHA ↵

   Creates a file, ALPHA, in the default directory.

2. !CREAT TEST TEST1 PACK1:TEST2 ↵

   Create three files, TEST and TEST1 in the default directory and TEST2 in the directory of PACK1. PACK1 must have been intialized previously.

!CTA

Batch Monitor

Assign a cassette unit and define its name

Syntax

!CTA logical_device_name *[cassette_label]*

Description

Assign an available cassette unit to a logical cassette that is referred to in a user program. An optional cassette label may be specified to identify the specific cassette to the Batch operator. No switches.

When the Batch Monitor encounters this command, it directs the operator to mount the tape. If you've added a *cassette_label*, the Batch Monitor further identifies the cassette with the label to the operator.

The operator then mounts the cassette on an available unit and responds via the system terminal with the name of the cassette unit selected, e.g., CT3. The system then rewinds the cassette to BOT (beginning-of-tape) and resets the tape file pointer to 0.

Example

Throughout an assembly language program, the programmer has consistently referred to a cassette by the mnemonic MYFILE. For instance, the third file on this cassette is MYFILE:2. The cassette has the label number 12345.

Upon encountering the BATCH command

```
!CTA MYFILE 12345 ↵
```

the system sends the operator the following message:

MOUNT CASSETTE 12345

The operator then selects a free cassette unit, e.g., CT1, places the cassette in the unit, and responds at the terminal

```
CT1 ↵
```

All program references to MYFILE now become references to CT1; e.g., MYFILE:3 becomes CT1:3.

!DELETE

Batch Monitor

Delete a file or directory

Syntax

!DELETE filename$_1$ *[filename$_2$...]*

Description

Delete files having the names given in the argument list. A **filename** can be preceded by a directory specifier if the directory has been initialized.

Any attempt to delete a link will instead delete the link's resolution file. To delete a link, use the !UNLINK command.

Template characters are permitted only when the **filename** is in the current directory.

Global Switches

/V Verify deletion with a list of names of deleted files on SYSOUT.

Local Switches

/A Delete only the files created on this date or later. Preceding argument is of the form mm-dd-yy, where mm and dd can be one or two digits.

/B Delete only the files created before this day. Preceding argument is of the form mm-dd-yy, where mm and dd can be one or two digits.

/N Do not delete files that match this name.

Examples

1. !DELETE ALPHA BETA GAMMA ↵

   Delete all files named ALPHA, BETA, and GAMMA.

2. !DELETE LIMIT.- ↵

   Delete all files with the name LIMIT and any extension (including null).

3. !DELETE A**B ↵

   Delete files AZYB and AXWB, but not file AB or AXB.

!DIR

Batch Monitor

Change the current directory

Syntax

DIR directoryname

Description

Change the current default directory within any single job of a Batch stream. At bootstrap time, a master device is established as the default directory. The DIR command sets another disk device, directory, or subdirectory as the default directory within a job. The command also initializes (INIT command) the directory.

At the end of each job, the default directory prior to the job is restored as the current directory.

directoryname is the name of a subdirectory, secondary partition, or primary partition. The argument can include the directory's filename extension (.DR), but the extension is not required. The directoryname can be in a pathname.

No switches.

Examples

1. !DIR MYPAK ↵

   Change all default filename references to the disk directory MYPAK.

2. !DIR DP1:DEF ↵

   Direct all filename references to subdirectory DEF, which resides in primary partition DP1.

!DISK *Batch Monitor*

Obtain count of blocks used and available in current partition.

Syntax

!DISK

Description

Obtain a count of the number of blocks used and the number of blocks left for the user in the current partition. If the current directory is a subdirectory, the size of the parent partition is indicated. No switches.

The net number of blocks available to a user (n) is always less than the number of physical blocks (p) on the disk, due to the space requirements for HIPBOOT and the structure of map directories under RDOS. The value n reported by the DISK command is the largest integer multiple of 16(10) that is less than or equal to p−6.

Example

```
!DISK ↓
LEFT: 2734, USED: 2130
```

This response indicates that 2734 out of a total of 4864 blocks are still available for use.

!DKP

Batch Monitor

Assign disk unit and define its name

Syntax

!DKP logical_device_name *[cartridge_label]*

Description

Assign an available disk drive to a logical disk unit referred to in a user program. An optional cartridge label can be specified to identify the specific disk to the Batch operator. Only the primary partition of the logical unit becomes available. No switches.

When the Batch Monitor encounters this command, it directs the operator to load the disk, identified by its cartridge label, if given.

The operator then loads the disk on an available disk drive and enters the name of the drive selected on the terminal; e.g., DP1. This response must be given only after the drive READY light has gone on.

Example

Throughout an assembly language program, the programmer has consistently referred to a disk pack as MYFILE; e.g., file ABC on his pack as MYFILE:ABC. This disk pack has the identification number 12345.

Upon encountering the Batch command

```
!DKP MYFILE 12345 ↵
```

the system sends the operator the following message:

MOUNT DISK PACK 12345

The operator then selects a free disk drive; e.g., DP1, mounts the cartridge and turns the RUN button ON. After the READY light goes ON, the operator enters DP1 at the terminal.

All program references to MYFILE now become references to DP1; e.g., MYFILE:TEST becomes DP1:TEST.

!DUMP

Batch Monitor

Copy file or files from current directory to file or device in dump format

Syntax

DUMP outputfilename *[filename$_1$...] [old filename/S new filename]...*

Description

Dump a given file or files onto a given file or device. The directory information for each file (name, length, attributes, creation and last access time) is written as a header to each dumped file. If no filenames are given, all nonpermanent files are dumped. If filenames are given, no name can be preceded by a device specifier. *filename* can be either a partition or subdirectory. Dumping a directory also dumps the contents of the directory.

If, while dumping files (DUMP/A) in an environment containing several directories, you abort the dump by typing CTRL-A, you should explicitly direct the CLI to a specific directory before making any further file references.

Template characters are permitted only when the filename is in the current directory.

Global Switches

/A Dump all files, permanent as well as 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 20 Kbytes each. Each segment of tape has a unique segment number placed in its dump heading, enabling the system to verify that tapes are later reloaded in proper sequence.

/V Verify dump with a list of names of dumped files on the console.

Local Switches

/A Dump only the files created on this date or after. Preceding argument is of the form mm-dd-yy, where mm and dd can be one or two digits.

/B Dump only the files created before this day. Preceding argument is of the form mm-dd-yy, where mm and dd can be one or two digits.

/N Don't dump files that match name.

/S Assign a new name to a file output in a dump (but retain its old name in the current directory).

!DUMP (continued)

Examples

1. !DUMP/A MT0:0 12-20-84/A ↵

   Dump all permanent and nonpermanent files created on or after Dec. 20, 1984 onto file 0 of magnetic tape unit number 0. This tape file can then be saved as backup for the disk.

2. !DUMP/L DUMPFI-.SV ↵
   EDIT.SV
   ASM.SV
   RLDR.SV

   Dump all nonpermanent files with the extension SV to the file DUMPFI and output a list of files dumped on the line printer.

3. !DUMP/A MT0:0 ABC/S DEF ↵

   File ABC is named DEF in the dump (but its name remains ABC in this system) even if ABC is permanent. DEF is dumped on file 0 of MT0.

!EOF

Batch Monitor

End user job stream

Syntax

!EOF

Description

Specify the end of a user job stream. This command is optional, and an end-of-file mark can be used in its place. No switches.

!EXEC

Batch Monitor

Execute a save (.SV) file

Syntax

!EXEC *[program_name] [program arguments]* !program_name *[program arguments]*

Description

Execute a save file. If no program name is given with the !EXEC command, the currently loaded core image (TMP.SV) is executed. Entering !program_name by itself is equivalent to and may be substituted for entering !EXEC program_name. No switches.

Program arguments are passed on to the communications file, FCOM.CM or COM.CM, created for this program.

If any data sets are required for the user program to execute, and these data sets do not exist as a disk file, the data must follow this Batch command, and must be accessed as file TMP by the user program.

Example

Give the following series of commands:

```
!JOB SMITH ↵
!FORT ↵
!RLDR TMP FORT.LB SMITH/S ↵
!EXEC SMITH ↵
!EOF ↵
```

Job SMITH performs a FORTRAN IV compilation, load, and execution. The command !SMITH could have been used instead of !EXEC SMITH.

!FILCOM

Batch Monitor

Compare two files

Syntax

!FILCOM filename$_1$ filename$_2$ *[listing file/L]*

Description

Compare two files, word by word, and print dissimilar word pairs. Dissimilar word pairs at the same displacements in the two files are printed on the terminal, or in a listing file if the /L switch is used. Words are printed in octal and the displacement where they occur in the files is also printed. File organizations of the two files can differ.

Global Switches

None.

Local Switches

/L Listing file.

Example

!FILCOM YIN YANG $LPT/L ↵

Compare the contents of files YIN and YANG word by word. Any dissimilar word pairs will be printed in octal on the line printer, beside their respective word displacements in the files:

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

If YANG were a null file, the entire contents of file YIN would be printed. Dashes would be printed for file YANG.

!filename

Batch Monitor

Execute the program or macro named filename

Syntax

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

Description

Direct the Batch Monitor to find and execute the file named **filename**. Batch 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 a .SV and .MC version of a file in the same directory, you can dirrect 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, or FCOM.CM for a foreground program.

Switches are user-definable.

Examples

1. DP1:MYFILE ↵

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

2. MYFILE.SV ↵

   Execute MYFILE in the current directory.

3. DP0:EDIT ↵

   Execute the text editor program in DP0.

!FORT

Batch Monitor

Compile a FORTRAN IV source file

Syntax

!FORT inputfilename *[outputfilename]*

Description

Compile a FORTRAN IV source file. Output may be a relocatable binary file, an intermediate source file, a listing file, or combinations of all three. The command name, FORT, must be used in compiling FORTRAN source programs; the name, FORT, cannot be changed.

By default, executing the command produces an intermediate source file, *inputfilename.SR* (output of compilation) and a relocatable binary file, *inputfilename.RB* (output of assembly). However, once assembly has been successfully completed, the intermediate source file is deleted. No listing is produced by the default command.

Global Switches

/A Suppress assembly. (Intermediate source file is deleted by default.)

/B Give brief listing (compiler source program input only).

/E Suppress error messages from compiler at SYSOUT. (Assembler error messages are not suppressed.)

/F Give equivalent FORTRAN variable names and statement numbers to symbols acceptable to the assembler.

/L Produced listing on SYSOUT.

/N Do not produce relocatable binary file.

/P Process only 72 columns per record or 72 characters per line (as in punched card images).

/S Save the intermediate source output file.

/X Compile statements that contain X in column 1.

Local Switches

/B Direct relocatable binary output to given filename. (Overrides global /N.)

/E Direct error messages to a given filename.

/L Direct listing output to given filename. (Overrides global /L.)

/S Direct intermediate source output to given filename.

Extensions

On input, search for filename.FR. If not found, search for filename. By default, search for TMP. On output, produce *TMP.RB* by default and *TMP.LS, TMP.SR* as described under switches and examples.

!FORT (continued)

Examples

1. !FORT/L ⤸

   Produce relocatable binary file TMP.RB with both compiler and an assembler listing on SYSOUT.

2. !FORT/N MYPAK:TABLE INTAB/S ⤸

   Compile the file TABLE from logical disk pack MYPACK and produce compiler source and assembly listing on SYSOUT and send intermediate source output file, INTAB, to the default directory. Do not produce a relocatable binary file from the assembly.

3. !FORT/A/L/S TABLE ⤸

   Produce and save intermediate source file TABLE.SR and listing file TABLE.LS containing compiler source input listing. Assembly is suppressed. (Note that /A implies /B.)

!FORTRAN

Batch Monitor

Compile a FORTRAN 5 source file

Syntax

!FORTRAN inputfilename *[outputfilename]*

Description

Perform a FORTRAN 5 compilation. Output may be a relocatable binary file, a listing file, or both. The command name FORTRAN must be used in all FORTRAN 5 compilations, and cannot be changed.

By default, execution of the command produces a relocatable binary file, *inputfilename.RB*. No listing is produced by default.

Global Switches

/B Give brief listing (compiler source program input only).

/C Check syntax only.

/D Debug aid. Line number and program name output on all run time error messages.

/L Produce listing on SYSOUT.

/S Generate code to do subscript checking.

/X Compile statements with X in column 1.

Local Switches

/B Direct relocatable binary output to given filename.

/E Direct error output to a given filename.

/L Direct listing output to a given filename. (Overrides global /L.)

Extensions

On input, search for filename.FR. If not found, search for filename. By default, search for TMP. On output, produce *TMP.RB* by default and *listingfile.LS* if local switch /L is given.

Examples

1. !FORTRAN/L MAIN ⤸

   Produce relocatable binary file MAIN.RB with both a compiler and a source listing on SYSOUT.

2. !FORTRAN FLIST/L ⤸

   Compile a FORTRAN 5 source program immediately following in the job stream and output a compiler and source listing to disk file FLIST.LS.

!FPRINT

Batch Monitor

Print a disk file in the specified format

Syntax

!FPRINT filename *[filename/L]*

Description

Print any readable disk file in either byte, decimal, hexadecimal, or octal format, with any printable ASCII characters printed on the right side. Any nonprinting characters are reported as periods (.). The location counter is always in octal.

Global Switches

/B Print in byte format.

/D Print in decimal format.

/H Print in hexadecimal format.

/L Output to line printer.

/O Print in octal format (default).

/Z Print locations starting at 0.

Local Switches

n/F Start at location n (octal).

name/L Overrides global /L, directs output to filename that precedes it.

n/T Stop at location n.

Example

```
!FRPINT/L TE1 )
```

Lists TE1 on the line printer. The mode is octal by default. The default output device is SYSOUT.

!GDIR

Batch Monitor

Print the current directory name

Syntax

!GDIR

Description

Print on SYSOUT the name of the directory in which you are currently working. No switches.

Example

```
!GDIR ↓
DK0
```

The message indicates that DK0 is the current default directory device.

!GMEM

Batch Monitor

Output background and foreground memory size allocations (Mapped RDOS only)

Syntax

!GMEM

Description

Display the current size of memory allocations for the foreground and background program areas. The size is given in 2048-byte blocks. No switches.

This command is useful only on machines with mapped addressing.

Example

```
!GMEM ↵
BG:19 FG:18
```

38,912 bytes of memory (19 times 2048) are available to the background, and 36,864 bytes of memory are available to the foreground.

!GSYS

Batch Monitor

Display the name of the current operating system

Syntax·

!GSYS

Description

Output the name of the current operating systeem on SYSOUT. No switches.

Example

```
!GSYS ⤸
SYS
```

The current operating system name, SYS, is output to SYSOUT. Note that the system does not include the save file extension (.SV) in the response.

!GTOD

Batch Monitor

Display the current date and time of day

Syntax

!GTOD

Description

Get the time of day and the current date, and output them to SYSOUT.

Example

```
!GTOD ↓
02/17/84 21:24:20
```

The message indicates that the time is 9:24:20 p.m., and the date is February 17, 1984.

!JOB

Batch Monitor

Identify beginning of job

Syntax

!JOB identification

Description

Identify the beginning of a job. The argument identification is a label assigned to the job. Up to 132 characters are allowed in the label, including alphanumerics and $. No switches.

This command delimits jobs within a job stream. Thus the Batch Monitor assigns all commands following !JOB to the same job until the Batch Monitor detects a new !JOB command.

Example

Given the following series of commands, two jobs would be defined in the job stream, SMITH and JONES.

```
!JOB SMITH ⤸
!ASM SYSOUT/L ⤸
(source deck)
!RLDR SMITH/S ⤸
!EXEC SMITH ⤸
!JOB JONES ⤸
!FORT ⤸
(source deck)
!RLDR TMP FORT.LB JONES/S ⤸
!EXEC JONES ⤸
!EOF ⤸
```

Job SMITH assembles, with listing to the system output device, loads, and executes the save file names SMITH. Job JONES performs a FORTRAN IV compilation and assembly, loads the program (calling it JONES), and executes JONES.

!LFE

Batch Monitor

Edit library files

Syntax

!LFE { A inputmaster *[...argn] [listing_device/L]*
A/M inputmaster1 *[...inputmastern] [listing_device/L]*
D inputmaster *[outputmaster/O]* arg1 *[...argn]*
I inputmaster *[outputmaster/O]* file1*[...filen]*
M outputmaster/O inputmaster1 *[...inputmastern]*
N *[outputmaster/O]* file1 *[...filen]*
R inputmaster *[outputmaster/O]*arg1file1 *[...argnfilem]*
T inputmaster *[listing-device/L][...inputmastern]*
X inputmaster arg1 *[...argn]* }

Description

Edit and analyze library files, which are sets of relocatable binary files having special starting and ending blocks, and which are usually designated by the extension .LB.

A, D, I, M, N, R, T, and X are keys designating LFE functions; inputmaster and outputmaster represent library files; arg represents logical records on the library files, and files are update files.

Action taken by the LFE depends upon the function given in the command.

A Analyze global declarations of inputmaster or a series of inputmasters, or of logical records specified from one inputmaster. Output is a listing with symbols, symbol type, and flags; no new output library file is created. Default output is to SYSOUT.

D Delete logical records, specified by args from inputmaster, producing *outputmaster*. Default output is to diskfile D.L1.

I Insert relocatable binary files, merging with logical records of inputmaster in the manner described under "Switches." Default output is to disk file I.L1.

M Merge library file (inputmasters) into a single library file named outputmaster. Default output is to disk file M.L1.

N Create new library file, *outputmaster*, from one or more relocatable binary files given by files. Default output is disk file N.L1.

R Replace logical records in inputmaster by relocatable binary files, producing *outputmaster*. Arguments are paired, with the first being the logical record and the second the relocatable binary file that replaces the logical record. Default output is to disk file R.L1.

T Output to the listing device (SYSOUT by default) the titles of logical records on inputmaster.

X Extract from library file, inputmaster, one or more relocatable binary files given by args. Output is one or more relocatable binary files named args.RB.

The name LFE cannot be changed.

Key Switches

/M Multiple input library files. The switch modifies the A function (not the filename LFE) and causes all library file names following, except the listing file, to be analyzed as one library.

Local Switches

/A Insert after. The switch modifies a logical record in an I function command line. Arguments following the switches are inserted after the logical record whose name precedes the switch. When neither a /A nor /B switch is given, inserts are made at the beginning of the new library file.

/B Insert before. The switch modifies a logical record in an I function command line. Arguments following the switch are inserted before the logical record whose name precedes the switch. When neither a /A nor /B switch is given, inserts are made at the beginning of the new library file.

/E Error listing directed to given filename.

/F Output analysis of each relocatable binary on a separate page (used only with /L).

/L Listing file. The switch modifies the name of a file to be used as listing output in the A function command line. (SYSOUT is used by default.)

/O Output library file. The switch always modifies outputmaster in D, I, M, N, and R functions.

Extensions

If the .LB extension for inputmaster or the .RB extension for an update file are not given in the command, LFE searches for inputmaster.LB or arg.RB, respectively. If not found, LFE searches for inputmaster or arg, respectively.

Examples

1. !LFE N $PTP/O A.RB C.RB ↵

   Create a library file, output to the punch from two disk files, A.RB and C.RB.

2. !LFE R MATH.LB $PTP/O ATAN $PTR TAN/2 ↵

   Output a new library file to the PTP, replacing ATAN on input file MATH.LB by a file on the PTR and replacing TAN on the input file by disk file TAN, or if not found, TAN.RB.

3. !LFE A/M MATH1.LB $PTR LIST/L MATH2.LB ↵

   Analyze library file MATH1.LB, $PTR, and MATCH2.LB as one library and write the listing output to disk file LIST.

!LINK

Batch Monitor

Create a link entry to a file

Syntax

!LINK linkfilename *[directory specifier:]*filename

Description

Create a link entry to another link or to a resolution file in the same or a different directory. The current directory is always presumed unless you specify another directory.

If a resolution file name is given that differs from the link filename, the link filename is called an alias. The resolution file can exist either in the directory or, if a directory specifier precedes the filename, in some other directory or partition.

Note that the creation of a link in no way implies that the resolution file exists; thus links can be made to nonexistent files. The system detects an unresolved link only when an attempt is made to access the resolution via the link file, at which time the system returns the error message *FILE DOES NOT EXIST*. No switches.

Example

```
!LINK ASM.SV DP0:ASM.SV ↵
```

Create a link entry named ASM.SV in the current directory to ASM.SV in the master directory DP0.

!LIST

Batch Monitor

List file information

Syntax

!LIST [*filename*$_1$... / *[primary partition:][secondary partition:] [subdirectory:] filename*]

Description

List information from the default directory or from any other directory about one or more files or link entries. If LIST has no *filename* argument, all nonpermanent files and link entries in the current directory are listed.

Template characters are permitted only when the filename is in the current directory.

Information that is listed for the files includes the file name and, depending on switches, one or more of the following: file size (in bytes), resolution file attributes, link access attributes, file creation date and time, date last opened, file starting address (UFTAD of UFD), and decimal file use count. The resolution file attributes and link access attributes, if any, are separated by a slash (/). The file starting address is enclosed within brackets, and the file use count is terminated with a period to indicate that it is a decimal figure.

The following information is listed for link entries: the link entry name, (link characteristic) and the resolution entry name with the directory specification name that was given when the link was created. An at sign (@) is printed when the resolution file was defined to exist in the current primary position.

The following sections list the symbols for file attributes and characteristics. See the !CHATR command for more information.

File Attributes

N Not a resolution entry, cannot be linked to.

P Permanent file, cannot be deleted or renamed.

R Read-protected file, cannot be accessed for reading.

S Save file; having been set, cannot be removed.

W Write-protected file, file cannot be altered.

0 No attributes.

* Current file attributes.

? First user-defined attribute.

& Second user-defined attribute.

Characteristics

A Attribute-protected file.

C Contiguously organized file.

D Randomly organized file.

!LIST (continued)

I Accessible by direct I/O only (SYS.DR and MAP.DR only).

L Link entry.

T Partition file.

Y Directory file.

Global Switches

/A List all files within the current directory, both permanent and nonpermanent files, giving filename, byte count, file attributes, and file characteristics.

/B Brief listing; gives only the filename.

/C List the creation time (year/month/day hour:minute).

/E List every category of file information (overrides /B, /C, /F, /O, and /U switches).

/F List the first address, i.e., the logical address of the first block in the file; list 0 is unassigned. (Enclosed within brackets.)

/K Do not list links.

/L Output listing on line printer.

/O List date file last opened (month/day/year).

/S Sort the output list alphabetically.

/U List file use count (in decimal terminated with a period).

Local Switches

/A List only files created this day or later. Proceeding argument is of the form mm-dd-yy where mm and dd can be one or two digits.

/B List only files created before this date. Proceeding argument is of the form mm-dd-yy where mm and dd can be one or two digits.

/N Do not list files that match this name.

Examples

1. !LIST/E/A ↵

   List all information for all files and link entries in the current directory, sending output to the terminal. A typical line of information describing a file would look like the following:

   FLI.SV 8160 SD 03/23/82 13:56 03/23/84 [004164] 0

   In this example, FLI.SV is the filename; it consists of 8,160 bytes, is a randomly organized save file, was created on 1:56 p.m., March 23, 1984, was last accessed on that same date, has a starting logical block address of 4164, and has a file use count of 0.

2. Typical lines describing link entries would look this way:

   !LIST ↵
   ABC.SV DP0:DEF.SV
   XXX.SV @:XXX.SV

   In the first link example, the link entry name is ABC.SV and the resolution file is defined with alias DEF.SV residing in primary partition DP0. In the second example, the link entry name is XXX.SV and the resolution file has the same name and resides in the primary partition. Link entries are always indicated by having 0 byte count and only the link characteristic.

!LOAD

Batch Monitor

Reload dumped files

Syntax

!LOAD inputfilename *[filename$_1$...]*

Description

Load or list a previously dumped file or files from a given device or directory into the current directory. If no file names are specified, all files as specified by switches in the input file are loaded. The LOAD comand can be used to list or load only those files that were previously dumped with the DUMP command, not files copied with FDUMP or XFER. Files to be loaded must bear names distinct from files exisiting in the current directory (unless /R or /N is given).

If a file to be loaded has partition, directory, or link characterisitics, these characteristics will persist after the load. In the case of a partition, the partition will be created with necessary disk file space. It is possible to load files selectively from any dumped directory. Neither the DUMP nor the LOAD command changes file access or file creation information specified for the files.

If files were dumped in segments using the DUMP/S command, these files must be loaded in the same sequence that they were dumped in. Failure to follow the same sequence will result in a CLI runtime error message.

Template characters are permitted only when the filename is in the current directory.

Global Switches

/A Load all files, including permanent files.

/B Give brief listing.

/E Suppress nonfatal error messages.

/I Ignore any checksum errors.

/K Do not load links.

/N Only list the files, do not load them; output list on terminal. If global /R is used with /N, only the most recent version of the file will be listed.

/R Load most recent version of file. The file's creation date is examined. If the existing file has the same or a more recent creation date than a file awaiting loading, the existing file is not deleted. If the current file's creation date is older than a file awaiting loading, the current file is deleted and the newer file is loaded.

/V Verify the load with a list on the terminal of the names of files loaded. Subdirectory and secondary partition names will be set off with preceding and following linefeeds.

!LOAD (continued)

Local Switches

mm-dd-yy/A	Load only files created this day or later. Preceding argument is of the form mm-dd-yy where mm and dd can be one or two digits.
mm-dd-yy/B	Load only files created before this day. Preceding argument is of the form mm-dd-yy where mm and dd can be one or two digits.
name/N	Do not load files that match this name.

Example

```
!LOAD $PTR ↵
```

Reconstruct whatever previously dumped, nonpermanent files are in the paper tape reader. The files will be reconstructed on disk and with the same names. File name, length, and attributes are entered in the file directory.

!MAC

Batch Monitor

Assemble source files using the macroassembler

Syntax

!MAC filename$_1$ *[filename$_2$...]*

Description

Assemble one or more source files using the macroassembler. Output may be a relocatable binary file, a listing file, or both. The command name, MAC, must be used in assembling programs; the name MAC cannot be changed.

Global Switches

By default, assembly input is TMP and assembly output is a relocatable binary file named TMP.RB (no listing file).

/A Cross-reference all user and semipermanent symbols.

/E Suppress error printouts on SYSOUT unless there is no listing file for the current pass.

/F Generate or suppress a form feed as required to produce an even number of assembly pages. By default, a form feed is always generated.

/K Keep MAC.ST at the end of the assembly. By default, MAC.ST is deleted at the completion of an assembly.

/L Produce listing file on SYSOUT. Listings include a cross referencing of the symbol table. MACXR.SV must be available on disk.

/N Do not produce relocatable binary file.

/O Override all listing suppression controls.

/R Do not create or delete .RB files. This switch is used in conjunction with the .RB pseudo-op.

/S Skip pass 2, and save a version of the assembler's symbol table, MAS.PS, which contains new symbols and macro definitions.

/U Append user symbols to the relocatable binary output.

/Z Print the DGC proprietary license heading at the top of assembly and cross reference listings. By default, the proprietary license heading is not printed.

Local Switches

/B Send relocatable binary output to the given file name (overrides global /N).

/E Send error output to the given filename.

/L Send listing output to the given filename (overrides global /L).

/S Skip pass 2 and save a version of the assembler's symbol table, MAC.PS, that contains new symbols and macro definitions.

!MAC (continued)

Extensions

On input, search for filename.SR. If not found, and the filename did not have an extension, search for filename.

On output, produce *filename.RB* for relocatable binary and *filename.LS* for listing (global /L switch), where filename will be the name portion of the first source file specified without a /S or /B local switch given.

Examples

1. !MAC/L ZIGS ⤸

   Assemble source file ZIGS, producing a relocatable binary file called ZIGS.RB, and a listing on SYSOUT.

2. !MAC/L LIB/S ABEL BABEL CABAL ⤸

   Assemble relocatable binary files ABEL, BABEL, and CABAL. File LIB contains macro definitions and thus is skipped during the second pass. A listing and cross-referenced symbol table are produced on SYSOUT.

!MDIR

Batch Monitor

Display the master directory name

Syntax

!MDIR

Description

Type the name of the current master directory. The master directory is the directory that contains the system save and overlay files, the spool file, and push space for program swaps. Template characters are not permitted. No switches.

Example

```
!MDIR ↵
DP0
```

Display the name of the master directory DP0.

!MKABS

Batch Monitor

Make an absolute binary file from a save file

Syntax

!MKABS save_filename absolute_binary_filename

Make an absolute binary file from a core image (save) file.

Description

!MKABS gives users the facility of converting files that are executable under the operating system into absolute binary files that can be executed on another machine without RDOS.

Global Switches

/S Start at address. The starting address of the save file as specified in USTSA of the file will be used as the address for an absolute binary start block. Default is a null start block, which causes the Binary Loader (for paper tape) to halt when loading is completed.

/Z Begin save file with core location 0 (by default, begins with location 16).

Local Switches

/F Starting address (relative to location 0 of the save file) from which the absolute binary file is to be created.

/S Starting address. An absolute binary start block will be output with the starting address specified by the preceding octal argument.

/T Last address (relative to location 0 of the save file) that is to become part of the absolute binary file.

Extensions

Search for save_filename.SV. If not found, search for save_filename.

Examples

1. !MKABS FOO $PTP ⤸

   Punch an absolute binary file on the paper tape punch from file FOO.SV or, if not found, from FOO.

2. !MKABS FOO $PTP 1000/A ⤸

   Punch an absolute binary file with a start block specifying 1000 as the starting address.

!MKSAVE

Create a save file from an absolute binary file.

Syntax

!MKSAVE absolute_binary_filename save_filename

Description

Create a save (core image) file from an absolute binary file.

Global Switch

/Z Create a save file beginning at location 0 rather than 16(8).

NOTE: The save file is not executable under RDOS.

Local Switches

None

Extensions

MKSAVE produces *save_filename.SV* as output, regardless of the extension specified by the save_filename argument.

Example

```
!MKSAVE/Z $PTR PACK1:A
```

Create a core image file on logical moving head disk PACK1 called A.SV, with the S attribute, from the absolute binary file loaded in the paper tape reader.

!MOVE

Batch Monitor

Copy files from current directory to another directory

Syntax

!MOVE targetdirectory *[filename$_1$...] [oldfilename/S newfilename]...*

Description

Move a given file or files onto a given file or device. The directory information for each file — name, length, attributes, creation and last access time — is preserved. If no filenames are given, all nonpermanent files are moved. If filenames are given, no name can be preceded by a device specifier. *filename* may be either a partition or subdirectory. Moving a directory file (*filename.DR*) moves the contents of the directory too.

Templates characters are permitted.

Global Switches

/A Move all files, including permanent files.

/D Delete original file once transfer is complete.

/K Do not move links.

/L List moved filenames on the line printer (overrides /V switch)

/R Move most recent version of the file. The file's creation date is examined. If the file in the destination directory has the same or a more recent creation date than the file in the current directory, the existing file is not moved. If the current directory's file creation date is older than the file awaiting a move, the current file is deleted and the newer file is moved.

/V Verify the move with a list on the terminal of the names of the moved files. Subdirectory and secondary partition names are set off with line feeds.

Local Switches

mm-dd-yy/A Move any file created this day or after. The argument is of the form mm-dd-yy, where mm and dd can be one or two digits.

mm-dd-yy/B Move any file created before this date. The argument is of the form mm-dd-yy, where mm and dd can be one or two digits.

name/N Do not move any files that match this name.

name/S Assign a new name to a moved file (but retain its old name in the current directory).

Example

```
!MOVE/D/K MYDIR-.SR ↵
```

Move all nonpermanent files with .SR extension into the destination directory MYDIR and delete the original files once the transfer is complete; do not move links.

!MTA

Batch Monitor

Assign magnetic tape unit to a logical magnetic tape name

Syntax

!MTA logical_device_name *[tape_label]*

Description

Assign an available magnetic tape unit to a logical magnetic tape unit that is accessed in a user program. An optional tape label can be specified to identify the tape reel to the Batch operator. No switches.

When the Batch Monitor encounters this command, it directs the operator to mount the tape, identified by its tape label if given.

The operator then mounts the appropriate tape reel on an available unit, and responds at the system terminal with the name of the unit selected, e.g., MT2. The system then rewinds the tape to BOT and resets the tape file pointer to 0.

Example

Throughout an assembly language program, the programmer has consistently referred to a reel of tape files by the mnemnoic MYFILE; e.g., the third file on this reel is MYFILE:2. This reel of tape has the identification number 12345.

Upon encountering the Batch command !MTA MYFILE 12345, the system sends the operator the following message:

MOUNT TAPE 12345

The operator then selects a free tape drive; e.g., MT1, mounts the tape, places the unit on line, and enters at the terminal MT1.

All program references to MYFILE now become references to MT1; e.g., MYFILE:3 becomes MT1:3.

!OVLDR

Batch Monitor

Create an overlay replacement file

Syntax

!OVLDR save_filename overlay_descriptor/N overlay_list
[overlay_descriptor/N overlay_list]...

Description

Create an overlay replacement file. You later use the !REPLACE command to substitute the overlays in the current overlay file with those in the replacement file. The overlay replacement file created by !OVLDR bears the same name as the save file (or overlay file) with which it corresponds, but the replacement file has the extension .OR. The original save file must have been loaded with a symbol table. The name OVLDR cannot be changed.

Global Switches

/A Produce an additional symbol table listing with symbols ordered alphabetically. (The local /L switch must also be given.)

/E Output error messages to SYSOUT when a listing file has been specified (local /L). By default when a listing file has been specified, error messages to SYSOUT are suppressed. (If no listing file is given, error messages are always output to SYSOUT.)

/H Print all numeric output in hexadecimal (radix 16). By default, output is in octal.

Local Switches

/E Output error messages to given filename.

name/L List symbol table on the output file whose name precedes this switch. The table will list symbols in numeric order.

/N Separate overlay descriptor name for collection of overlay replacement relocatable binaries.

Extensions

A search is made for save_filename with the .SV extension. If not found, a search is made for the file without the .SV extension. The output overlay replacement file will bear the name *save_filename.OR*.

Example

```
!OVLDR EXAMPLE.SV MODULE1/N ^ ⤸
NEWMODULE ⤸
```

Create an overlay replacement file, EXAMPLE.OR. In this replacement file is an overlay with binary NEWMODULE, which replaces the overlay with symbolic name MODULE1 in EXAMPLE.OL. The overlay replacement does not occur until the command !REPLACE is executed.

!PAUSE

Batch Monitor

Output operator message and pause for response

Syntax

```
!PAUSE message
```

Description

Output an operator message during the execution of a job, and suspend system activity until the operator responds GO at the system terminal. No switches.

Example

The following series of commands illustrate a job in which special forms must be mounted on the line printer before the job is executed. After the job, the forms must be replaced by the standard printer paper.

```
!JOB MONTH END ↓
!PAUSE REPLACE SPECIAL FORMS ^ ↓
(3-PART FORM 678) IN LINE PRINTER ↓
!EXEC INVENTORY ↓
!PAUSE REPLACE PRINTER PAPER ↓
```

Prior to executing the inventory program, the Batch Monitor displays the following message on the operator console.

!PAUSE REPLACE SPECIAL FORMS (3-PART FORM 678) IN LINE PRINTER

The operator now inserts these forms and types

```
GO ↓
```

on the terminal. After the job terminates, the following message is sent to the operator console:

!PAUSE REPLACE PRINTER PAPER

prompting the operator to remove the special forms and reinsert the regular printer paper. Having done this, the operator types

```
GO ↓
```

to resume Batch operation.

!PUNCH

Batch Monitor

Copy a file on the paper tape punch

Syntax

!PUNCH filename$_1$ *[filename$_2$...]*

Description

Copy ASCII file or files on the paper tape punch. The command is the equivalent of the following series of !XFER commands:

!XFER/A filename$_1$ $PTP... !XFER/A filename$_n$ $PTP

The source files can come from any device. If a parity error is detected, the message *PARITY ERROR, FILE:XXX* is output on SYSOUT and a backslash (\) is punched in place of the bad character; punching then continues. No switches.

Example

```
!PUNCH ALPHA.SR BETA.SR $PTR ↵
```

Punch files ALPHA.SR, BETA.SR, and the paper tape file mounted on the paper tape reader.

!RDOSSORT

Batch Monitor

Invoke the RDOSSORT Utility (RDOS)

Syntax

!RDOSSORT infile *[infile...] [outfile/O]* key *[key...] [arguments]*

Description

!RDOSSORT invokes the RDOSSORT sort/merge utility, which can rearrange, delete, and combine disk or tape files.

The Sort function reads records from an input file (infile) and sorts them according to the key specifications, switches, and arguments you specify in the command line. If you want to produce a sorted output file, the input file must exist on disk.

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, infile is the name of a disk or tape file.

If you omit *outfile/O*, there will be no output file; this is useful if you want only a key file or listing.

The data in infile will be sorted according to the key you specify; up to eight keys are permitted. These keys make up the Control Word, which is 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
- Input file collating order
- Input file field delimiters
- Output fields
- Output files
- Work files

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, use the local /R switch to enter their byte length (in decimal).

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 using the local /S switch to specify the filename which describes your sequence.

Input File Field Delimiters

To specify a lower limit, use the /B local switch to name the file containing the lower limit for the major key field; for an upper limit, use the /U local switch. If you omit either delimiter, all records input are output.

!RDOSSORT (continued)

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 in the b.f format (i.e., b:f). If you enter no output field specifier, each record is output in the same form as you input it.

Output Files

Use the /L local switch to specify a listing file; the default is the terminal. Unless you specify a file for sorted keys with /K, no key file is produced.

Work Files

During a sort operation, RDOSSORT creates up to six work files in the current directory, and names them SORTWn.TP, where n is a number from 1 to 6. In many sorts, work file 1 is most active, followed by 4, 2, 5, 3, and 6. For 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 *RDOS/DOS Sort/Merge and Vertical Format Utilities* (DGC No. 069-400021) for more information.

Global Switches

/D Sorts records in descending ASCII order. (Default is ascending.)

/M Merges records. (The default operation is Sort.)

/N Does not list sort or merge statistics. (By default, Sort/Merge lists the statistics it produces.)

Local Switches

name/B Makes file name contain the lower limit field.

name/D Deletes input file name after sorting it. The system ignores this switch if you specify no output file.

name/K Writes sorted keys to file name.

name/L Lists sorted output on file or device name (overrides global /N).

name/O Identifies name as the sorted or merged output file.

n/R Defines decimal number n as the input record size in bytes. (Default is 80.)

name/S Specifies file name as the collating sequence. (Default is ascending ASCII.)

name/U Makes file name contain the upper limit field.

name/W Makes file name 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 ^
$LPT/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 07/02/82*
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

Batch Monitor

Release a directory or device from system initialization

Syntax

!RELEASE device_name

Description

Prevent further I/O access to a logical disk device, cassette, or magnetic tape unit, and release the associated physical device. This command must be issued before a disk cartridge, magnetic tape reel, or disk pack can be physically removed from a unit. No further access to the device is permitted until the logical name is reinitialized by a !CTA, !MTA, or !DKP command. The !RELEASE command rewinds cassette and magnetic tape reels. No switches.

A maximum of five logical disk or tape devices may be active in the system at any one time. All logical devices are released at the termination of each job. Jobs requesting a logical device when five devices are already active will cause an error message to be output and the offending job to be flushed from the stream.

Examples

1. !RELEASE PACK1 ↵

   Permits logical disk pack PACK1 to be removed and its drive to be released for use by other jobs. The operator message

   DISMOUNT DP1

   is issued, indicating that PACK1, the cartridge to be removed, is mounted on DP1.

2. !RELEASE MYTAPE ↵

   Rewind logical tape MYTAPE and release its transport for use by other jobs. The operator message

   DISMOUNT MT0

   is issued, indicating that MYTAPE is mounted on MT0.

!RENAME

Batch Monitor

Rename a file

Syntax

!RENAME oldname$_1$ newname$_1$ *[...oldname$_n$ newname$_n$]*

Description

Change the current name of a file or files. No switches.

Examples

1. !DELETE Q.SV ↵
   !RENAME QTEST.SV Q.SV ↵

   Replace the old version of Q.SV with a new version, one previously named QTEST.SV.

2. !RENAME PACK2:A1 APCK2:A B1 B ↵

   Rename file A1 to A on logical moving-head disk PACK2. Rename file B1 to B in the current dirctory.

!REPLACE

Batch Monitor

Replace overlays in an overlay file

Syntax

!REPLACE savefilename

Description

Replace overlays in an overlay file named **savefilename.OL** with overlays in a previously created overlay replacement file, **savefilename.OR**. The overlay replacement file was created with the command !OVLDR. Actual replacement occurs as soon as there are no outstanding overlay load requests. No switches.

Example

!REPLACE ABC ↵

Replace overlays in file ABC.OL with overlays in file ABC.OR. The command !OVLDR must have been used to created overlay replacement file ABC.OR.

!REV

Batch Monitor

Display the revision level of a save file

Syntax

!REV savefilename*[.SV]*

Description

Display the revision level of a save file. The save file must have the S attribute. Revision level information will be displayed as a major revision number followed by a period and a minor revision number. Both major and minor revision levels can be in the range 0–99. No switches.

The .REV pseudo-op is used to assign a major and minor revision level number to a save file. If this pseudo-op is not used in a save file, then the revision levels of this save file are displayed as 00.00.

Example

```
REV HITTITE.SV ↵
03.07
```

Display revision level of HITTITE. The response of 03.07 indicates that the major revision level of HITTITE.SV is 03, and the minor revision level of this file is 07.

!RLDR *Batch Monitor*

Load relocatable binary files to produce an executable save file

Syntax

```
                ⎧ root_binary... root_binary...                                    ⎫
                ⎪                                                                  ⎪
!RLDR           ⎨ root_binary... [ overlay_binary .....]                           ⎬
                ⎪                                                                  ⎪
                ⎪ NREL partition/F ZREL partition/Z root_binary... [root_binary... ⎪
                ⎪     [overlay_binary.....]]                                       ⎪
                ⎩                                                                  ⎭
```

Description

Create a save file from the loading of relocatable binary files, and optionally create an overlay file from other relocatable binaries. The save file is named TMP.SV by default.

By default, the system library (SYS.LB) is searched after the last binary file name. This causes modules from SYS.LB to be placed at the end of the root program.

By default, each relocatable load produces no symbol table listing and positions the symbol table so that the end of the symbol table coincides with the first address not loaded by the program only if there is a global /D. The name RLDR cannot be changed.

Global Switches

/A Produce an additional symbol table listing with symbols ordered alphabetically. (The local switch /L must also be given.)

/C Load compatible with RTOS/SOS conventions, i.e., INMAX is set at 440, save file starts at 0 (as with /Z global switch), USTSA contains the program starting address, and SYS.LB is not searched unless specified by the user.

/D Load symbolic debugger DEBUG from SYS.LB. The symbolic debugger IDEB will be loaded instead of DEBUG only if IDEB.RB appears somewhere in the RLDR command line. Note that the symbol table will not be written to the save file unless a /D switch is given.

/E Output error messages to SYSOUT, when a listing file has been specified.

/H Print all numeric output in hexadecimal (radix 16). By default, output is printed in octal.

/M Suppress load map and all error message output. This speeds loading, but should be used with caution since it suppresses all error messages.

/N Inhibit a search of the system library SYS.LB. By default, SYS.LB is searched.

/S Leave symbol table at the high end of memory.

/X Allow up to 128_{10} system overlays.

/Y Allow up to 256_{10} system overlays.

NOTE: /X and /Y are mutually exclusive switches.

Local Switches

By default, the first input filename is used with an .SV extension to form the name of the output file, and also with an .OL extension to form the overlay filename. By default, user symbols are not loaded.

/C Specify preceding octal value as the number of channels required. This value is placed in USTCH of the User Status Table and overrides any channel number appearing in a .COMM TASK statement.

/E Output error messages to the file whose name precedes the switch.

/F Specify preceding octal value as the foreground NREL partition address.

/K Specify preceding octal value as the number of tasks required. This value overrides any task number appearing in a .COMM TASK statement.

/L List symbol table on the output file whose name precedes the switch. The table lists symbols in numeric order.

/N Force NMAX, the starting address for loading a file, to an absolute address given by the octal number preceding the switch. The specified value must be higher than the current vlaue of NMAX when the argument is encountered.

/S Give save file the preceding filename with the .SV extension. Overlay file is also given the preceding filename but with the .OL extension.

/U Load user symbols from the relocatable binary preceding the switch.

/Z Make preceding octal value the foreground ZREL partition address.

Extensions

A search is made for each input file with the name root-binary.RB. If not found, then a search is made for root-binary.

The default output filename will be *rootbinary.SV*. Otherwise, the output filename will be the filename preceding the switch /S with the .SV extension.

Examples

1. !RLDR A B C PACK1:D ⤸

   Load files A, B, and C from the default directory device and D from logical disk PACK1 to produce save file A.SV on the default directory device.

2. !RLDR A/S $PTR ⤸

   Load the file in the paper tape reader and produce a save file named A.SV.

3. !RLDR/D A B C ⤸

   Load files A, B, and C together with the symbolic debugger to produce save file A.SV.

4. !RLDR A 4400/N B SYSOUT/L ⤸

   Load A and B. Loading of B starts at 4400(8). A numeric memory map is printed on SYSOUT.

5. !RLDR R0 [A, B, D,D] R1 R2 [E, F G, H] ⤸

   Load R0, R1, and R2 as a background root program with two overlay areas. Overlay area 0 is situated between R0 and R1; overlay area 1 follows binaries R1 and R2. Binaries A, B, C and D constitute overlay numbers 0, 1, 2, and 3 of node 0. Likewise, E becomes overlay 0 of node 1. F and G become overlay 1 of node 1, and H becomes overlay 2 of node 1. The overlay file contains both overlay segments and is named R0.OL.

!SAVE

Batch Monitor

Rename TMP.SV file

Syntax

!SAVE filename

Description

Rename TMP.SV to filename, deleting any previous file that bears the name being given to TMP.SV. filename will be created in the current directory. Output always has the .SV extension. No switches.

Example

Given the following series of commands:

```
!JOB JONES ↵
!ASM ↵
(source deck)
!RLDR ↵
!SAVE TEST ↵
!TEST ↵
```

Job JONES causes an assembly and relocatable load of a program to occur. At the end of the load, the save file is given the default name TMP.SV. This is changed to TEST, and any other save file on the default directory device named TEST or TEST.SV is deleted. Save file TEST.SV is then executed.

!TPRINT

Batch Monitor

Print the tuning file

Syntax

!TPRINT filename

Description

Print the tuning file. The tuning file contains the number of requests and number of failures for these system resources:

- System buffers
- System stacks
- System cells
- System overlays

filename is the name of the system tuning file that is to be printed. It has the same name as the system under which the tuning file was created. Additionally, with /O you can print an overlay frequency report if you specified one at system generation.

Global Switches

/L Print on the line printer.

/O Print overlay frequency report.

Local Switches

None.

Example

!TPRINT/L BSYS ⤸

Print the tuning file for the system BSYS on the line printer.

!TUOFF

Batch Monitor

Stop recording in the tuning file.

Syntax

!TUOFF

Description

Stop recording the use of system resources in the turning file. TUOFF does not delete the contents of the tuning file. No switches.

Example

!TUOFF ⤸

Terminate recording in the tuning file.

!TUON

Batch Monitor

Start recording in the tuning file

Syntax

!TUON

Description

Initiate recording in the tuning file of the number of requests and failures for the following system resources:

- System buffers
- System stacks
- System cells
- System overlays

No switches.

Example

!TUON ↓

Initiate recording in the tuning file.

!UNLINK

Batch Monitor

Remove a link entry

Syntax

UNLINK linkname$_1$ *[...linkname$_n$]*

Description

Delete link entries in the current directory. Link entry names may be preceded by a pathname. Template characters are permitted only when the **linkname** is in the current directory.

Global Switches

/V Verify deletion by listing on the terminal the names of deleted files.

Local Switches

filename/N Do not delete any links matching this specifier.

Example

```
!UNLINK TEST.-)
```

Delete the link named TEST with all its extensions.

!XFER

Batch Monitor

Copy the contents of a file to another file

Syntax

!XFER sourcefile destinationfile

Description

Transfer a file to another file, organizing the destination file differently as specified. If the destination file does not exist, it is created.

The system will detect parity errors in ASCII files, and it will detect parity errors in binary files on magnetic or cassette tape. When a parity error is detected in the input file, the message *PARITY ERROR, FILE:xxx* will be output to the terminal. If one or more parity errors are detected in a paper tape file, a backslash will be substituted for each bad character. If a parity error is detected in a magnetic tape or cassette file, the transfer is terminated. By default, files are transferred sequentially. Use local switches to alter data organization of files. Template characters are not permitted.

Global Switches

/A Transfer an ASCII file line by line, taking appropriate read/write action, such as inserting line feeds after each carriage return when transfer is from disk to line printer.

/B Append the source file to the destination file. The destination file must already exist.

Local Switches

destinationfile/R Organize the destination file randomly.

destinationfile/C Organize the destination file contiguously. Both source and destination files must be disk files. Be sure to use this switch when transferring .SV files.

Examples

1. !XFER $PTR Q ⤸

   Transfer the file in the paper tape reader to a disk file named Q.

2. !XFER $PTR $PTP ⤸

   Punch another tape, identical to the one read from the paper tape reader.

3. !XFER TEST1:MYFILE TEST2:MYFILE ⤸

   Transfer MYFILE from logical disk pack TEST1 to logical disk pack TEST2.

4. !XFER A B/R ⤸

   Copy file A, with random file organization, into a file named B.

End of Chapter

Chapter 10
CLI Command Dictionary

This chapter describes each CLI command and system utility command in alphabetical order. We also present tables of CLI command summaries organized by functional category — file and directory management commands, system control commands, and system utility commands. You can use the tables for quick lookups and comparisons, and use the dictionary for a fuller understanding.

Command Summaries

Tables 10-1 through 10-4 briefly list the commands according to function. Note that some commands appear in more than one functional table. The categories are

- File management
- Directory management
- System control
- System utilities

See Appendix A for a list of all CLI commands and utilities in one alphabetical table.

The commands described in Table 10-4 invoke system utilities. These are programs that the CLI executes; they are not part of the CLI, and the CLI is not active when they run. When a utility terminates (either normally or after a fatal error), it returns to the CLI.

See Chapter 9 for documentation on Batch processing and a list and dictionary of Batch Monitor commands.

Table 10-1. File Management Commands

Command	Function
APPEND	Combine two or more files.
BPUNCH	Punch a binary file.
BUILD	Build a file of filenames.
CCONT	Create a contiguous file of specified size.
CHATR	Change a file's attributes.
CHLAT	Change a file's link access attributes
CLEAR	Set file or device use count to zero.
CRAND	Create a random file.
CREATE	Create a sequential file (RDOS and DG/RDOS).
DELETE	Delete a file.
DUMP	Dump a file in CLI DUMP format.
ENDLOG	Close the LOG.CM file.
FILCOM	Compare the contents of two files.
FPRINT	Print a file in octal, decimal, hexadecimal, or byte format.
LINK	Create a link entry to a resolution file.
LIST	List file information.
LOAD	Load dumped files (from DUMP command).
LOG	Start recording in the log file LOG.CM.
MKABS	Make an absolute file from a save file.
MKSAVE	Make a save file from an absolute file.
MOVE	Copy a file to a directory.
PRINT	Print a file on the line printer.
PUNCH	Punch an ASCII file on paper tape punch.
RENAME	Rename a file.
REV	Display the revision level of a program.
SAVE	Rename a breakfile.
TYPE	Type a file on the terminal.
UNLINK	Remove a link entry.
XFER	Copy the contents of one file to another file.

Table 10-2. Directory Management Commands

Command	Function
CDIR	Create an RDOS subdirectory or DOS directory.
COPY	Copy diskette$_1$ to diskette$_2$ (DOS).
CPART	Create a secondary partition (RDOS, DG/RDOS).
DELETE	Delete a directory.
DIR	Change the current directory.
DISK	Display number of blocks used, free, and free contiguous in the current partition or DOS diskette.
DUMP	Copy current directory in dump format.
EQUIV	Temporarily rename a disk or magnetic tape (RDOS).
FDUMP	Fast dump the current directory to magnetic tape (RDOS).
FLOAD	Fast load a fast dumped (FDUMP) file into the current directory (RDOS).
GDIR	Display the name of the current directory.
INIT	Initialize a directory or device for system recognition.
LDIR	Display the name of the last current directory.
LOAD	Reload dumped (DUMP) files.
LIST	List file information.
MDIR	Display the name of the master directory.
MOVE	Copy files to a directory.
RELEASE	Release a directory or device from initialization.

Table 10-3. System Control Commands

Command	Function
BOOT	Bootstrap a system from disk.
CHAIN	Overwrite the CLI with an executable program.
CLEAR	Set file or device use count to zero.
DISK	Display the number of disk blocks used, free, and free contiguous.
ENDLOG	Stop recording the log file LOG.CM.
EXFG	Execute a program in the foreground (RDOS).
FGND	Describe the foregound program status.
GMEM	Display background/foreground memory areas (mapped RDOS).
GSYS	Display the current system name.
GTOD	Display the current system time.
INIT	Initialize a disk directory or device for system recognition.
LOG	Start recording in log file LOG.CM.
MCABOOT	Transmit a system over an MCA line (RDOS).
MESSAGE	Display a text message on terminal display.
POP	Return to the program on the next higher level.
RELEASE	Release a directory or device from initialization.
SDAY	Set the system calendar.
SMEM	Set the background/foreground memory areas (mapped RDOS).
SPDIS	Disable spooling to device (RDOS).
SPEBL	Enable spooling to device (RDOS).
SPKILL	Delete data spooled to device (RDOS).
STOD	Set system clock.
TPRINT	Print tuning file (RDOS).
TUOFF	Stop recording in tuning file (RDOS).
TUON	Start recording in tuning file (RDOS).

Table 10-4. System Utility Commands

Command	Function
ALGOL	Compile an ALGOL source file (RDOS).
ASM	Assemble a source file, producing an .RB file.
BASIC	Invoke the BASIC interpreter.
BATCH	Invoke the Batch Monitor to execute Batch job streams (RDOS).
CLG	Compile, load, and execute a FORTRAN IV source file.
DDUMP	Dump a disk, partition, or directory onto diskettes.
DEB	Debug a program.
DLOAD	Load a disk, partition, or directory previously dumped by DDUMP.
DO	Execute a CLI macro, replacing dummy arguments with specified arguments.
EDIT	Invoke the Text Editor.
ENPAT	Insert patch(es) in filename; also see PATCH.
FDUMP	Fast dump the current directory to magnetic tape (RDOS).
FLOAD	Fast load a fast-dumped (FDUMP) file into the current directory (RDOS).
FORT	Compile a FORTRAN IV source program.
FORTRAN	Compile a FORTRAN 5 source program (RDOS).
LFE	Create or edit .RB library files.
MAC	Assemble a source file into a relocatable binary (RB) file with the Macroassembler.
MEDIT	Invoke the Multiuser Text Editor
NSPEED	Edit text with the NOVA SPEED editor.
OEDIT	Edit disk file locations with the Octal Editor.
OVLDR	Create an overlay replacement file.
PATCH	Install patch(es) created by ENPAT.
RDOSSORT	Sort a file or merge files with the Sort/Merge program (RDOS).
REPLACE	Replace overlays in an overlay file.
RLDR	Process relocatable binary files to form an executable program.
SEDIT	Edit disk file locations with the Symbolic Editor.
SPEED	Edit text with the ECLIPSE Supereditor (RDOS).
SYSGEN	Generate a new operating system.
VFU	Create or load a VFU file for a data channel line printer (RDOS).

Command Dictionary Format

Each dictionary entry begins with the command name, a capsule description, and a command format, called syntax. The entries include a discussion of the command and its uses, lists of global and local switches, and one or more examples of the command. The dictionary is organized in alphabetical order.

To interpret command format properly, you must understand the typesetting 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 to 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 pathname, for example PRINT SUBDIR:FILENAME. We note exceptions in the individual entries.

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

In some cases, we have used the uparrow (^) 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 terminal. However, if you type more than 132 characters on a line, you receive the message

LINE TOO LONG

and the CLI ignores the entire line.

ALGOL

Utility

Compile an ALGOL source file (RDOS)

Syntax

ALGOL filename...

Description

The ALGOL utility compiles and assembles 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.

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.

Global Switches

/A Does not assemble the compiled file.

/B Brief listing (ALGOL input to the compiler only).

/E Suppresses console error messages from the compiler. (Assembler error messages are not suppressed.)

/L Produces listing file (filename.LS).

/N Checks with assembler, but does not produce a binary file (useful for finding errors).

/S Saves the intermediate source file, under filename.SR.

/U Appends user symbols to binary output file.

Local Switches

name/B Assigns name to the binary output file (overrides global /N).

name/E Sends error listing to file name.

name/L Sends listing output to file name (overrides global /L).

name/S Sends the intermediate source file to file name.

Examples

1. *R*
   ALGOL MAIN ⤸

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

2. *R*
   ALGOL/A/L RAY ⤸

   Compile (but do not assemble) file RAY.AL or RAY, and write the intermediate source file to disk file RAY.SR. ALGOL source listing goes to disk file RAY.LS.

3. *R*
   ALGOL/E/B SUBR $LPT/L ⤸

   Produce binary file SUBR with a brief ALGOL source listing on the line printer; suppress compiler error messages.

APPEND

Command

Copy one or more files to new file

Syntax

APPEND outputfilename inputfilename *[...inputfilename]*

Description

APPEND creates **outputfilename**, and copies the contents of the **inputfilenames** into it. The original input files do not change.

You can use a pathname argument for any of the filename arguments. No switches.

Examples

1. *R*
   APPEND TAPEFILES MT0:1 MT0:2 ↵
   R

   Creates TAPEFILES and copies into it the contents of tape files 1 and 2 from the tape on drive MT0.

2. *R*
   APPEND PTRFILES $PTR/3 ↵
   LOAD $PTR. STRIKE ANY KEY
   .
   .
   R

   Creates disk file PTRFILES, and prompts for three tapes to be loaded into the paper tape reader; copies tapes to PTRFILES.

ASM *Utility*

Assemble source file(s) to produce a relocatable binary (.RB) file

Syntax

ASM filename...

Description

The Extended Assembler, ASM, assembles one or more assmebly language source files to produce a relocatable binary (.RB) file. Once you produce the .RB file and handle any errors detected by ASM, you use the .RB file to produce an executable .SV file with the RLDR utility.

If you do not specify switches, ASM produces a relocatable binary file named *filename.RB* and no listing. *filename.RB* is taken from the first source file you specify for assembly on the command line.

By using global and local switches, you can produce the .RB file, an assembly listing, error listing, symbol tables, and cross-reference symbol listings.

For source filenames specified on the command line, the CLI searches for filename.SR first, and then for filename. For files that ASM produces, ASM uses the name of the first source file you specified for assembly on the command line. It names the relocatable binary file *filename.RB*, and names the listing file (local /L) *filename.LS*, unless you use the /B, /L, or local /S switches.

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. Table 10-5 lists the codes.

Table 10-5. Assembly Error Codes

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

For more information on the ASM Utility, refer to the *RDOS/DOS Assembly Language and Program Utilities* manual (DGC 069-400019).

Global Switches

/E Suppresses error messages at the terminal. Must use with global /L to send the messages to a printed listing file, or with local /L to send messages to a disk file.

/L Creates disk file filename.LS and sends listing to it; or appends listing if filename.LS already exists. filename is taken from the first source file specified for assembly in the command line.

/N Overrides creation of the .RB file (useful for checking assembly errors).

/S Skips pass two of assembly, and stores the assembler's symbol table in file (F)BREAK.SV. The console displays (F)BREAK. You can then rename (F)BREAK.SV (with the SAVE command) to store this symbol table that contains your own semipermanent symbols.

/T Excludes symbols in the cross-reference listing (use with global /X).

/U Includes user symbols in the .RB file.

/X Produces symbol/page cross-reference listing. Assembler file XREF.SV must be available on disk.

Local Switches

name/B Assigns name to the .RB file, instead of using the name of the first source file specified on the command line. (Overrides global /N.)

name/E Sends error messages to file name.

name/L Sends the listing to file name (overrides global /L).

name/N Excludes assembly listing of file name from assembly listings of other files (specified by a global or local /L).

name/S Skips file name on pass two of assembly. Use this switch only for files that do not contain any storage words.

Examples

1. *R*
   ASM/N FILE1 ⤸

   Assembles source file FILE1 and sends error messages to the terminal; /N suppresses creation of the .RB file. This checks for errors, which you can correct before creating the .RB file.

2. *R*
   ASM MYFILE $PTP/B $LPT/L ⤸

   Assembles MYFILE; punches the .RB file on the paper tape punch ($PTP/B); produces a listing on the line printer ($LPT/L).

3. *R*
   ASM/L (A,B,C,DP4:D) ⤸

   Assembles as separate files A, B, C, and D in DP4. Produces A.RB, B.RB, C.RB, and D.RB, and listing files A.LS, B.LS, C.LS, and D.LS; places all output files in the current directory. See Chapter 2 for an explanation of the use of parentheses.

4. *R*
   ASM/L PARU/S PRIG1 PRIG2 ⤸

   Assembles PRIG1 and PRIG2, producing PRIG1.RB; scans the user parameter file PARU.SR on pass one to find values for symbols in PIG1 and PRIG2. Sends the listing to PRIG1.LS.

BASIC

Utility

Invoke the BASIC interpreter

Syntax

BASIC

Description

Invoke the BASIC interpreter, which allows you to work with the BASIC language interactively. A BASIC .SV file, configured via BASIC System Generation (BSG), must exist before the system can execute this command. System generation procedures for BASIC are described in *Extended BASIC System Manager's Guide* (DGC No. 093-000119).

BATCH *Utility*

Invoke the Batch Monitor to execute a job stream (RDOS only)

Syntax

BATCH *[jobfile(s) ...] [outfile/O] [logfile/G]*

Description

BATCH invokes the Batch Monitor to 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 can also contain data sets and can refer to devices. Many CLI-derived commands and RDOS utilities are available under Batch, as detailed in Chapter 9.

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, the Monitor searches for jobfile.

See Chapter 9 for more information on batch processing and batch commands.

Local Switches

filename/O Sends output to filename; default (line printer) name is SYSOUT.

filename/G filename 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
```

Defines a job stream with jobs serially input in two files; the first is disk file DAILY.JB, and the second is file 1 of MT0. Sends log information about the batch job to a disk file named LOG. The line printer is SYSOUT by default.

BOOT *Command*

Load and run an operating system or a stand-alone program

Syntax

BOOT { disk / *[directory:]*sysname }

Description

BOOT releases the current system and bootstraps a new system or a program into execution.

The argument sysname can be the filename of an operating system program file, or a stand-alone program file (see "Stand-Alone Programs," below). It can also name a link entry to an operating system file, 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 partition containing sysname becomes the master directory.

The program xBOOT.SV must reside in the primary partition of the disk that holds sysname. xBOOT.SV is ABOOT.SV, EBOOT.SV, or MBOOT.SV, depending on your system configuration.

When you use the BOOT disk format, BOOT displays a *FILENAME?* prompt. Enter the system filename or the name of a stand-alone program; include the /A switch with stand-alone programs. You can respond to *FILENAME?* with a line terminator (NEW LINE or CR) to bootstrap a system with the name SYS.SV, the default system name. You must use the BOOT disk format to execute certain stand-alone programs.

All directories involved in the BOOT command must be initialized on the system. A disk named as an argument must be a valid RDOS, DOS, or DG/RDOS disk. A pathname used as an argument cannot include an RDOS subdirectory or DOS directory.

Stand-Alone Programs

A stand-alone program is a program that runs without an operating system. You can execute a stand-alone program from the CLI with the BOOT command if the program conforms to the following rules:

1. The save (.SV) file must be randomly organized. (All .SV files created with RLDR are random by default. However, using XFER to transfer a file can change the file's organization.)
2. The stand-alone program 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 starting address of the .SV file, or −1. If it contains the starting address, the program will self-start. If location 2 contains −1, the computer will halt after the load; press CONTINUE to continue, or, on machines with Virtual Console, type P.
4. Location 5 must contain 0.

See the RLDR global switches /C and /Z for loading information.

To bootstrap a program that does not follow these rules, execute a MKSAVE/Z command on the program; then use the BOOT command naming the directory that contains the program. When BOOT asks

FILENAME?

Type the program name, and append the /A switch. BOOT loads the program; you must then execute it via the front panel switches or the Virtual Console.

Restart Feature

For ECLIPSE and NOVA computers, except microNOVAs, if the data switches are all up when you type BOOT, BOOT tries to bootstrap the system specified. If you omit the sysname, BOOT tries 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 do not want to invoke this feature, make sure the data switches are not all up when you type BOOT.

For microNOVA systems, see the .BOOT system call in the system reference manual for restart features.

Local Switches

/A When you use the BOOT disk form, it issues a *FILENAME?* query. If you append /A to your filename response, you inform BOOT that you are loading a stand-alone program that does not conform to default BOOT conventions.

Examples

1. *R*
   BOOT DP4 ⤸
   MASTER DEVICE RELEASED
   FILENAME? MYSYS ⤸
   .
   .

   Loads BOOT.SV from DP4, releases the master directory, and asks for the system name (*FILENAME?*). When you supply the system name (MYSYS), BOOT bootstraps MYSYS from disk directory DP4. MYSYS requests the date and time, then activates the CLI.

2. *R*
   BOOT DP0F:SYS64K ⤸

   Bootstraps the system named SYS64K from directory DP0F.

3. *R*
   BOOT DP0:RTOS ⤸

   Bootstraps the RTOS system save file from directory DP0.

4. *R*
   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

Command

Punch binary file(s)

Syntax

BPUNCH filename *[...filename]*

Description

Punch a file or files in binary on the paper tape punch. To punch an ASCII file, use PUNCH. The original files can reside on any device. No switches.

The BPUNCH command is the equivalent of the following XFER command:

XFER filename(s) $PTP

Examples

1. *R*
   BPUNCH FEE.SR FI.SR FO.RB FUM.RB ↵
   R

   Punch source files FEE and FI, and relocatable binary files FO and FUM on the paper tape punch.

2. *R*
   BPUNCH $PTR ↵
   R

   Punch a paper tape duplicate of the tape in the paper tape reader.

BUILD

Command

Build a file consisting of filenames

Syntax

BUILD outputfilename *[inputfilename ...]*

Description

BUILD creates an output file containing the string of characters following the output filename. The string is usually a collection of filenames. You can use template characters and date switches to select the filenames. Build looks in the current directory to satisfy the templates and date switches you set. You can edit the resulting output file as needed.

BUILD is useful for creating a file containing filename arguments. You can use the output file as an argument to a command by invoking it as an indirect file; the command then uses each filename in the output file as a filename argument. This is particularly helpful with commands or utilities that do not allow the use of templates, or for using a given group of filenames repeatedly.

BUILD deletes outputfilename (if it exists) before creating the new output file.

Global Switches

/A Includes all filenames, whether or not they have permanent attributes. (By default, BUILD does not recognize files that have attribute P (permanent).)

/K Excludes all link entry filenames.

/N Excludes extensions; writes filenames to the output file without including their filename extensions.

Local Argument Switches

mm-dd-yy/A Includes only filenames whose creation dates are this date or after.

mm-dd-yy/B Includes only filenames whose creation dates are before this date.

Arguments mm (month) and dd (day) can be one or two digits.

name/N Excludes from the output file any filenames that match name. name can include template characters.

Examples

1. *R*
   BUILD ABC -.SR TEST-.↵
   R

   Creates file ABC, and writes two categories of filenames into it: those whose names have a .SR extension, and those whose names begin with TEST and have no extension.

2. *R*
   BUILD RBFILES -.RB ABC.RB/N↵
   R

   Creates RBFILES, and includes all relocatable binary files (i.e., all filenames that have a .RB extension) except for file ABC.RB.

BUILD (continued)

3. *R*
   BUILD RECENT -.SR 8-2-84/A ⤸
   R

   Builds file RECENT from all filenames with .SR extensions created on or after August 2, 1984.

4. *R*
   BUILD TAPEFILES -.SR TEST-.SR/N ⤸
   R
   LIST/S/A/E @TAPEFILES@ ⤸
   FOO.SR 266 D 07/06/84 16:31 07/06/84 [006564] 0
   .
   .
   .
   R

   The BUILD command creates TAPEFILES, which contains all filenames that have a .SR extension, excluding .SR files whose filenames begin with the letters TEST.

   The LIST/S/A/E @TAPEFILES@ command line invokes TAPEFILES as an indirect file (by enclosing the filename with @ symbols), as an argument to the LIST command.

   The LIST/S/A/E command lists the size, date, and characteristics of each filename contained in file TAPEFILES, displaying the files in alphabetic order.

CCONT

Command

Create a contiguous file

Syntax

CCONT filename blockcount *[...filename blockcount]*

Description

CCONT creates a contiguously organized file in the current directory, or in the specified directory. The **filename** you assign can be in a pathname. The file will have the fixed length you specified as the **blockcount** argument; **blockcount** is a decimal number (one block equals 512 bytes). The file will have the characteristic C, which indicates a contiguous file.

To insert information into the file, use a text editor or the CLI command XFER. The data in the file cannot exceed the length you specify in blockcount.

Global Switch

/N Do not zero the data words in the file. Without /N, CCONT fills the file with null values, which takes time.

Examples

1.
```
R
CCONT NEWFILE 16 ↵
R
LIST NEWFILE ↵
NEWFILE 8192 C
R
```

Creates the contiguous file, NEWFILE, in the current directory, and allocates it a fixed length of 16 contiguous disk blocks. Note that LIST returns the size in bytes: 512 (bytes per block) multiplied by 16 (number of blocks) equals 8192 bytes.

2.
```
R
CCONT FILE4 24 DP1:PART2:ACCESS 48 ↵
R
LIST FILE4 DP1:PART2:ACCESS ↵
FILE4 12288 C
DP1:PART2:ACCESS 24576 C
R
```

Creates FILE4 in the current directory, allocating a fixed length of 24 disk blocks to the file; and creates the file ACCESS in secondary partition PART2 on disk DP1, allocating 48 disk blocks to the file.

CDIR

Command

Create a subdirectory

Syntax

CDIR directory_name

Description

CDIR creates a subdirectory, a variable-length directory in which to store files. Subdirectories can reside in a primary or secondary partition of a disk. The subdirectory is the lowest level of the three-tiered RDOS directory structure.

The directory_name argument supplies the filename for the directory; the directory_name argument can be in a pathname. RDOS appends a .DR extension (the filename extension for directories) to the name you supply.

RDOS assigns the subdirectory an initial length of 512 bytes (one block), and assigns the file characteristics DY, where:

D denotes a random file

Y denotes a directory

You should assign each of your subdirectories a unique name, as you can initialize only one subdirectory of a given name at any one time.

You can create subdirectories from within a primary partition or a secondary partition. If you attempt to use CDIR to create a subdirectory within another subdirectory, CDIR does not execute and you receive the message *DIRECTORY DEPTH EXCEEDED.*

No switches.

Examples

1. *R*
   DIR SECPART ⤸
   R
   CDIR WHATEVER ⤸
   R

   Creates WHATEVER.DR as a subdirectory to secondary partition SECPART.

   When you use the name of a subdirectory as a directory name argument, for example with the DIR command, you do not need to include the .DR extension:

   R
   DIR WHATEVER ⤸
   R

   When you use the name of a subdirectory as a filename argument, for example with the LIST command, you must specify its extension:

   R
   LIST WHATEVER.DR ⤸
   WHATEVER.DR 512 DY ⤸
   R

2. *R*
   CDIR DP1:DRONER ⤸
   R

   The previous example creates subdirectory DRONER.DR on disk DP1 (a primary partition), using a directory specifier to indicate that the subdirectory is to be created in DP1. Equivalent commands would be

   DIR DP1 ⤸
   CDIR DRONER ⤸

CHAIN

Command

Load and run a program in place of the CLI

Syntax

CHAIN program_name

Description

CHAIN overwrites the CLI by chaining program_name (a .SV file) into execution on the current level. Under RDOS, use CHAIN only in special circumstances and with caution. The program should chain control back to the CLI with the .EXEC system call.

CHAIN is useful for programs that use five swap levels. (When you execute a program from the level 0 CLI, it runs on level 1; hence it has only levels 2, 3, and 4 available.) In DOS only, CHAIN can speed up utility program commands (e.g., CHAIN ASM MYFILE).

In RDOS, if the program you chain issues a .RTN instruction from level 0 in the background, the system halts in exceptional status. Also, the CTRL-C CTRL-A or CTRL-C CTRL-B sequences 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 Passes control to the debugger.

Example

```
R
CHAIN BEHEMOTH ↵
.
.
.
```

Program BEHEMOTH.SV executes on the level normally used by the CLI. The program uses the .EXEC system control to return control to the CLI.

CHATR

Command

Change a file's attributes

Syntax

CHATR filename *[sign]* attributes *[...filename [sign] attributes]*

Description

Assigns or removes file access attributes for a file. This controls the types of access permitted to this file. No switches.

sign can be + (plus), to add an attribute, or − (minus), to remove an attribute. When you specify an attribute with no plus + or −, the new attribute replaces any existing attributes.

To remove all attributes (except attribute S), enter 0 as an argument. A file having no attributes allows full access to the file, except for execution access. In order to execute a file, the file *must* have the S attribute. The RLDR utility assigns this attribute when it creates an executable program file.

The LIST command displays file access attributes for a file as part of its output, as follows:

FILENAME sss xxxx

where *sss* represents the size of the file (in bytes) and *xxxx* represents the letters designating file characteristics and attributes. File attributes and characteristics are listed below.

The file access attributes for a file apply to all methods of accessing the file, including:

- direct access from within the file's directory
- direct access through use of a directory specifier
- indirect access through a link file (see the LINK command)

Closely related to CHATR is the CHLAT command. CHLAT assigns a *link* access attribute to a file, which controls indirect access to the file, but does not affect direct access.

File Access Attributes

N — Prohibits links to this file. (You can create a link to the file from another directory, but the link will not work.)

P — Makes this a permanent file. The file cannot be deleted or renamed while it has this attribute. Note that the following commands ignore permanent files, unless you include a local /A switch: BUILD, DUMP, LIST, LOAD, MOVE.

R — Read-protects this file. The file cannot be displayed, or copied via DUMP, FDUMP, MOVE, or XFER, but it can be executed.

S — Permits execution of this file. RLDR assigns this attribute when it creates an executable program file. If you remove this attribute, the file cannot be executed.

W — Write-protects this file. The file cannot be modified. (Note that the W attribute does not protect a file from deletion.)

0 — Removes all removable attributes, except attribute S.

* — Retains all existing attributes.

? — User-defined attribute.

& — User-defined attribute.

CHATR (continued)

File Characteristics

A Attribute-protected. (You can set A with .CHATR system call. Once set, A cannot be removed, and you cannot assign or remove other attributes.)

C Contiguous file, assigned when the file is created.

D Random file, assigned when the file is created.

L Link entry, assigned when a link file is created.

T Partition, assigned when the file is created.

Y Directory, assigned when the file is created.

Note that a combination of attribute P and characteristic A on a file makes that file permanently resistant to deletion. Use this combination with caution.

Examples

1. *R*
   CHATR OLDFILE 0 NEWFILE R ⤸
   R

   Removes all attributes of OLDFILE, and replaces any existing attributes for NEWFILE with the read-protect attribute (R).

2. *R*
   CHATR MYFILE −R +W ⤸
   R

   Removes the read-protect attribute (R) from MYFILE, and assigns the write-protect attribute (W). Any attributes previously assigned remain the same.

3. *R*
   CHATR PASSWORDS 0 ⤸
   R

   Removes all attributes from file PASSWORDS, except S if the file already has the attribute S.

4. *R*
   CHATR PASSWORDS R ⤸
   R

   Assigns the read-protect attribute (R) to file PASSWORDS. Commands that display the contents of files are not effective for this file unless you first remove the R attribute.

5. *R*
   LIST PASSWORDS ⤸
   PASSWORDS 186 RD
   R

   File PASSWORDS has the read-protect attribute, R, and the random file characteristic, D.

CHLAT

Command

Change a file's link access attributes

Syntax

CHLAT filename *[sign]* attributes *[...filename [sign] attributes]*

Description

Assigns or removes link access attributes for filename. A file's link access attributes control a link user's access rights to that file.

sign can be + (plus) to add an attribute, or − (minus) to remove an attribute. When you specify an attribute without a + or a − sign, the new attribute replaces any existing attributes.

To remove all link access attributes, enter 0 as an argument.

See the CHATR command for information on setting file access attributes. The difference between file access attributes and link access attributes is that file access attributes apply to all methods of access, including link access. Link access attributes apply only to link access; they have no effect when a file is accessed from within its own directory or through use of a directory specifier.

The LIST command displays link access attributes for a file, by appending them, preceded by a slash (/), to the display of any existing file access attributes.

See the LINK command for information on creating and using link files.

Link Access Attributes

N Prohibits links to this file. (You can create a link to the file from another directory, but the link will not work.)

P Makes this a permanent file. Users linking to this file cannot delete or rename it.

R Read-protects this file. Users linking to this file cannot display the file (but can execute it).

S Permits execution of this file.

W Write-protects this file. Users linking to this file cannot modify the file. (Note that users can delete the file if they use DELETE on their link file. See the UNLINK command for information about removing links.)

0 Removes all removable attributes, except attribute S.

* Retains all existing attributes. Similar to using = to add attributes to existing attributes.

? User-defined attribute.

& User-defined attribute.

CHLAT (continued)

Examples

1. *R*
   LIST ROSETTA.SV ↵
   ROSETTA.SV 331 SD
   R

   The LIST command shows that file ROSETTA.SV has no link access attributes, but has file attribute S (execute) and file characteristic D (random file).

2. *R*
   CHLAT STONE.SV P ↵
   R
   LIST STONE.SV ↵
   STONE.SV 331 SD/P
   R

   The CHLAT command assigns the permanent (P) link access attribute, ensuring that users who link to the file STONE can't delete it.

CLEAR

Command

Set file use count to zero

Syntax

CLEAR *[filename ...]*

Description

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; the use count should be 0 when the file is closed. If a system fails when a file is open, its use count remains nonzero when the system is rebootstrapped, and you cannot delete or rename the file.

Use the LIST/U or LIST/E command to determine a file's use count.

If an abnormal operating system shutdown occurs, you 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 from all other directories that were initialized at the abnormal shutdown.

You also receive the *PARTITION IN USE* message if someone shut off power to the computer on the master directory drive before releasing the master directory. Again, clear all directories that were initialized when power was turned off.

A DOS system gives no *PARTITION IN USE* 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.

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 terminal display.

Examples

1. CLEAR/A/V/D ↵

   .
   .
   .

   Clears use count of all files and devices in the current directory except CLI files; verify filenames cleared.

2. CLEAR/V DP4:DEMPSEY ↵
   CLEARED DP4:DEMPSEY
   R

   Clears and verifies use count of file DEMPSEY, in directory DP4.

CLG

Utility

Compile, load, and execute a FORTRAN IV program

Syntax

CLG filename *[... filename]*

Description

CLG (compile, load, and go) compiles FORTRAN IV programs, assembles them with the Extended Assembler, loads them with the Relocatable Loader, and executes the final .SV file.

The FORTRAN IV compilation and assembly steps are optional. In the argument list, you can include FORTRAN IV source files (filename.FR), assembly language files (filename.SR), or assembled binary files (filename.RB). CLG output includes one or more temporary source files, one or more binary files, and the save file.

In addition to the local switches below, you can append RLDR local switches to each name of a binary file. To create overlays with CLG, enclose the filenames in brackets.

Global Switches

/B Lists only the source (input) program.

/E Suppresses compiler error messages. Assembler error messages are not suppressed.

/M Suppresses load map (symbol table).

/T Makes this a multitask program. (The multitask FORTRAN library, FMT.LB, must be available on disk.)

Local Switches

file/A Assembles and load this file; does not compile.

file/E Directs error messages to this file.

file/L Directs listing output to this file.

file/O Executes RLDR phase only on this file, does not compile or assemble.

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 file, *filename.SR*. Produce binary file, filename.RB. Produce save file *filename.SV* (unless local /S was included in the command line).

For CLG to execute properly, the following files must be in the current directory, or available through links:

CLG.SV

FORT.SV

FIV.SV

ASM.SV

RLDR.SV

RLDR.OL

SYS.LB

FORT.LB

You can produce FORT.LB by merging 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* (DGC No. 093-000053) and the *FORTRAN IV Runtime Library User's Manual* (DGC No. 093-000142).

Examples

1. *R*
   CLG A B C ⤸

   Compiles A.FR (or A), producing A.SR; assemble A.SR into A.RB and deletes A.SR. Does the same with B.SR (or B) and C.SR (or C). Loads A.RB, B.RB, C.RB and FORT.LB to produce A.SV. Executes A.SV.

2. *R*
   CLG/B DOGE @LPT/L ⤸

   Compiles DOGE.FR (or DOGE), lists DOGE.FR on the line printer, and produces DOGE.SR. Assembles DOGE.SR into DOGE.RB and deletes DOGE.SR. Processes binary DOGE and the FORTRAN IV library producing DOGE.SV. Executes DOGE.SV.

3. *R*
   CLG/T JAKE FB/O SR/A $LPT/L ⤸

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

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

COPY *Command*

Copy the contents of a DOS diskette onto another DOS diskette (DOS only)

Syntax

COPY sourcediskette destinationdiskette

sourcediskette specifies the device name of the diskette you want to copy. **destinationdiskette** specifies the device name of the diskette to which the copied files are transferred.

Description

Copies all files and directories from **sourcediskette** to **destinationdiskette**. Before you issue this command, make certain that you have initialized **sourcediskette**.

destinationdiskette must *not* be initialized. During the transfer, all existing files on **destinationdiskette** are destroyed and replaced by the files from **sourcediskette**.

When the copy is complete, **destinationdiskette** has the same system directory and directory structure as **sourcediskette**; the diskettes are virtually identical.

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 (blocks 0 and 1 of the bootstrap utility) is not part of the file structure, it is not copied. If you want to bootstrap from a destination diskette that is lacking a bootstrap, you need to install a root on it.

Global Switch

/L Produces a list of the filenames of each file copied at the line printer (device name $LPT). This switch will send the list to your terminal if you first redirect line printer output to your terminal with the command LINK $LPT $TTO.

Example

1.
```
R
DIR DP0 ↵
R
COPY DP0 DP1 ↵
R
```

The DIR command initializes diskette DP0 and makes it the current directory. The COPY command duplicates DP0 (the source diskette) on DP1 (the destination diskette). Diskette DP1 was never initialized, so you do not have to release it before removing it from its drive.

2. *R*
INIT DP2 ⤶
R
INIT DP3 ⤶
R
COPY DP2 DP3 ⤶
DEVICE ALREADY INITIALIZED: DP3
R
RELEASE DP3 ⤶
R
COPY DP2 DP3 ⤶
R

This example initializes both diskettes — DP2 and DP3 — and attempts to copy DP2 on DP3. DOS cannot COPY to an initialized diskette. Once we release DP3, the COPY command is successful.

CPART *Command*

Create a secondary partition (RDOS and DG/RDOS only)

Syntax

CPART partition_name blockcount

Description

CPART creates a secondary partition, named **partition_name**. The secondary partition is a type of directory that reserves a fixed amount of contiguous disk blocks for storing files and subdirectories.

You supply a name and a **blockcount** (the number of contiguous blocks to be reserved) for the partition on the CPART command line. **blockcount** specifies the maximum amount of disk storage space, in blocks, you want to allocate to the partition. If the **blockcount** you specify is not an integer multiple of 16, the **blockcount** defaults to the next lower multiple. You cannot specify fewer than 48 disk blocks.

RDOS appends a .DR extension (the filename extension for directories) to the name you assign. RDOS assigns the secondary partition the file characteristics CTY, where:

C denotes a contiguous file

T denotes a secondary partition

Y denotes a directory

Secondary partitions must be subsets of a primary partition. If you attempt to create a secondary partition from a directory other than a primary partition, the CPART command does execute, and you receive the error message *DIRECTORY DEPTH EXCEEDED.*

No switches.

Example

```
R
DIR DJ1 ↵
R
CPART SECPART 144 ↵
R
```

The DIR command makes the primary partition DJ1 (where DJ1 is the device name for a diskette) the current directory. The command CPART SECPART 144 creates a secondary partition called SECPART.DR on DJ1, and allocates a fixed length of 144 contiguous disk blocks to it.

When you use SECPART.DR as a directory argument, you do not have to specify its .DR extension:

```
R
DIR SECPART ↵
R
```

This command makes SECPART the current directory.

When you use SECPART.DR as a filename argument, you must specify its extension:

```
R
LIST SECPART.DR ↵
SECPART.DR 73728 CTY
R
```

Note that the LIST command displays the 144 blocks as 73728 bytes. (One disk block equals 512 bytes; 144 disk blocks equal 73728 bytes.) The characters CTY denote the file characteristics for a secondary partition.

CRAND *Command*

Create a random file

Syntax

CRAND filename *[...filename]*

Description

CRAND creates **filename**, a randomly organized file in the current directory, or in the directory specified in a pathname. The created file will have an initial length of 0, no file attributes, and the file characteristic D, which indicates a random file.

To insert information into the file, use a text editor or the CLI command XFER. The length of the file expands as you add data to it.

Note that to execute an executable program (.SV) file, the file must be a random file.

No switches.

Examples

1. *R*
   CRAND NEWFILE ⤸
   R
   LIST ⤸
   NEWFILE 0 D
   R

   Creates the random file, NEWFILE, in the current directory.

2. *R*
   CRAND FILE1 FILE2 DP1:CORP:MEETINGS ⤸
   R
   LIST DP1:CORP:MEETINGS ⤸
   DP1:CORP:MEETINGS 0 D
   R

   Creates FILE1 and FILE2 in the current directory, and creates the file MEETINGS in directory CORP on disk DP1.

CREATE

Command

Create a sequential file (RDOS, DG/RDOS)

Syntax

CREATE filename *[...filename]*

Description

CREATE creates a sequentially organized file in the current directory, or in the specified directory. The file will have an initial length of 0, and is not assigned any file characteristics. No switches

To add information into the file, use a text editor or the CLI command XFER. The length of the file expands as you add data to it.

(Under DOS, CREATE makes a random file.)

Examples

1. *R*
   CREATE SEQFILE ↵
   R
   LIST ↵
   SEQFILE 0
   R

   Creates the sequential file, SEQFILE, in the current directory.

2. *R*
   CREATE FILE3 DP1:PART1:DATAFILE ↵
   R
   LIST FILE3 DP1:PART1:DATAFILE ↵
   FILE3 0
   DP1:PART1:DATAFILE 0
   R

   Creates FILE3 in the current directory, and creates the file DATAFILE in secondary partition PART1 on disk DP1.

CSSORT

Utility

Invoke the CSSORT Sort/Merge Program

Syntax

CSSORT infile outfile key *[...key] [arguments]*

Description

CSSORT invokes the CSSORT Sort/Merge program, which can reorder and combine records in a file, according to your specifications. You can use CSSORT on any random file (characteristic D) or contiguous file (characteristic C), but not on sequential files (no characteristics).

The Sort function takes records from the input file, sorts them according to command line specifications, and creates a sorted output file.

Your input files should be set up so that each line of the file has separate fields, definable by column number (position) and length. You can then sort on each field by specifying column number and length with key arguments on the command line. You can also select individual fields to be recorded in the output file with the /F local switch.

The Merge function combines records from up to six input files, and creates a single merged output file. Use the /M global switch to select the Merge function.

Global Switch

/L Assigns the output file a line-sequential file type.

/M Specifies a Merge operation. (The default operation is Sort.)

/N Skips the terminal display of sort statistics.

/S Assigns the output file a fixed-sequential file type. (Fixed-sequential is the default; you do not have to specify this switch.)

/V Assigns the output file a variable-sequential file type.

Local Switches

name/A Writes Sort/Merge statistics to file name.

name/C Specifies the file, name, containing a user-specified collating sequence.

pos:len/F Specifies the position and length of an input field. pos is a decimal number specifying the position of the first character to be copied to the output record; len is a decimal number specifying the length of the input field.

name/I Informs CSSORT that input file name is an ICOBOL indexed file. (The default file type is fixed-sequential.)

pos:len/K Specifies the position and length of a key. pos specifies the position of the first byte in the input record; len is the byte length of the key in the input record.

name/L Informs CSSORT that input file name is a line sequential file. (The default file type is fixed-sequential.)

n/N Specifies n bytes in the input record. The default is 4096 bytes. This switch is mandatory for fixed-sequential files only, and useful for other file types to speed the sorting process.

name/O Identifies name as the Sort/Merge output file. If file name exists prior to the Sort/Merge operation, CSSORT will terminate with an error message.

name/R Informs CSSORT that input file name is an ICOBOL relative file. (The default file type is fixed-sequential.)

name/S Informs CSSORT that input file name is an fixed-sequential file. This is the default CSSORT file type; you do not have to specify this switch.

name/V Informs CSSORT that input file name is a variable-sequential file. (The default file type is fixed-sequential.)

name/W Specifies file name as a temporary sorting file (or work file). CSSORT creates up to six temporary work files, and deletes them when the sort completes. If you omit this switch, CSSORT names its work files SORTWn.TP, where n is a number from 1 to 6.

Examples

1. *R*
   CSSORT STEST.SI STEST.SO/O 1:20/K ↵
   .
   .
   .
   R

   Invokes the CSSORT Sort function to sort on the field in file STEST.IN that begins in column 1 and has a length of 20 characters. CSSORT writes the sorted output to file STEST.OT. Note that the .SI ("sort input") extension for the input file, and the .OT ("sort output") extension for the output file, are optional extensions.

2. *R*
   CSSORT/M STEST1.SO STEST2.SO MTEST.MO/O ↵
   .
   .
   .
   R

   Invokes the CSSORT Merge function (/M) to merge sorted files STEST1.SO and STEST2.SO into output file MTEST.MO.

DDUMP

Utility

Dump files from the current directory to one or more diskettes, in DDUMP format

Syntax

DDUMP *[devname]*

Description

If you do not specify switches and arguments in the command line, DDUMP begins an interactive dialog requesting you to supply needed arguments. In a DDUMP dialog, DDUMP accepts only uppercase letters.

DDUMP transfers copies of files from the current directory to diskette(s) in DDUMP format, which retains all original file and directory information. When you load the files back to disk (with the DLOAD utility), DLOAD uses the information to reconstruct the original file and directory structure on the disk.

DDUMP supports multivolume diskette dumps. If all of the files you are dumping do not fit on a single diskette, DDUMP prompts you to insert a new diskette into the drive and continues the dump.

To load multivolume diskettes (with DLOAD), you must supply the diskettes in the same order in which they were dumped. You should mark all the diskettes used during a single dump session with appropriate identifiers and dates, and include numbers indicating the order in which they were dumped.

DDUMP overwrites existing data on a diskette if you specify the /I switch or answer Y to the INIT/F part of the dialog. Otherwise, DDUMP uses only the largest area of contiguous unused blocks on the diskette.

You need access to the following files to use the DDUMP utility:

DDUMP.SV

DDUMP.OL

DDUMP.ER

For more information on DDUMP, refer to the *RDOS, DOS, DG/RDOS Backup and Move Utilities* manual.

Global Switches

/A Includes permanent files in the dump.

/C Checks (verifies) dumped files against the originals.

/D Includes directories subordinate to the current directory in the dump.

/F Includes the resolution files of link files. (Must use with the /K (include link files) switch; otherwise this switch is ignored.)

/I Performs an INIT/F on the diskette, which enables DDUMP to overwrite any existing information on the diskette.

/K Includes link files in the dump.

/L Produces a list of the filenames of each file dumped on the line printer (device name $LPT). This switch overrides the /V switch.

/V Displays (verifies) the filenames of each file dumped at the terminal.

Example

```
R
DIR CHARLIE ↵
R
DDUMP/I/D/V DJ1 ↵
  STORY.
  RETURN.DB
  DIDHE.$$
  MTA        <---------      DIRECTORY
    TRAIN.
    NICKEL.SV
  RIDE4EVER.
  BOSTON.RB
R
```

Erases all previously existing data on diskette DJ1 (/I), and dumps all current directory files and subordinate directories (/D) to diskette DJ1. The /V switch displays the filenames of each file dumped at the terminal. An arrow points to the directory MTA; indented filenames are from the MTA directory, which is subordinate to the current directory.

DEB

Utility

Load a program into memory and start executing at the debugger address

Syntax

DEB program_name

Description

DEB lets you debug a program while executing it. A symbolic debugger must have been loaded as part of the program .SV file; you do this with the /D switch in RLDR. DEB transfers control to the debugger from within the .SV file; the debugger then performs a line feed 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.

After successfully debugging your program, you will probably want to correct the source program, and then reassemble and reload it.

For more information on debugging, see the *RDOS/DOS Debugging Utilities* manual.

No switches.

Example

```
R
DEB PROG.SV ⤸
START + 15/006751 ⤸
START + 15$B ⤸
$R
.
.
7BSTART + 15
0... 1... 2... 3...$P
.
.
$V
BREAK
R
```

The DEB command line invokes the debugger for file PROG.SV. The debug commands examine a location, set a breakpoint at START + 15, and start the program executing. (The ESC key in debug commands echoes as a $.) Debugging continues. ESC V creates the breakfile, saving the the current state of the debugged program in file BREAK.SV, and returns to the CLI.

DELETE

Command

Delete a file or directory

Syntax

DELETE filename *[...filename]*

Description

Deletes the files named. You can use pathnames to name files in directories if the directories have been initialized.

To delete a subdirectory or partition, you must first free it with the RELEASE command, then type DELETE directoryname.DR. When you delete a directory, the CLI wipes out the directory and all of the files it contains.

CAUTION: Use the UNLINK command to remove a link file. If you try to delete a link file with DELETE, the link file remains, but its resolution file is deleted (attributes permitting).

The DELETE command does not delete files that have the attribute P (permanent). See the CHATR command for more information.

Global Switches

/C Requests confirmation of each deletion. The system repeats each filename, then waits for you to confirm the request by entering NEW LINE or ↓. To prevent the deletion, press any key other than the command line terminator.

/L Produces a list of the filenames of deleted files on the line printer ($LPT). This overrides the /V switch.

/V Verifies that files are deleted by displaying their filenames at the terminal.

Local Switches

mm-dd-yy/A Deletes only files created on 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.

Any command line arguments that include templates are affected by local switches, though the switches might be attached to other arguments.

DELETE (continued)

Examples

1. *R*
   DELETE/C PROGX.-⤸
   PROGX.: ⤸ *
   PROGX.LS: ⤸ *
   PROGX.SR: n
   PROGX.RB: ⤸ *
   PROGX.SV: n
   R

   The command line instructs the CLI to delete all files having the name PROGX and any extension, and the /C switch allows us to confirm the deletions.

   The CLI displays each filename, one at a time, that matches the template PROGX.-. Press the command line terminator in response to each file you want to delete; the CLI deletes the file and displays an asterisk. Type n in response to each file you want to keep. The CLI does not echo the n, and continues with the command.

2. *R*
   DELETE/V -.LS ⤸
   DELETED A.LS
   DELETED COM.LS
   DELETED MAP.LS
   R

   Deletes all files having an .LS extension, verifying the deletion by listing the filenames.

3. *R*
   DELETE/V -CHART-.- 1-6-84/B ⤸
   DELETED NEWCHART.OU
   DELETED NEWCHART.IN
   DELETED XCHART1.OU
   DELETED XCHART4.OU
   R

   Deletes all files that have the characters CHART in the middle of their names *and* have creation dates earlier than January 6, 1984. The /V switch causes the CLI to display the names of the files deleted.

   R
   DELETE GRAPHICS:FOOD.SR ⤸
   R

   Deletes the file FOOD.SR in directory GRAPHICS. Note that without global switches, the only response to this command is an R prompt.

DIR

Command

Change the current directory

Syntax

DIR directory_name

Description

DIR makes the **directory_name** your new current directory. **directory_name** can be a subdirectory, secondary partition, or primary partition. **directory_name** can be in a pathname.

The current directory is the directory from which you work, and whose files you can access without pathnames. When you specify filename arguments without including directory specifiers, RDOS assumes you are referring to files from your current directory.

If the directory you specify is not already initialized (opened for access), DIR initializes it. (See the INIT command description for more information on directory initialization.) When you change your current directory, the previous current directory remains initialized.

To access a directory with the DIR command, its parent directories, if any, must be initialized. If you use a pathname in an argument to the DIR command, (e.g., DIR PART1:SECPART:SUBDIR), DIR initializes all directories listed in the path. When a directory is already initialized, you can supply its name to the DIR command without using a pathname, from anywhere within the directory structure.

When you start up RDOS, the master directory is your current directory.

You close directories that have been initialized through the DIR command with the RELEASE command. If you release the current directory, RDOS places you in the master directory. (See the RELEASE command description for more information.)

Other commands that are helpful in conjunction with the DIR command are GDIR, which displays the name of the current directory, and LDIR, which displays the name of the previous current directory.

No switches.

DIR (continued)

Examples

1. *R*
   DIR PARSETZ ⤸
   R
   GDIR ⤸
   PARSETZ
   R

   The command DIR PARSETZ makes directory PARSETZ the new current directory. The GDIR command displays the name of the current directory — in this example, GDIR displays PARSETZ.

2. *R*
   DIR DP1:APPLICAT:DBMS ⤸
   R
   GDIR ⤸
   DBMS
   R

   This example makes directory DBMS the new current directory. Directory DBMS resides on diskette DP1 in secondary partition APPLICAT. DIR initializes all directories listed in its directory specifier argument.

3. *R*
   DIR PARSETZ ⤸
   R
   GDIR ⤸
   PARSETZ
   R

   We know that directory PARSETZ is initialized because it was a previous current directory. We can therefore use it as an argument to the DIR command without specifying all the directories in the path from directory DBMS to directory PARSETZ.

DISK *Command*

Display the allocation of disk storage space for the current partition.

Syntax

DISK

Description

Displays the current allocation of disk space, in decimal blocks, for the current partition. One disk block equals 512 (decimal) bytes.

The display reports the allocation of disk blocks for the current partition as follows:

LEFT: fffff USED: uuuuu MAX. CONTIGUOUS: ccccc

where:

fffff is the number of blocks free for use.

uuuuu is the number of blocks in use.

ccccc is the number of blocks forming the largest contiguous unallocated space.

If you add the number of unused blocks to the number of blocks in use, the result is the total amount of space allocated to the partition.

If the current directory is a subdirectory, DISK calculates the space usage for its parent partition. If the current directory is a primary partition, DISK surveys the entire disk for space usage.

When determining space requirements, note that for the master directory, the bootstrap program and the system files require at least 16 blocks. Other directories require some space for their system and map directories (SYS.DR and MAP.DR).

No switches.

Examples

1. *R*
   DIR DJ1 ⤸
   R
   DISK ⤸
   LEFT: 520 USED: 88 MAX. CONTIGUOUS: 414
   R

   This response, from diskette DJ0, indicates that 520 out of a total of 608 blocks are still available for use, and that the largest amount of free contiguous blocks is 414.

2. *R*
   DIR DP0 ⤸
   R
   DISK ⤸
   LEFT: 1499 USED: 23161 MAX. CONTIGUOUS: 910
   R

   Disk DP0 has 1499 free blocks, and 23161 blocks in use, out of a total of 37660 blocks. The largest amount of free contiguous space is 910 blocks.

3. *R*
   DISK ⤸
   LEFT: 1478 USED: 8298 MAX. CONTIGUOUS: 544
   R

   This response indicates that 1478 blocks from the original 9776 of the disk are still available for use, with 544 contiguous blocks available.

DLOAD

Utility

Load files, previously dumped with the DDUMP utility, from diskettes to the current directory

Syntax

DLOAD *[devname]*

Description

If you do not specify any switches or arguments on the command line, DLOAD begins an interactive dialog requesting you to supply needed specifications.

DLOAD restores copies of files from the diskette(s) in *devname* (created with the DDUMP utility) to the current directory. DLOAD uses the dump information written by DDUMP to reconstruct the original file and directory structure in the current directory.

Be certain that your current directory can accommodate the appropriate levels of subdirectories to be loaded. DLOAD restores all files from the diskette(s) in *devname*, except for files that have the same names as files in the current directory.

If your DDUMP set uses multiple diskettes, DLOAD prompts you to insert each diskette into the drive, one at a time and in the order in which they were dumped, until all the files are loaded back.

In a DLOAD dialog, DLOAD accepts only uppercase letters.

The files you need to use the DLOAD utility are:

DLOAD.SV

DLOAD.OL

DDUMP.ER (This file contains the messages for both the DDUMP and the DLOAD utilities.)

Global Switches

/F Lists at the terminal the files on the diskette to be loaded, without performing the load.

/K Includes link files in the load.

/L Produces a list of the filenames of each file loaded on the line printer (device name $LPT). This switch overrides the /V switch.

/V Displays (verifies) the filenames of each file loaded at the terminal.

Example

```
R
DIR NEWCHARLIE ↵
R
INIT DJ1 ↵
R
DLOAD/V DJ1 ↵
  STORY.
  RETURN.DB
  DIDHE.$$
  MTA        <----------        DIRECTORY
    TRAIN.
    NICKEL.SV
  RIDE4EVER.
  BOSTON.RB
R
```

DLOAD loads all files previously dumped (with DDUMP — see the DDUMP command example) from diskette DJ1 into the current directory. The /V switch displays the names of all files and directories loaded. An arrow points to the MTA directory.

DO

Utility

Execute CLI macro (.MC) files, replacing dummy argument variables with specified arguments

Syntax

DO macroname *[argument ...]*

Description

The DO utility executes CLI macro (.MC) files, replacing dummy argument variables within the file with the arguments you specify on the command line.

Macro files, also known as .MC files, are executable files that contain series of complete CLI command lines. You create macro files with a text editor, making sure that the filename includes a .MC extension. You can execute all of the commands contained in a .MC file by entering the filename of the .MC file as a CLI command.

The DO utility adds some versatility to macro files, through its ability to handle dummy argument variables.

No switches.

Dummy Argument Variables

A dummy argument is a user variable having the form %n%, where n is a number (n cannot be a 0). The DO utility can handle up to 512 dummy arguments in a single macro file.

You might use a dummy argument in a command line, as follows:

```
PRINT %1%
```

If you supplied a filename argument such as THISFILE, the command line would expand to read

```
PRINT THISFILE
```

Supplying the Arguments on the DO Command Line

DO uses the first argument you specify on the command line to expand dummy argument %1%, the second argument on the command line corresponds to dummy argument %2%, and so on. If a particular dummy argument appears more than once in the macro file, DO replaces the dummy argument with its corresponding actual argument each time it appears throughout the macro file.

If you specify fewer arguments on the command line than exist in the macro file, DO replaces the left over dummy arguments in the macro file with nulls.

You can specify a null on the command line by using the number sign, #, as an argument.

For example, the following line exists in macro file GEORGE.MC:

```
PRINT %1% %2% %3%
```

The DO command line DO GEORGE FILE1 FILE2 FILE3 causes the line to expand to:

```
PRINT FILE1 FILE2 FILE3
```

The DO command line DO GEORGE # FILE2 causes the line to expand to:

```
PRINT FILE2
```

Examples

1. *R*
   TYPE CLOSEDIR.MC ↵
   MESSAGE CLOSING DIRECTORY %1% ...
   DIR %1%
   DELETE/V -.BU
   MESSAGE PRINTING DIRECTORY %1% LISTING ON THE PRINTER
   LIST/S/A/E/L
   RELEASE %1%
   R

   The example displays the contents of macro file CLOSEDIR.MC, which contains a dummy argument, %1%, that allows the CLOSEDIR macro to work for any directory you specify.

2. The next example executes the macro file with DO, supplying an argument for dummy argument %1%.

   R
   DO CLOSEDIR FROGS ↵
   CLOSING DIRECTORY FROGS ...
   DELETED TEST1.BU
   DELETED FOO.BU
   PRINTING DIRECTORY FROGS LISTING ON THE PRINTER
   R

   The DO command line executes macro file CLOSEDIR.MC, supplying one argument, FROGS, to replace any %1% dummy arguments that exist in the macro file.

DUMP

Command

Store one or more disk files from the current directory to a dump format file on another directory or device

Syntax

DUMP *[[destination:]dumpfilename] [...filename] [argument]*

Description

destination specifies the name of the directory or device (disk, diskette, magnetic tape, or paper tape) to which you are transferring the files. This directory or device must be initialized. If you omit this part of the argument, RDOS creates the dump file in your current directory, which is probably not what you intend.

dumpfilename is the name of the file that will contain the dumped files. You need to use this name again to load the files back (with the LOAD command). When dumping to disk, DUMP creates this file. When dumping to tape, give a tape file number for this argument.

filename(s) specifies the files to be dumped. If you do not specify any filename arguments, DUMP transfers all files and directories subordinate to the current directory.

You can use templates when referring to files in the current directory.

DUMP creates a dump file (*dumpfilename*) on the named destination to contain the files dumped from the current directory.

The dump file contains file information and file contents for each file dumped, in a format (*dump format*) that stores the files compactly on the destination. When you load the dump file back to disk (with the LOAD command), LOAD uses the dump information to reconstruct the original files and directory structures on disk.

Use DUMP to create backup copies of files and directories, for storage rather than for interactive use. If you want to copy files and directories to another disk or directory and use them just as you would in your current directory, use the MOVE command.

If you omit all switches and arguments in the command line, DUMP dumps all nonpermanent files (files that do not have the attribute P) and directories subordinate to the current directory.

So if the current directory is a primary partition or diskette, all nonpermanent files are dumped; if the current directory is a secondary partition, the secondary partition and all its subdirectories and files are dumped.

If you specify filename arguments, only those files are dumped. To dump directories subordinate to the current directory (and their contents), specify their filenames (include the .DR extension). Note that you cannot use a primary partition name such as DJ0 as a valid filename argument.

When you use the /V or /L switches, DUMP displays the names of dumped files, each on a separate line, as follows: files from the current directory are indented two spaces, directories are preceded by an asterisk, and files from a subordinate directory are indented four spaces, to help you determine exactly which files were dumped.

A note to DOS users: If the current directory is a write-protected diskette, all files in the dumpfile will receive the attributes APW.

DUMP's complementary command is LOAD. If you dump files using DDUMP (for disks) or FDUMP (for tapes), which are faster but less versatile versions of DUMP, you must load the files with their complementary utilities, DLOAD or FLOAD. The advantage of the DUMP command over these faster utilities is that it allows you to select specific files for dumping, using a number of criteria such as filename extension and creation dates. DDUMP and FDUMP dump only entire directories.

Global Switches

/A Includes permanent (attribute P) files in the dump.

/K Excludes link files.

/L Produces a list of the dumped files on the line printer (device name $LPT). (This switch overrides the /V switch.)

/S Dumps a file on segments of paper tape. The file is punched in segments of up to 20 Kbytes each. Each tape segment is headed by a unique segment number, which enables the system to verify that the tapes will LOAD in proper sequence.

/V Displays (verifies) the filenames of each file dumped at the terminal.

/W Includes the resolution files for any link files dumped, provided that all directories in the path to the resolution file are currently initialized.

Local Switches

mm-dd-yy/A Dumps only files created on this date or after. Arguments mm (month) and dd (day) can be one or two digits.

mm-dd-yy/B Dumps only files created before this date. Arguments mm (month) and dd (day) can be one or two digits.

filename/N Excludes files that match filename from the transfer. filename may include templates.

oldname/S newname
Assigns newname to file oldname when it is transferred, but the file retains its original name (oldname) in the current directory.

Local arguments /A, /B, and /N do not affect filename arguments that are specified explicitly (i.e., named without the use of templates) on the command line.

DUMP (continued)

Examples

1. *R*
   DIR DP0F ↵
   R
   INIT MT0 ↵
   R
   DUMP/A/L MT0:0 7-12-84/A ↵
   R
   RELEASE MT0 ↵
   R

   Dumps all directories and files, including permanent (attribute P) files (/A), from current directory DP0F that were created on or after July 12, 1984, to file 0 of the tape on drive MT0. This tape now provides a backup for these files. The line printer produces a listing of the dumped filenames (/L).

2. *R*
   DUMP/V DP1:SAVEFILES -.SV ↵
   RED.SV
   BLUE.SV
   .
   .
   CHARTREUSE.SV
   R

   Dumps to file SAVEFILES on disk DP1 all files in the current directory with a .SV extension, and lists the filename of each file dumped at the terminal (/V).

3. *R*
   DUMP/A MT0:2 APRIL/S APRIL.BU ↵
   R

   Dumps permanent file APRIL to file 2 of the tape on MT0. Names the dumped copy of APRIL APRIL.BU (which allows you to load it into the same directory as APRIL, if necessary).

4. *R*
   DUMP/A/L DP4:84JUN22.BU -.RB/N 6-22-84/A ↵
   R

   Creates dumpfile 84JUN22.BU (named for a date) on disk DP4, then dumps all files (except .RB files) created on or after June 22, 1984 to file 84JUN22.BU on disk DP4. /L lists, on the line printer, the filenames of each file and directory dumped.

EDIT

Utility

Invoke the EDIT text editor utility

Syntax

EDIT *[filename]*

Description

The EDIT utility creates and modifies text files. Use EDIT without a filename argument to create a new text file. When you create a file with EDIT, the file has random block organization. The command EDIT filename lets you edit an existing file.

When you invoke EDIT, the utility generates an asterisk (*) prompt, and you type EDIT commands in response to this prompt. EDIT commands can create a new file; read in an existing file for editing; add, delete, and change information in the file; and store the modified file on disk.

To end an editing session and return to the CLI, type H and press the ESC key twice.

To abort an editing session, a CTRL-C CTRL-A will return the EDIT prompt (*), and a CTRL-C CTRL-B will return you to the CLI.

EDIT commands are documented in the *RDOS/DOS Text Editor* (DGC 069-400016), and an introduction to the EDIT utility appears in the "SPEED and EDIT Text Editors" chapter of the *Introduction to RDOS* (DGC 069-400011).

No switches.

Example

```
R
EDIT ↓
*                    The * indicates EDIT is ready to accept commands.
.
.                    You work on the file using EDIT commands.
.
*H ESC ESC           Typing H and pressing the ESC key twice terminates EDIT.
R
```

ENDLOG *Command*

Close the log file opened by LOG

Syntax

ENDLOG *[password]*

ENDLOG orders the system to stop output to the log file previously opened with a LOG command. You must close a log file before you can examine or delete it.

To print or delete the log file after closing it, use its full name as an argument. The LOG command names the background log file LOG.CM, and the foreground log file FLOG.CM. If you included a *password* argument with the LOG command to open the log file, you must include the same *password* argument with ENDLOG to close the log file.

This command, ENDLOG *[password]*, appears as a last entry in the log file.

No switches.

Example

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

The password GSTONE is required with ENDLOG since it was used when the log file was opened with LOG/H.

ENPAT

Utility

Create a patch file

Syntax

ENPAT patchfile

Description

The ENPAT utility lets you create patches for .SV and .OL files; a patch is a one-word change to a program or overlay file. Patches are often used to update operating system files. Once you create a patchfile with ENPAT, you install the patches into the program or overlay file via the PATCH utility.

The ENPAT command creates **patchfile**, or opens it for appending, if it already exists. ENPAT then asks five questions about each patch, accepts valid answers, and places them in the **patchfile**. Later you install the **patchfile** with the PATCH utility.

A patch for a program (.SV) file can contain symbols (provided that a load map is available to the PATCH program), octal numbers, or expressions including a symbol, an operator, and an octal number. The most common operators are

+ (addition)
− (subtraction)
@ (indirection)

The PATCH program cannot 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 and PATCH are explained in greater detail in the *RDOS/DOS Debugging Utilities* (DGC No. 069-400020).

ENPAT asks the following questions for each one-word patch. If you give an invalid response, it returns an error message and repeats the question.

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

Answer 0 if the patches in this file will be installed in a .SV file, answer 1 for a .OL file. Next, it asks:

PATCH LOCATION?

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

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:

NEW CONTENTS?

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

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.)

ENPAT (continued)

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 enter multiple patches that concern one condition.

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

EXIT (0=NO 1=YES)

Answer 0 ↵ to return to question 1, and enter another patch; anwer 1 ↵ to create the patch file and return to the CLI.

Example

Assume that you want to insert the patches listed below in patch 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

```
R
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)? ... ↵
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+16 ↵
CONDITIONAL: ↵
EXIT (0=NO 1=YES) 1 ↵
R
```

The PATCH utility example in this chapter continues this example to show installation of the patches in patchfile IPB.

EQUIV *Command*

Assign a temporary name to a disk or tape drive (RDOS)

Syntax

EQUIV newname oldname

Description

EQUIV assigns a temporary name (newname) to a disk, magnetic tape unit, or cassette. newname replaces the oldname until the device is released (with RELEASE).

Properly applied, EQUIV enables you to write device-independent programs. You can write a generic device specifier into your programs, and use EQUIV at runtime to match the generic specifier to the actual device. (See the example below.)

You must use EQUIV to rename a device *before* you initialize it with INIT. The new name then exists only until you release the device. After release, all devices revert to their original specifiers. You cannot use EQUIV on a secondary partition, subdirectory, or the master device.

Global Switch

/P Displays a message to mount a tape or disk, and pauses for user intervention.

Example

Your program refers to all magnetic tape files as TAPEDECK. This gives your program device independence at runtime. Before running the program, you issue the commands:

```
R
EQUIV/P TAPEDECK MT0 ↵
MOUNT TAPEDECK ON UNIT MT0, STRIKE ANY KEY
R
```

This command temporarily changes the name of tape drive MT0 to TAPEDECK. The /P global switch displays a request to the user to mount a tape and strike a key on the keyboard when ready.

```
R
INIT MT0 ↵
FILE DOES NOT EXIST: MT0
R
```

RDOS no longer recognizes tape drive MT0; you must use your new name for all tape operations until after you release the device.

```
R
INIT TAPEDECK ↵
R
LOAD/N TAPEDECK:1 ↵
MASM2
MASM4.LS
R
```

The INIT TAPEDECK command initializes the tape on drive MT0 (TAPEDECK). The LOAD command transfers the files from the second file (file 1) on tape MT0 (TAPEDECK) to disk.

EQUIV (continued)

R
EQUIV MT0 TAPEDECK ⤸
DIRECTORY IN USE
R

You cannot use EQUIV again for the same device until you release the device.

R
RELEASE TAPEDECK ⤸
R

The RELEASE command releases tape drive TAPEDECK, and the tape drive name reverts back to MT0.

R
INIT MT0 ⤸
R

EXFG *Command*

Execute a program in foreground memory (RDOS and DG/RDOS only)

Syntax

```
EXFG programname
```

Description

EXFG executes a program in foreground memory.

When there is no memory separation, or when you want to use the background, you execute programs by typing in the name of the program. You execute programs in the foreground the same way, except you use EXFG to specify that the execution take place within foreground memory boundaries.

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.

Chapter 6 explains memory separation between background and foreground, and explains which system resources are shared between the two grounds.

How you use EXFG depends on whether your system is mapped or unmapped. RDOS offers several system calls for handling foreground and background programs.

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 (such as assemblies or compilations) in the foreground; or you can execute interactive programs like the CLI, and access them through the foreground console(s).

A foreground CLI has different names for system 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.

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 EDIT text editor (EDIT.RB) and its multiterminal counterpart (MEDIT.RB). If you choose, you can use RLDR to configure one of these with boundary addresses, and edit in the foreground (on a second console).

Before you execute any user program in the foreground, load and run it in the background, and check its ZREL and NREL requirements with SEDIT, or OEDIT, or the load map. 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).

EXFG (continued)

If you attempt to execute in foreground a program that would require space needed by the CLI, RDOS will reject the command, and display the following 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 higher priority.)

Examples

1. *R*
   EXFG DP1:CRESS ⤸
   R

   Executes program CRESS, on primary partition DP1, in foreground memory. In a mapped system, the foreground as set by SMEM must accommodate CRESS; if CRESS won't fit, the system returns the error message:

   INSUFFICIENT MEMORY TO EXECUTE PROGRAM

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

   INSUFFICIENT MEMORY TO EXECUTE PROGRAM

   In either system, if CRESS executes, the CLI remains active in the background; the user could then try to execute a different program in the background. Again, if the background does not have enough space to execute the program, RDOS displays the *INSUFFICIENT MEMORY* message.

2. The following example shows how you might run a foreground program in an unmapped system.

   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:

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

   Configures RECASTF.SV to run in the high section of user ZREL and NREL memory.

   You now issue EXFG RECASTF to execute RECASTF in the foreground. If it executes, you can execute another program in the background.

3. The next example demonstrates running a foreground program on a mapped system.

```
R
GMEM ⤸
BG: 52 FG: 0
R
```

Checks the memory allotment for each ground: 52 Kbytes for background, none for foreground.

```
R
SMEM 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
```

Assembles a complex program in the foreground.

```
R
NEWFILE ⤸
.
.
.
.
```

Execute NEWFILE in the background.

FCOPY

Utility

Duplicate a diskette or copy a file (DG/RDOS only)

Syntax

FCOPY *[source destination]*

Description

If you omit arguments on the command line, FCOPY uses a menu to prompt you for the arguments.

source specifies the device name of the diskette or the name of the file that you want to duplicate. This argument can include a pathname to specify a file outside the current directory, but cannot include templates.

destination specifies the device name of the diskette or the filename on which FCOPY creates the duplicate.

FCOPY can duplicate a diskette to another diskette using only one disk drive. The destination diskette must be hardware formatted, and must not contain any bad blocks. FCOPY stops with an error message if it encounters a bad block.

FCOPY can duplicate any diskette that has a Data General Corporation hardware format.

FCOPY can also copy individual files to another diskette if your system has only one diskette drive. Otherwise, the MOVE command is preferable for moving particular files because it offers more flexibility, such as the use of templates.

See *Using DG/RDOS on DESKTOP GENERATION™ Systems* for documentation of FCOPY.

Global Switches

/C Specifies a file copy operation.

/D Specifies a diskette duplication

/V Verifies the duplicate or copy by checking it against the original.

Example

```
R
FCOPY ↵
DG/RDOS Diskette Transfer Utility REV n.nn
Command:              STRIKE  FUNCTION-KEY-1  FOR  OPTIONS
-------------------------------------------------------------------------------

1.  Copy a file
2.  Duplicate a Diskette
3.  Exit from program

(1) Continue (2) Restart (3) Return to Main Menu
        Select desired option: 1 ↵

2 ↵

Enter source diskette name: DJ0 ↵
Enter destination diskette name: DJ0 ↵

Do you want verification of the duplicate? (Y/N)N Y ↵

Writing time will be longer due to the verification option.

Select CONTINUE if the pathnames above are correct.

(1) Continue (2) Restart (3) Return to Main Menu
        Select desired operation: 1 ↵

Insert SOURCE diskette: DJ0 ↵

Elapsed time min:sec
Reading from source file ...

Insert DESTINATION diskette: DJ0 ↵
Elapsed time min:sec
Writing to destination file ....
.
.
.
Transfer and verification completed

(1) Continue (2) Restart (3) Return to Main Menu
```

Select 2 to duplicate a diskette. FCOPY asks for source diskette, and reads as much information into memory as it can fit. FCOPY asks for the destination diskette and writes from memory all data from the source. FCOPY repeats this stage until it completes the duplication.

FDUMP *Utility*

Dump files from the current directory to one or more magnetic tapes

Syntax

FDUMP MTn:f

Description

FDUMP transfers copies of files from the current directory and its subordinate directories to magnetic tape. MTn:f specifies the device name of the magnetic tape to which you are dumping files, where n is the unit number of the tape drive, and f is the number of the tape file (usually file 0).

FDUMP does not copy files that have the read-protect attribute R. To load FDUMP files back from magnetic tape, use the FLOAD utility.

FDUMP supports multivolume tape dumps. If all of the files you are dumping do not fit on a single tape, FDUMP prompts you to mount a new tape and continues the dump.

You must load these tapes (with FLOAD) in the same order in which they were dumped. You should mark all the tapes used during a single dump session with appropriate identifiers and dates, and include numbers indicating the order in which they were dumped.

If all of the files you dump fit on part of a single tape, you can use the same tape to include additional FDUMP tape files. This is called *stacking*. Note that FDUMP writes three end-of-file marks after each tapefile it creates. This means that to stack dumps on a single tape you must dump to tape file 0, and then tape file 3, 6, 9, 12, and so on. Do not mix FDUMP and DUMP files on one tape.

FDUMP initializes the specified tape drive and releases it when it has finished dumping to that tape (thus rewinding the tape). If you are using alternate tape drives in a multivolume tape dump, be sure to remove each tape when it is rewound to prevent FDUMP from overwriting the tape.

The FDUMP utility uses less tape and executes faster than the DUMP command.

Global Switches

/L Produces a list of the filenames of each file dumped on the line printer (device name $LPT). This switch overrides the /V switch.

/V Displays (verifies) the filenames of each file dumped at the terminal.

Local Switches

MTn:0/A Specifies a new tape on alternate tape drive n. When the first tape is filled, the dump continues on file 0 of this alternate tape.

filename/L Produces a list of the filenames of each file dumped in disk file filename. This overrides the global /L and /V switches.

Example

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

Dumps all files and directories subordinate to the current directory (disk DZ0) to tape MT0, file 0. If FDUMP fills tape MT0 before it has finished dumping all of the files, it continues the dump on tape MT1, file 0.

FGND *Command*

Disclose whether or not a foreground program is running

Syntax

FGND

Description

FGND allows you to check the status of the foreground. It returns one of two messages, depending on whether or not a program is currently running in the foreground.

No switches.

Example

R
FGND ⤸
NO FOREGROUND PROGRAM RUNNING
R
EXFG MYPROG ⤸
R
FGND ⤸
FOREGROUND PROGRAM RUNNING
R

FGND tells us no programming running in the foreground. After we use the EXFG command to execute a program in the foreground, FGND tells us a program is running.

FILCOM

Command

Compare the contents of two files

Syntax

FILCOM filename$_1$ filename$_2$ *[outputfile/L]*

Description

FILCOM compares two files, word by word, and displays dissimilar word pairs, in octal, at the terminal. To send output elsewhere, use the /L switch. File organizations of the two files can differ; e.g., you can compare a random and contiguous file.

Local Switch

name/L Sends output to *outputfile*

Example

R
FILCOM YIN YANG $LPT/L ↵
R

Compares files YIN and YANG word-by-word. Prints 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 that YIN (note the dashes for these locations in YIN). If either YIN or YANG were a null file, FILCOM would print dashes for the null file and the entire contents of the other file.

FLOAD

Utility

Load files previously dumped with the FDUMP utility from one or more magnetic tapes to the current directory

Syntax

FLOAD MTn:f

Description

MTn:f specifies the device name of the magnetic tape from which files are to be moved to the current directory, where n is the unit number of the tape drive and f is the number of the tape file.

FLOAD transfers copies of files from a dump tape (created with the FDUMP utility) to the current directory. FLOAD restores the original file and directory structure to disk.

Be certain that your current directory can accommodate the appropriate levels of subdirectories loaded. FLOAD restores all files from the tape, except for files that have the same names as files existing in the current directory.

FLOAD automatically initializes the specified tape drive, but does not release it.

If the files you dumped with FDUMP required more than one tape, FLOAD prompts you to mount each tape used onto a tape drive, one at a time and in the order in which they were dumped, until all the files are loaded back.

If you stacked more than one FDUMP tape file on a single tape, you must load them back using the same tapefile numbers (0, 3, 6, etc). See the FLOAD description for more information about stacking.

Global Switches

/L Produces a list of the filenames of each file loaded on the line printer (device name $LPT). This switch overrides the /V switch.

/V Displays (verifies) at the terminal the filenames of each file loaded.

/N Lists at the terminal the filenames on the tape, without performing a load operation. When used with the /L switch, the list is produced at the line printer.

Local Switches

MTn:0/A Continues the load from tape file 0 on alternate tape drive n when the first tape is loaded.

filename/L Produces a list of the filenames of each file loaded in disk file filename. This overrides the /L global switch.

Example

```
R
DIR DZ0 ↵
R
FLOAD/L MT0:0 MT1:0/A ↵
R
```

Loads the contents of tape MT0, file 0, and tape MT1, file 0, to the current directory (disk DZ0). FLOAD restores the entire file and directory structure previously dumped with the FDUMP command. (See the FDUMP command example.)

FORT

Utility

Compile a FORTRAN IV source file

Syntax

FORT filename *[... filename]*

Description

FORT compiles filename, a FORTRAN IV source file. For output, you can specify a binary file, an intermediate source file, a listing file, or combination of all three. On input, the command searches for inputfilename.FR; if not found, it searches for inputfilename.

If you omit switches, FORT 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 default.

To use FORT, the following files must be available in the current directory or available via links:

FORT.SV (the FORTRAN IV interface)
FIV.SV (the compiler)
ASM.SV (the extended assembler)

Global Switches

/A Suppresses assembly (the intermediate source file is deleted unless you use global /S). Useful for checking compile errors.

/B Lists compiler source program input only.

/E Suppresses error messages from compiler. (Assembler error messages are not suppressed.)

/F Gives equivalent FORTRAN variable names and statement numbers to symbols acceptable to the assembler. (By default, FORT provides assembler translation of all code except variable names and statement numbers.) Include /U to inform the Debugger of the /F names.

/L Produces listing file (*filename.LS*).

/N Supresses relocatable binary file. Useful for checking assembly errors.

/P Compiles no more than 72 columns per line (as in punched card images).

/S Saves the intermediate source file.

/U Appends user symbols during the assembly phase (must be used with /F).

/X Compiles statements that contain X in column 1.

Local Switches

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

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

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

name/S Directs intermediate source output to file name.

FORT (continued)

Examples

1. *R*
   FORT/L MAIN ↵

   Compiles and assembles FORTRAN IV program MAIN or MAIN.FR to produce binary file MAIN.RB; sends both a compiler and an assembler listing to file MAIN.LS.

2. *R*
   FORT/F/U DP1:TABLE $LPT/L ↵

   Compiles and assembles FORTRAN IV file TABLE.FR (or TABLE) on DP1 and produces binary file TABLE.RB. Writes compiler and assembler listings to the line printer ($LPT/L), and appends user symbols during assembly (/U).

FORTRAN

Utility

Compile a FORTRAN 5 file (RDOS only)

Syntax

FORTRAN filename *[...filename]*

Description

Performs a FORTRAN 5 compilation. Output may be a binary file, a listing file, or both. See the *FORTRAN 5 Reference Manual* (DGC No. 093-000085).

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

Global Switches

/B Generates a brief listing (compiler source program input).

/C Checks syntax only.

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

/I Does not list source from INCLUDE files.

/K Does not delete compiler temporary files after compilation.

/L Produces a listing file named *filename.LS*.

/N Compiles, but does 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 Generates code for subscript checking.

/X Compiles statements with X in column 1.

Local Switches

name/B Gives 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

1. *R*
   FORTRAN/L STRANG ↵

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

2. *R*
   FORTRAN FMARK FLIST/L ↵

   Compiles FORTRAN 5 source program FMARK or FMARK.FR and outputs source and compiler listings to disk file FLIST.

FPRINT *Command*

Display a disk file in the specified format

Syntax

FPRINT filename

Description

Prints a disk file in either byte, decimal, hexadecimal, or octal format, with printable ASCII characters on the right side. Output goes to your terminal unless you use FPRINT/L.

Any nonprinting characters are reported as periods (.). If you omit switches, the locations are printed in octal at the terminal. You can specify a first and last location with local switches.

FPRINT offsets locations by 16 (octal) unless you include the /Z switch.

Global Switches

/B Prints in byte format.

/D Prints in decimal.

/H Prints in hexadecimal.

/L Outputs to the line printer.

/O Prints in octal (default).

/Z Prints locations starting at zero.

Local Switches

n/F Starts at location n (octal).

name/L Sends output to file named (overrides global /L).

n/T Stops at location n (octal).

Examples

1. *R*
   FPRINT/L TE1 ↵
   R

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

2. *R*
   FPRINT/B/L MYFILE 2000/F 3500/T ↵
   R

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

GDIR

Display the name of the current directory

Syntax

GDIR

Description

GDIR gets and displays the name of the current directory. The value that the CLI stores and fetches for the GDIR command is the same as that of the %GDIR% CLI variable.

See the DIR command description for information on changing your current directory.

No switches

Examples

1. *R*
   GDIR ↵
   MANHATTAN
   R

   MANHATTAN is the current directory.

2. *R*
   GDIR ↵
   SKYSCRAPER
   R
   RELEASE SKYSCRAPER ↵
   R

   In this example, we determine the name of the current directory with GDIR so that we can release it with the RELEASE command. An equivalent command would be

   RELEASE %GDIR%

   where %GDIR% is the CLI variable that contains the same value the GDIR command displays.

GMEM *Command*

Display background and foreground memory allocations (Mapped RDOS only)

Syntax

GMEM

Description

Displays the current memory allocation to background and foreground memory areas. The size is given in memory pages, where one page equals 1024 words (or 2048 bytes).

GMEM works only on systems with mapped addressing.

No switches

Examples

1. *R*
   GMEM ⤸
   BG: 80 FG: 104
   R

   80 pages of memory are available to the background memory area (BG:); 104 pages of memory are available to the foreground (FG:).

2. *R*
   GMEM ⤸
   BG: 184 FG: 0
   R

   No memory is available to the foreground memory area; all available user memory is currently allocated to the background. This means that no foreground currently exists. Before you can run a program in the foreground, you must allocate memory to the foreground with the SMEM command.

GSYS

Command

Display the name of the current operating system

Syntax

GSYS

Description

Ouptuts on the terminal display the filename of the operating system that is currently running.

No switches.

Examples

1. *R*
   GSYS ⤸
   SYS
   R

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

2. *R*
   XFER/A $TTI OPSYS.MC ⤸
   MESSAGE THE CURRENT OPERATING SYSTEM IS %GSYS% ⤸
   ^Z
   R
   OPSYS ⤸
   THE CURRENT OPERATING SYSTEM IS SYS64K
   R

   In this example, we create a macro that contains the variable %GSYS%. Upon invoking the macro, we see that the name of the currently running operating system is SYS64K.

GTOD

Command

Display the time and date

Syntax

GTOD

Description

Displays the current system time and date. No switches.

Example

R
GTOD ↓
10/17/84 21:24:20
R

The message indicates that the time is 9:24:20 p.m., and the date is October 17, 1984.

ICOBOL

Utility

Compile an ICOBOL source program

Syntax

ICOBOL source_file *[list_file] [error_file]*

Description

The Interactive COBOL Compiler is a single-pass compiler that prepares a COBOL source program for processing by the runtime system. The compiler recognizes card or CRT line formats, and either indexed or sequential file organization. ICOBOL compilation produces optional listing files as well as two object files: the Data Division in *source_file.DD*, and the Procedure Division in *source_file.PD*.

For more documentation, see the *Interactive COBOL User's Guide (RDOS, DG/RDOS)* (DGC No. 069-705014).

Global Switches

/C Card format source; the compiler ignores all characters beyond column 72 of a card-format source line. Maximum line length for CRT format is 132 characters, an RDOS restriction. Without this switch, the default is CRT format.

/D Add a symbol table to the .DD object program file for runtime debugging. The compiler ignores this switch if you suppress the object program with the /N switch.

/E Suppress error messages from the listing file. This switch is not required if you direct error messages to an error file with the local /E switch.

/L Use source_file.LS as the compilation listing file. This switch is overriden with the local /L switch.

/N Do not produce an object program. Without this switch, the compiler produces two object files, *source_file.DD* and *source_file.PD*. This switch can't be used with the /U switch.

/S Append the debugging symbol table, compiler statistics, and additional data to the listing file.

/U Append a decompilation of the object program to the listing file. This switch can't be used with the /N switch.

/X Append a cross-reference table to the listing file.

Local Switches

error_file/E Sends the error listing to error_file.

source_file/I Indicates that the source file has indexed organization, as produced by IC/EDIT. Enter the source filename without its extension. Use this switch with the global /C switch for card-format source code.

list_file/L Sends compilation listing to list_file.

file/P Purges the contents of the existing list_file or error_file before writing new output. Can be used only with local /E or local /L. Without this switch, the compiler appends output to the existing file.

ICOBOL (continued)

Example

R
ICOBOL/X/S INDEPEN/I INDLIST/L INDERR/E ↵

With this command line, /X and /S direct that a cross-reference table, debugging symbol table, and compiler statistics are added to the listing file. The listing file is given the name INDLIST and the error file INDERR, while the object files will be INDEPEN.DD and INDEPEN.PD. The /I local switch tells ICOBOL that the source file INDEPEN has an indexed organization.

ICX

Utility

Invoke the Interactive COBOL Runtime Environment

Syntax

ICX *[program_name/x]*

Description

The ICX command invokes the Interactive COBOL runtime system. The runtime system can execute in CLI mode, or in log-on mode.

With CLI mode, you must specify directly to the CLI the name of the Interactive COBOL program to run (argument *program_name*). Include the /C or /D global switch to specify CLI mode.

With log-on mode, you run Interactive COBOL programs through the log-on menu.

Global Switches

/B Enables RDOS or DG/RDOS interrupts (CLI mode). In log-on mode, enables interrupts from the master terminal.

/C CLI mode; does not invoke the debugger. You must specify /C or /D to invoke Interactive COBOL in CLI mode.

/D CLI mode; invokes the debugger. You must specify /C or /D to invoke Interactive COBOL in CLI mode.

/I Enables interrupts from Interactive COBOL programs (CLI mode). In log-on mode, allows interrupts at all terminals other than the master terminal.

/P Disables PASS (log-on mode only).

/S Enables spooling (log-on mode only).

Local Switches

program_name/x /x is a logical switch defined in an Interactive COBOL program. name is the name of the program to be executed. For CLI mode only.

n/F Specifies the maximum number of Interactive COBOL ISAM files that may be open concurrently. n must be a value from 4 to 64. The default maximum is 16.

n/M Assigns the master console to the QTY line specified, where n is the number of the QTY line in the range 0 to 63. Log-on mode only.

n/N Specifies the maximum number of additional channels that can be opened. n must be a value from 0 to 237. The default maximum is 16.

n/S Specifies the maximum program size in Kilobytes, where n must be an odd number from 3 to 31. The default maximum is 27 Kbytes.

n/T For CLI mode, n/T specifies the number of tasks. If you specify this argument, you must specify one terminal, or ICX returns an error. For log-on mode, n/T specifies the maximum number of terminals and detached jobs that can run during runtime execution, where n is a value from 1 to 33. The default is 1 active terminal.

ICX (continued)

Examples

1. *R*
   ICX/C NEWPROG ⤸

   Invokes the Interactive COBOL runtime environment in CLI mode.

2. *R*
   ICX ⤸

   Invokes the Interactive COBOL runtime environment in log-on mode.

IMOVE

Utility

Dump and load files between disk, diskette, or magnetic tape (DG/RDOS only)

Syntax

IMOVE devname *[filename ...]*

Description

devname specifies the device name of the magnetic tape or diskette to which or from which you are moving files.

filename(s) specifies one or more files to be moved. If you do not specify filename arguments for a dump operation, IMOVE dumps all files from your current directory, including subordinate directories and their files to the named device. If you do not specify filename arguments in a load operation, IMOVE loads all files from the named device to your current directory.

The *filename* argument can include a pathname to specify files outside of the current directory, but the use of templates is prohibited.

IMOVE transfers files in dump format from one device to another. When you dump disk files (with the /D switch), the dump format retains all file and directory information in a compact format. When you load the files back to disk (with the /D/F switches), IMOVE uses the dump format information to reconstruct the original files and directory structures on disk.

Whether you are dumping or loading files, be sure to select the appropriate current directory, as IMOVE uses the current directory as a reference point. Also, be certain to initialize the device you plan to use prior to issuing the IMOVE command.

IMOVE supports multiple diskette operations, but does not support multiple tape operations.

If the files in a single dump session do not fit on one diskette, IMOVE fills each diskette and prompts you to supply additional diskettes until all of the files are dumped. It is important to label the outside of each diskette used with the date of the dump and the order of sequence, since you must load the diskettes in the same order in which they were dumped.

IMOVE is compatible with MOVE utilities on AOS-based operating systems. This means that you can load files to DG/RDOS that were created by an AOS-based MOVE utility, and you can load files to an AOS-based system that were created by IMOVE. Because line termination characters are different for RDOS and AOS systems, include the /C switch to ensure that RDOS text files are readable on AOS systems, and AOS files are readable by DG/RDOS. Do not use the /C switch when transferring .SV files.

The diskettes you use for backup must be hardware formatted, but need not be software formatted with DKINIT.

IMOVE (continued)

Global Switches

/C Converts line termination characters in files. Use to transfer text files (but not binary (.SV) files) between DG/RDOS and AOS-based operating systems.
When dumping, /D/C converts <CR> to NEW LINE.
When loading, /D/F/C converts NEW LINE to <CR>.

/D Specifies dump format. /D is required with all IMOVE functions except /H (help).

/D dumps files to diskette.
/D/F loads files from diskette.
/D/T dumps files to magnetic tape.
/D/T/F loads files from magnetic tape.

/F Loads files from magnetic tape or diskette to current directory.

/H Displays help text about the IMOVE command line and switches. No files will be transferred, regardless of other switches.

/L Lists on line printer ($LPT) names of files moved.

/N Specifies no file transfer, but your command line is interpreted, and the filenames of the files to be moved appear at your terminal. Use to test the effect of your command line.

/O Overwrites existing files in the current directory with identically named backup files from diskette or magnetic tape in a restore (load) operation. Without /O, existing same-name files are preserved.

/R Chooses most recent version of a file in a restore (load) operation. If an existing file of the same name has the most recent creation date, it is preserved. If a backup file is most recent, it overwrites the existing file.

/T Moves files to or from magnetic tape. Required for magnetic tape operation. Without /T, IMOVE assumes a diskette.

/V Verifies that each file is moved by displaying its name at the terminal.

Examples

1. The following example dumps files to diskette.

```
R
DIR STEAMBOAT ↓
R
INIT DJ1 ↓
R
IMOVE/D/V DJ1 ↓
Please insert disk 1, then press NEW LINE to continue. ↓
07-APR-84
   TWAIN.SR
   TWAIN.RB
   TWAIN.SV
   HUCK1.
   HUCK2.
   .
   .
   .
   RIVERQUEEN.SV
This diskette has been exhausted: DJ1

Please insert disk 2, then press NEW LINE to continue. ↓
   BECKY.
   FULLER.SV
   DEWITT.OL
   .
   .
   .
   SAMUEL.
   CLEMENTS.DB
R
```

IMOVE (continued)

2. The following example loads back to disk the files dumped in the previous example.

```
R
DIR STEAMBOAT ↵
R
INIT DJ1 ↵
R
IMOVE/D/F/V DJ1 ↵
Please insert disk 1, then press NEW LINE to continue. ↵
07-APR-84
  TWAIN.SR
  TWAIN.RB
  TWAIN.SV
  HUCK1.
  HUCK2.
  .
  .
  .
  RIVERQUEEN.SV
This diskette has been exhausted: DJ1

Please insert disk 2, then press NEW LINE to continue. ↵
  BECKY.
  FULTON.SV
  DEWITT.OL
  .
  .
  .
  SAMUEL.
  CLEMENTS.DB
R
```

3. The following example dumps files to tape.

```
R
DIR FRAGMENTS ↵
R
INIT MT0 ↵
R
IMOVE/D/T/V MT0:0 ↵
  02-APR-84 11:45:43
  KRISPIES.
  FLAKES.
  CORN.
  CHEERIOS.
  RICE.
  WHEATIES.
  GRAPENUTS.
R
RELEASE MT0 ↵
R
```

4. The following example loads back from tape the files dumped in the previous example.

```
R
DIR FRAGMENTS ↵
R
INIT MT0 ↵
R
IMOVE/D/T/F/V MT0:0 ↵
  02-APR-84 11:45:43
  KRISPIES.
  FLAKES.
  CORN.
  CHEERIOS.
  RICE.
  WHEATIES.
  GRAPENUTS.
R
RELEASE MT0 ↵
R
```

INIT

Command

Initialize a directory or a device for I/O

Syntax

INIT {directory_name / device_name}

Description

INIT introduces a device or directory to the operating system for I/O by initializing that device or directory. Before the introduction, all files on the device or directory are effectively closed.

To initialize a directory, the directory's parent directory must also be initialized. If you use a pathname as an argument to INIT (e.g., INIT DP1:SECPART:SUBDIR), all directories listed in the specifier are initialized.

After you initialize a directory or device, you can access all files within it (for example, through pathnames or links). All files on an initialized tape unit, directory, or diskette are available until you release the device or directory with the RELEASE command.

Your operating system has a limit on the number of directories that can be initialized at any one time. This is a variable that is set at system generation with SYSGEN; the maximum number is 64. If you attempt to initialize a directory when at the limit, you receive the message *NO MORE DCBS*. Use the RELEASE command to remove directories from system recognition to avoid running into the limit.

Bootstrapping automatically initializes the directory that holds the current system (i.e., the master directory), and makes it the current directory.

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

Table 5-1 lists RDOS device names.

INIT with the /F switch performs the final step in software formatting of disk or diskette media. This is the only switch to INIT.

Global Switch

/F Fully initializes a tape, disk, or diskette by erasing all existing files and information from the tape or disk(ette). The /F switch is ignored when issued for a secondary partition or subdirectory.

On a disk or diskette, INIT/F prepares the device to accept RDOS files by writing a new system file directory (SYS.DR) and map directory (MAP.DR). INIT/F of a disk or diskette is the last step in software formatting. After processing disk media with DKINIT, you initialize it fully with INIT/F. INIT/F destroys any existing data on disk media.

On a tape, INIT/F rewinds the tape and writes two end-of-file (EOF) marks at the beginning of the tape, effectively erasing the tape by allowing the system to overwrite existing information.

Because INIT/F erases data, the CLI issues a CONFIRM inquiry after INIT/F. You must type Y to execute the command; the CLI displays YES at the console and fully initializes the device. The CLI interprets any other character you type as NO; it displays *NO* at the console and cancels the command. Note that the CLI waits for a response when it encounters an INIT/F command in a macro.

Examples

1. *R*
   INIT DJ1 ⤸
   R

   Initializes the primary partition of the diskette in drive DJ1. Directories subordinate to primary partition DJ1 must now be initialized separately.

2. *R*
   INIT SECPART:NEWDIR ⤸
   R

   Initializes both secondary partition SECPART and its subdirectory NEWDIR. All files in directories SECPART and NEWDIR are available for use.

3. *R*
   LIST ABC:FOLLY ⤸
   NO SUCH DIRECTORY: ABC:FOLLY
   R
   INIT ABC ⤸
   R
   LIST ABC:FOLLY ⤸
   ABC:FOLLY 88 D
   R

   In this example, we try to LIST a file in directory ABC. On receiving the message NO SUCH DIRECTORY, we initialize ABC, and the command works when we try it again.

4. *R*
   INIT/F MT1 ⤸
   CONFIRM? Y ⤸
   YES
   R

   Fully initializes the tape on tape drive MT1 by rewinding the tape, erasing it, and writing two end-of-file (EOF) marks at the beginning of the tape. (The system begins writing files on the tape at the double end-of-file.)

5. *R*
   INIT/F DJ0 ⤸
   CONFIRM? N ⤸
   NO
   R

   In this example, when the system requests confirmation for a full initialization of diskette DJ0, we type N so we can make certain that this diskette can be overwritten. The command aborts and returns us to the CLI.

LDIR

Command

Display the name of the previously current directory

Syntax

LDIR

Description

LDIR displays the name of the previously (or last) current directory. The name returned by LDIR is the value stored in the CLI variable %LDIR%.

See the DIR command description for information on changing the current directory.

No switches.

Examples

1. *R*
   GDIR ↵
   DRONER
   R
   DIR DJ0 ↵
   R
   GDIR ↵
   DJ0
   R
   LDIR ↵
   DRONER
   R

   The GDIR command displays the name of the current directory, DRONER. The DIR command changes the current directory to directory DJ0. GDIR displays the name of the new current directory, DJ0. LDIR displays the name of the previous current directory, DRONER.

2. *R*
   LDIR ↵
   DRONER
   R
   DIR %LDIR% ↵
   R

   This example uses the CLI variable %LDIR% to change the current directory back to directory DRONER.

LFE

Utility

Create, edit, and analyze library files

Syntax

LFE { A inputmaster *[...argn$_n$] [listing_device/L]*
A/M inputmaster$_1$ *[...inputmaster$_n$] [listing_device/L]*
D inputmaster *[outputmaster/O]* arg$_1$ *[...arg$_n$]*
I inputmaster *[outputmaster/O]* file$_1$*[...file$_n$]*
M outputmaster/O inputmaster$_1$ *[...inputmaster$_n$]*
N *[outputmaster/O]* file$_1$ *[...file$_n$]*
R inputmaster *[outputmaster/O]*arg$_1$file$_1$ *[...arg$_n$ file$_m$]*
T inputmaster *[listing-device/L][...inputmaster$_n$]*
X inputmaster arg$_1$ *[...arg$_n$]* }

Description

Edit and analyze library files, which are sets of relocatable binary files having special starting and ending blocks, and which are usually designated by the extension .LB.

A, D, I, M, N, R, T, and X are keys designating LFE functions; inputmaster and outputmaster represent library files; arg represents logical records on the library files, and files are update files.

Action taken by the LFE depends upon the function given in the command. The function keys are as follows.

A Analyze global declarations of inputmaster or a series of inputmasters, or of logical records specified from one inputmaster. Output is a listing with symbols, symbol type, and flags; no new output library file is created. Default output is to SYSOUT.

D Delete logical records, specified by args from inputmaster, producing outputmaster. Default output is to diskfile D.L1.

I Insert relocatable binary files, merging with logical records of inputmaster in the manner described under "Switches." Default output is to disk file I.L1.

M Merge library file (inputmasters) into a single library file named outputmaster. Default output is to disk file M.L1.

N Create new library file, outputmaster, from one or more relocatable binary files given by files. Default output is disk file N.L1.

R Replace logical records in inputmaster by relocatable binary files, producing outputmaster. Arguments are paired, with the first being the logical record and the second the relocatable binary file that replaces the logical record. Default output is to disk file R.L1.

T Output to the listing device (SYSOUT by default) the titles of logical records on inputmaster.

X Extract from library file, inputmaster, one or more relocatable binary files given by args. Output is one or more relocatable binary files named args.RB.

The name LFE cannot be changed.

LFE (continued)

Key Switches

/M Multiple input library files. The switch modifies the A function (not the filename LFE) and causes all library file names following, except the listing file, to be analyzed as one library.

Local Switches

/A Insert after. The switch modifies a logical record in an I function command line. Arguments following the switches are inserted after the logical record whose name precedes the switch. When neither a /A nor /B switch is given, inserts are made at the beginning of the new library file.

/B Insert before. The switch modifies a logical record in an I function command line. Arguments following the switch are inserted before the logical record whose name precedes the switch. When neither a /A nor /B switch is given, inserts are made at the beginning of the new library file.

/E Error listing directed to given filename.

/F Output analysis of each relocatable binary on a separate page (used only with /L).

/L Listing file. The switch modifies the name of a file to be used as listing output in the A function command line. (SYSOUT is used by default.)

/O Output library file. The switch always modifies outputmaster in D, I, M, N, and R functions.

Extensions

If the .LB extension for inputmaster or the .RB extension for an update file are not given in the command, LFE searches for inputmaster.LB or arg.RB, respectively. If not found, LFE searches for inputmaster or arg, respectively.

Examples

1. *R*
   ```
   LFE N ZED.LB/O ZED.RB ZEPHYR.RB ⤸
   ```
   Creates a new library file called ZED.LB from binary files ZED.RB and ZEPHYR.RB.

2. *R*
   ```
   LFE A/M ZUT1.LB ZUT0.RB ⤸
   ```
   Analyzes as one library ZUT1.LB and the binary file ZUT0.RB. Analysis goes to the line printer.

3. *R*
   ```
   LFE I MAL.LB MAL1.LB/O RSK/A MOD1 ⤸
   ```
   In library MAL.LB, inserts binary MOD1.RB after binary RSK. Name the new library MAL1.LB.

4. *R*
   ```
   LFE A MYLIB(1,2,3).LB MYLIB(1,2,3).AN/L ⤸
   ```
   Analyzes libraries MYLIB1.LB, MYLIB2.LB, and MYLIB3.LB; stores analysis in disk files MYLIB1.AN, MYLIB2.AN, and MYLIB3.AN.

5. *R*
   ```
   LFE N $PTP/O A.RB C.RB/B ⤸
   ```
   Creates a new library file from binary files A.RB and C.RB; punches it on the paper tape punch.

LINK

Command

Create a link to a file in another directory

Syntax

LINK { resolution_file/2 | link_file *[resolution_file]* }

Description

resolution_file/2 creates a link file with the same name as the resolution file, when the resolution file is in the parent directory of the current directory.

link_file assigns a name to the link file. This argument can be a pathname. If you omit *resolution_file*, the CLI creates a link to link_file in the parent directory.

resolution_file is the name of the file to which you are forming a link, the destination of the link.

The LINK command creates a file in the current directory that points to a file in another directory (the *resolution_file*). With a link file, you can enter its name from the current directory, and the CLI accesses its resolution file. The link file contains no data but only points to another file.

To use a link, all directories involved in the resolution chain must be initialized.

The use of links is the preferred way to execute programs and CLI utilities from directories other than those in which they reside. If a utility file (e.g. utility.SV) comes with an overlay file or error file (e.g. utility.OL or utility.ER), you need to create links to these files as well. The descriptions for each utility in this chapter list the files you will need to execute that utility.

To create a link entry to another disk or diskette, use a pathname in *resolution_file*. Be sure that the path contains only one directory specifier.

LINK resolution_filename/2 creates a link to the current directory's parent directory. The parent directory could be a disk or diskette (primary partition) or a secondary partition. Since utility programs usually reside in the master directory (along with the operating system), you could link to each utility file using this format — if the master directory is the parent directory.

Use LINK link_file resolution_file

- If you want the name of your link file to differ from the name of the resolution file (it will be an "alias" name). For an alias link name, if the resolution file is an executable program file (.SV) or an overlay file (.OL), your link name must include the .SV or .OL extension.
- If the resolution file is not in the current directory's parent directory.

Use LINK link_file if the resolution file is in the current directory, and you want to assign an alias in the link_file argument.

It is good practice to assign a link file the name of its resolution file so that you don't have to keep track of different names for a file that performs the same function.

If you create a link file to another link file, the CLI will link to its resolution file, and so on until it reaches the destination.

You can create a link to a file whose attributes prohibit linking (see the CHATR and CHLAT commands), but the link will not work.

LINK (continued)

CAUTION: To remove a link, use the UNLINK command. If you attempt to remove a link with the DELETE command, you will delete the resolution file and not the link file. The resolution file can be protected from this mistake if it is assigned the attribute P. (See the CHATR and CHLAT command for information about assigning attributes to files.)

No switches.

Examples

1. *R*
   DIR BRANCH ⤸
   R
   LINK EDIT.SV/2 ⤸
   R

   This creates the link file EDIT.SV to the editor in DP0 (BRANCH's parent directory). Anyone in directory BRANCH can now edit a file without using a directory specifier. The link file appears in LIST command output as follows:

   R
   LIST EDIT.SV ⤸
   EDIT.SV @:EDIT.SV
   R

   LIST displays the link filename and its resolution. The at sign (@) indicates the parent directory.

2. *R*
   DIR LEAVES ⤸
   R
   LINK TXT.SV DP0:EDIT.SV ⤸
   R

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

3.
```
R
DIR DP1 ↵
R
LINK EDIT.SV DP0:EDIT.SV ↵
R
```

This creates a link file on diskette DP1 to the resolution file EDIT.SV on disk DP0.

4.
```
R
DIR DJ1 ↵
R
LINK EDIT.SV/2 ↵
R
EDIT MYFILE ↵
LINK DEPTH EXCEEDED: EDIT.SV
R
```

In this example, the LINK command does not report an error, but when we try to use the link we find a problem. In this case, the current directory is diskette DJ1. We use the form of the link command that searches for the resolution file in the parent directory. DJ1 does not have a parent directory, and the link was made to itself.

To correct the problem, you would remove the link and recreate it, including a directory specifier naming the directory in which the resolution file resides, as follows:

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

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

LIST

Command

Display information about the files in a directory

Syntax

LIST *[[directory_name:] file ...]*

Description

When you omit arguments and switches, LIST displays all nonpermanent files in the current directory, in the form:

FILENAME x y

where *x* gives the size of the file in bytes, and *y* gives the file's characteristics and attributes. (Table 10-6 describes file characteristics; Table 10-7 describes file attributes.)

When you include global switches with the LIST command, LIST displays a variety of information about each file. The information depends on the switches used, and can include:

- file size, in bytes
- file characteristics (see Table 10-6)
- file access attributes and link access attributes (see Table 10-7)
- file creation date and time, in the form mm/dd/yy hh:mm
- the date the file was last opened for I/O, in the form mm/dd/yy
- the logical octal address of the first block in the file, in the form nnnnnnnn
- the use count of the file (in decimal) (the number of users who currently have the file open for I/O — see the CLEAR command for more information)

You can use local switches to specify files that were created before or after a specified date, or to exclude specified files from the display.

For a link file, LIST displays only the filename of the link and its resolution filename. A @ symbol indicates that the resolution file resides on the current directory's parent partition. For example, if directory ABC is a subdirectory of partition ALPHABET, ALPHABET is ABC's parent partition. The LINK command describes how to create and use link files.

Table 10-6 lists file characteristics and their meanings. RDOS assigns file characteristics to a file when it is created.

Table 10-6. File Characteristics

Characteristic	Meaning
D	Random file.
C	Contiguous file.
L	Link file.
T	Partition file. (Partitions also have characteristics C and Y.)
Y	Directory file. (Partitions, subdirectories.)

Table 10-7 lists file and link access attributes and their meanings. See the CHATR command to change a file's access attributes; see the CHLAT command to change a file's link access attributes.

Table 10-7. File Attributes

Attribute	Type of File	Comment
P	Permanent file	You can't delete or rename files with P set. Note that the P attribute is not recognized by the BUILD, DUMP, LIST, LOAD, or MOVE commands unless you include the /A switch.
S	Executable program file (.SV)	Allows file execution. The RLDR utility assigns S when creating a file.
W	Write-protected file	You cannot edit or modify a file with W set.
R	Read-protected file	You cannot display or read a file with R set.
A	Attribute-protected file	You cannot change the file's attributes, including attribute A.
N	Resolution-protected file	You cannot access the file by way of a link.
?	User-definable attribute	You can use this to mark files for your own record-keeping.
&	User-definable attribute	You can use this to mark files for your own record-keeping.

Global Switches

/A Lists all files within the current directory, *including permanent files.* If you do not use the /A switch, LIST excludes permanent files (those with the P attribute) from the display. For each file, LIST displays all categories of file information.

/B Brief listing, displays filenames only.

/C Includes file creation date and time in the display.

/E Displays all categories of file information (overrides switches /B, /C, /F, /O, and /U).

/F Includes the logical address of the first block in the file in the form nnnnnnn; displayed as 0 if unassigned.

/K Excludes link files from the list.

/L Sends the LIST command output to the line printer.

/N Lists only link files.

/O Includes in the display the date the file was last accessed.

/S Displays the list of files alphabetically, sorted by filename.

/U Includes the file use count in the display.

LIST (continued)

Local Switches

mm-dd-yy/A Lists files with creation dates on or after the specified date. Arguments **mm** (month) and **dd** (day) can be one or two digits.

mm-dd-yy/B Lists files with creation dates before the specified date. Arguments **mm** (month) and **dd** (day) can be one or two digits.

filename/N Exclude files that match this **filename**.

Local switches have effect only when the other arguments in the command line are null arguments (when no files are specified) or template arguments.

Examples

1. *R*
   LIST ↵

```
FILE1.      68    D
PROGX.     479    D
FILE2.      50    D
R
```

   Lists all nonpermanent files in the current directory, and displays the size of each in bytes, and any attributes and characteristics. In this example, all files have the characteristic D (indicating a random file), and no attributes.

2. *R*
   LIST/A/S ↵

```
FILE1.      68    D     01/06/84    16:37    02/16/84    [006564] 0
FILE2.      50    D     02/16/84    13:24    02/16/84    [006565] 0
PROGX.     479    D     02/23/84     9:43    02/23/84    [006566] 0
PROJLIST   145    PD    01/04/84    10:56    02/24/82    [006567] 0
R
```

   Lists all files in the current directory, including permanent files, and displays the following, from left to right:

   - the size of the file
   - file characteristics and attributes
   - file creation date and time
   - the date the file was last opened for I/O
   - the logical address of the first block in the file
   - the file use count

   The /S switch causes LIST to sort the filenames in alphabetical order.

3. *R*
LIST/K/S -.SV 1-25-84/A ↵

PROG4.SV 342 D
PROG5.SV 654 D
INVEN.SV 1305 D
R

Lists all executable program files (-.SV), sorted alphabetically (/S), that were created on or after January 25, 1984 (1-25-84/A). The output does not include link files (/K).

4. *R*
LIST/A SUBDIR:EDIT.SV ↵
EDIT.SV @:EDIT.SV
R

Lists file EDIT.SV in directory SUBDIR. EDIT.SV is a link file whose resolution is a file of the same name residing in the parent directory of SUBDIR. The /A switch ensures that LIST will locate the file EDIT.SV, even if EDIT.SV is a permanent file.

LOAD

Command

Restore DUMP files to the current directory

Syntax

LOAD *[source:]*dumpfilename *[filename ...]*

Description

source: specifies the disk, diskette, directory, magnetic tape, or paper tape that contains the files you are loading to your current directory.

dumpfilename specifies the filename you assigned to the dump file when you created it with the DUMP command.

filename(s) specifies the files to be transferred into your current directory from the specified source directory or device. This argument can include templates when referring to files in the dump file that are not directories.

LOAD restores the files and directories contained in a dump file (created with the DUMP command) on a directory or device to the current directory.

If you omit filenames and switches, all nonpermanent files from the dump file are loaded. With global switches, you can select filenames for loading. The /N (no load) global switch interprets the command line without loading any files, to allow you to check the files that will be loaded.

Files to be loaded must have different filenames from files in the current directory (unless you specify the /N, /O, or /R switches). Neither dumping nor loading changes a file's attributes, creation date, or directory characteristics.

Be certain that your current directory can accommodate the appropriate levels of subordinate directories to be loaded.

The LOAD command loads only files that were previously dumped with DUMP. For files dumped with the FDUMP or DDUMP utilities, use FLOAD or DLOAD.

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 error message.

Global Switches

/A Loads all files, and includes permanent (attribute P) files.

/B Displays the filenames of files loaded at the terminal.

/E Suppresses nonfatal error messages.

/K Excludes link files from the transfer.

/L Produces on the line printer a list of the filenames loaded. Overrides /V switch. When used with global /N switch, produces line printer listing without loading any files.

/N Does not load files; displays a list of the filenames at the terminal.

/O Deletes files in the current directory that have the same filenames as the files being loaded, replacing the old file with the file loaded.

/R Loads the most recent version of file. When a file to be loaded has the same name as a file in the current directory, RDOS checks each file's creation date. If the existing file is older, it is replaced by the file awaiting loading. If the existing file has the most recent creation date, it is retained and the new file is not loaded.

/V Verifies the load by listing filenames of each file loaded at the terminal. 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 Loads only files created this date or after. Arguments mm and dd can be one or two digits.

mm-dd-yy/B Loads only files created before this date. Arguments mm and dd can be one or two digits.

filename/N Excludes files that match filename from the load. filename may contain templates.

Local arguments /A, /B, and /N do not apply to filenames that are specified explicitly (i.e., without the use of templates) on the command line.

Examples

1.
```
R
INIT MT0 ↵
R
LOAD/V MT0:1 ↵
   NORTON
   BSA
   SUZUKI.
   KAWASAKI.
* DAVIDSON.DR
   HARLEY.
   HONDA.
R
RELEASE MT0 ↵
R
```

Loads into the current directory all nonpermanent files previously dumped to file 1 of the tape on tape drive MT0. LOAD displays the filenames as they are loaded (/V). Indents and asterisks reflect the directory structure of the files loaded.

2.
```
R
LOAD/V $PTR -.SV ↵
LOAD $PTR, STRIKE ANY KEY. ↵
EDIT.SV
ASM.SV
R
```

Loads into the current directory all nonpermanent files with the extension .SV from the paper tape reader, and lists the files loaded at the terminal.

LOAD (continued)

3. *R*
   LOAD/L/A MT1:3 -.SV 12-15-84/A TEST-.-/N ↵
   R

   Loads from file 3 of the tape on MT1 all files with a .SV extension except those files whose names begin with the characters TEST, and any files created before December 15, 1984; /L lists the filenames on the line printer.

4. *R*
   DIR DZ0 ↵
   R
   LOAD/V/R DP4:84JUN22.BU -.SR ↵
   R

   Loads (into current directory DZ0) all .SR files from dumpfile 84JUN22.BU, on disk DP4. (See DUMP command example.) The /R switch specifies that if any file to be loaded has the same name as a file in DZ0, check the file's creation date, and do not load the file unless the dumped version is newer.

LOG *Command*

Record the current CLI session in the log file (F)LOG.CM

Syntax

LOG *[password] [directory/O]*

Description

LOG records CLI dialog that appears at your terminal in a file named LOG.CM (or FLOG.CM for a foreground CLI). The CLI creates the log file, or appends information to the existing log file, then records each line of CLI dialog in it. Only one current log file may exist at a time in any ground.

The ENDLOG command closes the log file. Under RDOS and DOS, the log file closes itself when you release the master directory or issue the BOOT command. Under DG/RDOS, you should always use ENDLOG to close the log file.

You cannot examine, print, or delete the log file while it is open.

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 Places a header at the beginning of LOG.CM. This header contains the title LOG FILE, and directory and date information.

/T Traces 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

directory_name/O Writes LOG.CM in the specified directory, overriding the default of using the current directory.

LOG (continued)

Examples

1. *R*
   LOG/H GSTONE ↵
   R

   Records 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] 10/17/84 3:0:0*

   GATE is the name of the current directory.

2. 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 DPOF.

   If you open the log file with global /T (trace) switch and type

   LAST ↵

   the system writes the following lines into the log file:

   LAST
   = = > LAST
   = = > DIR DPOF
   = = > MESSAGE NOW IN DIRECTORY DPOF
   NOW IN DIRECTORY DPOF
   R

   If you open the log file without /T, these lines appear in the log file without the trace, as follows:

   LAST
   NOW IN DIRECTORY DPOF
   R

MAC

Utility

Assemble source file(s) with the Macroassembler to produce a relocatable binary (.RB) file.

Syntax

MAC filename *[...filename]*

Description

The Macroassembler, MAC, creates a relocatable binary (.RB) file from assembly language source files. Once you produce the .RB file and handle any errors MAC detects, you can use the .RB file to create an executable .SV file with the RLDR utility.

MAC will use macro definitions that you specify in your source file with macro calls, and incorporate them as part of the .RB file. See the *RDOS/DOS Assembly Language and Program Utilities* manual for instructions.

If you omit switches, MAC produces a binary file named filename.RB and no listing file. You can include switches to produce an assembly listing, an error listing, a symbol table file, and a symbol cross-reference listing.

To use MAC, you need to have access to the following files:

MAC.SV

MACXR.SV

MAC.PS

NOTE: If you receive many undefined symbol messages from MAC, try creating a new MAC.PS file.

For the source files you specify on the command line, the CLI searches for filename with the .SR extension first, and then searches for filename.

MAC (continued)

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.
W	Text error.

Global Switches

/A Cross-references all user and semipermanent symbols (symbols defined by pseudo-ops). Use with global /L for a line printer copy; use with local /L for a disk file copy.

/E Suppresses error messages at the terminal. If you do not specify the creation of a listing file (global or local /L), error messages cannot be suppressed with /E.

/F Generates or suppresses a form feed on printed listings, as necessary to produce an even number of listing pages. By default, MAC generates one form feed after listing each filename.

/K Keeps the MAC.ST symbol table after the assembly. By default, MAC.ST is deleted at assembly completion.

/L Appends an assembly listing to disk file *filename.LS*; /L creates the file if none exists, using the filename of the first file on the command line.

/M Flags multiple-defined symbols on pass one.

/N Suppresses creation of the .RB file (useful for checking assembly errors).

/O Overrides all listing suppression controls (as specified with .NOLOC, .NOMAC, etc.).

/S Skips pass two and copies MAC.ST (symbol table and macro definitions) to file MAC.PS. Deletes the old MAC.PS (if any) first.

/T Recognizes and stores eight-character symbols from the source file(s). The resulting binary file is in extended .RB format. The cross-reference will show five-character symbols. By default, MAC recognizes and stores only the first five characters of symbol names. For restrictions on use of this switch, see *RDOS/DOS Assembly Language and Program Utilities.*

/U Includes user symbols in the relocatable binary output.

Local Switches

name/B Assigns the filename **name** to the .RB file (overrides global /N).

name/E Writes error messages to file **name**.

name/L Writes the listing and the cross-reference to file **name** (overrides global/L).

name/S Skips 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 **name** as a permanent symbol file for this assembly. If you omit this switch, the assembler uses file MAC.PS.

Examples

1. *R*
   MAC/L ZORRO ↵

   Assembles source file ZORRO.SR or ZORRO, producing relocatable binary file ZORRO.RB. The /L global switch sends the listing to file ZORRO.LS.

2. *R*
   MAC LIB/S A B C $LPT/L ↵

   Assembles files A.SR or A, B.SR or B, and C.SR or C, producing binary file A.RB. The local /S switch scans file LIB (or LIB.SR) on pass one for macro definitions and values for externals in A, B, and C. The listing is sent to the line printer ($LPT/L).

3. *R*
   MAC/L/U EX<1 2 3 4> ↵

   Assembles EX1, EX2, EX3, and EX4 (with or without .SR extensions), producing EX1.RB. The global /U switch includes user symbols in file EX1.RB; the /L switch sends the listing to file EX1.LS. (See Chapter 2 for an explanation of the use of angle brackets.)

MCABOOT *Command*

Boot an operating system or execute a program on another CPU via an MCA line (RDOS only).

Syntax

```
MCABOOT { MCAT:n [sysname/S] [filename...] }
        { MCAT:n program/S [filename...]   }
```

Description

MCAT:n specifies the Multiprocessor Communications Adapter (MCA) transmitter, where n is a number from 1 to 15 that indicates the MCA receiver. (To use the MCAT attached to the secondary controller, use the format MCAT1:n.)

sysname/S names the system filename you want to transmit. You can omit this argument to transmit a system file having the default name SYS.

program/S names any program that the BOOT command can execute. (See the BOOT command description.)

filename names file(s) that you want to send along with the system file or program file (no directory pathnames allowed). The global /F switch must accompany this argument.

MCABOOT transmits an operating system or executable program via an MCA (Multiprocessor Communications Adapter) line. The command is most useful when the receiving CPU's disk has no RDOS system file or an inappropriate RDOS file. (If the receiver's disk has an operating system file, it could be bootstrapped conventionally — see the BOOT command.)

Include the global /F switch only when the receiving CPU's disk is new or holds valueless material: /F specifies full disk initialization, which destroys all existing files on the disk. If you specify /F, the CLI save and overlay files are sent along with any other filenames you enter: the receiving CPU writes all these to its disk after the full initialization.

Omit the global /F switch when the receiver's disk holds needed data. MCABOOT performs a partial initialization with overlays, and does not send the CLI files. When you use MCABOOT without /F, you can transmit only one system file or executable program file, and you cannot specify additional files.

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(octal) (for network MCA) or 1000047(octal) (for network MCA1), in the data switches, pressing RESET, then PROGRAM LOAD.

At transmission, if you specified an operating system, the MCA bootstrap is sent, followed by the system save and overlay files. If you specify /F, CLI.SV and CLI.OL is sent, followed by the optional filenames. After the files have been transmitted, the system requests date and time information at the receiver's console, as in a traditional bootstrap. The CLI then gains control at the receiving CPU.

If you have transmitted an executable program, it will gain control at the receiver.

Global Switch

/F Performs a full initialization on the receiver's disk. (If you omit this switch, MCABOOT performs partial initialization with overlays.)

Local Switch

name/S Sends a system (or program) name other than the default system name (SYS.SV/SYS.OL). You can omit the .SV extension.

Examples

1. *R*
   MCABOOT/F MCAT:2 ↵

   Fully initialize the disk of CPU2, and send the default operating system — SYS.SV and 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 logon information on its console.

2. *R*
   MCABOOT/F MCAT:2 FORT.SV FIV.SV FORT.LB ↵

   Transmit the default system (SYS.SV and 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.

3. *R*
   MCABOOT MCAT1:3 32K.SV/S ↵

   Transmit 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

Command

Display the name of the master directory

Syntax

MDIR

Description

MDIR displays the name of the master directory, the directory in which RDOS system files are kept. The value that RDOS returns for MDIR is the same as that returned for the %MDIR% CLI variable.

No switches.

Examples

1. *R*
   MDIR ⤸
   DE0
   R

   Primary partition DE0 is the master directory.

2. *R*
   DIR DE0 ⤸
   R

   This command, which makes master directory DE0 the current directory, is equivalent to the command

   R
   DIR %MDIR% ⤸
   R

MEDIT

Utility

Invoke the Multiuser Text Editor (RDOS)

Syntax

MEDIT terminals *[clock_units]*

Description

Enables several users to edit text at the same time, on the number of terminals specified. Serves each user as a single user, though response time is degraded slightly. See the EDIT command description in this dictionary and the *RDOS/DOS Text Editor* manual for more information.

The terminals argument signifies the maximum number of terminals you want MEDIT to support. Enter a decimal number for terminals, for example, 8.

The optional *clock_units* argument indicates the number of system real-time clock units after which RDOS should force task rescheduling.

MEDIT is identical to EDIT but for these exceptions:

- MEDIT ignores CTRL-A.
- MEDIT ignores CTRL-C.
- On all MEDIT terminals, including the master console, the command H$$ does *not* return control to the CLI. H in MEDIT is equivalent to UH$$ (close I/O file and restore attributes). CTRL-F returns control to the CLI.

No switches.

Example

```
MEDIT 6 ↵
*
```

This allows text editing at 6 terminals simultaneously. The MEDIT asterisk prompt (*) appears at terminals that are on-line.

MESSAGE

Command

Display a text string

Syntax

MESSAGE *["]* textstring*["] ...["] [textstring] ["]*

Description

MESSAGE displays the **textstring** you specify as a message on the console. This command is useful for indirect and macro command files.

The /P global switch allows you to delay further execution until someone strikes a key on the keyboard. This is useful if your macro or indirect file directs a user to mount a device, such as a disk or tape, and you need to halt processing until the device is ready.

You can enter the **textstring** inside quotation marks (e.g., "HELLO"), or omit the quotation marks. If you use quotation marks, the MESSAGE command returns all characters literally, except for the quotation marks (which are text string delimiters), semicolons, carriage returns, and form feeds (which are command delimiters). A single quotation mark (an unmatched one) within a quoted string is illegal. The quoted text string cannot include more than 72 characters.

If you omit quotation marks, the CLI interprets special characters such as % for variables and @ for indirect files. Angle brackets (<>), parentheses, commas, and slashes (/) are interpreted as they would be in a CLI command line. (To have these characters interpreted literally, place them inside quotation marks.) Unquoted text strings are delimited as a CLI command — that is, with ↓ or ; (semicolon). An unquoted text string can contain as many as 132 characters.

Global Switch

/P Directs a pause: after displaying textstring, displays the message *STRIKE ANY KEY TO CONTINUE*, and waits for someone to strike a key at the terminal keyboard.

Examples

1. *R*
   TYPE LOGON.MC ↵
   DIR USERSDISK
   CHATR SYS.<SV,OL> +WP CHATR CLI.<SV,OL,ER> +WP
   MESSAGE WELCOME ABOARD.
   MESSAGE USER DIRECTORIES AVAILABLE ARE:
   LIST -.DR
   R

   The above sequence shows the contents of the macro file LOGON.MC. Next we execute LOGON.MC.

   R
   LOGON ↵
   WELCOME ABOARD.
   USER DIRECTORIES AVAILABLE ARE:
   SECONDPART.DR507904CTY
   SUBDIR.DR 512 DY
   SUBDIRA.DR 512 DY
   .
   .
   .
   R

2. The macro file DUMPTAPE, below, uses the /P switch to delay execution until someone takes a particular action.

   TYPE DUMPTAPE.MC ↵
   CHATR SYS.<SV,OL> -WP CLI.<SV,OL> -WP ↵
   MESSAGE THE CURRENT DIRECTORY IS %GDIR% ↵
   MESSAGE "MOUNT DUMP TAPE ON MT0, AND READY THE TAPE DRIVE" ↵
   MESSAGE/P ENJOY ↵
   INIT MT0 ↵
   DUMP/V MT0:0 ↵
   CTRL-Z
   R

MKABS *Command*

Make an absolute binary file from a disk program (.SV) file

Syntax

MKABS program_name absolute_filename

Description

MKABS creates an absolute binary file from a memory image .SV file. After you convert an executable program into an absolute binary, you can execute it without a disk, using paper tape with the Binary Loader.

Extensions

The CLI searches for program_name.SV; if it does not find it, it searches for program_name.

Global Switches

/S Starting address switch. The starting address of the .SV file as specified in USTSA of the file will be used as the address for the absolute binary start block. For paper tape, the default is a null start block that halts the Binary Loader when loading is completed.

/Z Begins save file at memory location 0; default is 16 (octal). This configures the file as a stand-alone program, which you cannot execute under RDOS, but which can execute by itself.

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.

Examples

1. *R*
   MKABS/Z FOO $PTP ↵

   Punch an absolute binary file on the paper tape punch from file FOO.SV or, if not found, from FOO.

2. *R*
   MKABS/Z ARK $PTP 1000/S ↵

   Punch an absolute binary file with a start block that specifies 1000_8 as the starting address.

3. *R*
   MKABS/Z SFILE.SV SFILEABS.SV ↵

   Make an absolute disk file SFILEABS.SV from disk file SFILE.SV.

MKSAVE *Command*

Make a disk .SV file from an absolute binary file

Syntax

MKSAVE absolute_filename program_name

Description

MKSAVE creates a memory image .SV file from an absolute binary file. MKSAVE appends the .SV extension to the program_name you supply, and assigns the S attribute to the created .SV file.

MKSAVE is often used (in place of RLDR) to process stand-alone programs that fulfill BOOT command requirements.

Global Switch

/Z Start save file at memory location 0 rather than at the default of 16 (octal). You can execute the resulting file as a stand-alone program (see the BOOT command), but you cannot execute it under the CLI.

Example

```
R
DIR DP0 ↵
R
MKSAVE/Z $PTR DP1:SHHH ↵
R
```

Creates a disk save file called SHHH.SV in directory DP1 from the absolute binary file loaded in the paper tape reader ($PTR). SHHH.SV begins at memory location 0; you can execute it by typing BOOT DP0, and responding to the *FILENAME?* query with SHHH.SV/A.

MOVE *Command*

Copy files from the current directory to the specified directory

Syntax

MOVE destination_dir *[... filename]*

Description

destination_dir specifies the name of the destination directory to which the files are moved.

filename specifies the files in the current directory that are to be moved. If you do not specify any filename argument, all nonpermanent files in the current directory are moved.

MOVE filename arguments permit the use of templates.

MOVE transfers copies of one or more files from the current directory to the specified destination directory. The destination directory must be initialized (see the INIT command).

For each file it transfers, MOVE preserves file statistics such as filename, length, attributes, time of creation, and time of last access.

Link files retain their original resolutions, so you may want to use the /K switch, and create new link files from scratch so their resolutions are valid for the new (destination) directory.

You cannot move a directory with MOVE.

DOS users: If you copy a file with MOVE from a write-protected diskette, the copy automatically receives attributes APW, which means that you can never delete or modify the copy.

Global Switches

/A Moves all files, including permanent files.

/D Deletes the original files from the current directory after copying them.

/K Excludes link files from the transfer.

/L Lists moved filenames on the line printer (overrides /V switch).

/R Retains most recent version of the file. When a file in the current directory has the same name as a file in the destination directory, the system checks both files' creation dates. If the file in the current directory is newer, it replaces the file in the destination directory. If the file in the current directory is older, no transfer takes place.

/V Lists on the terminal the names of each file moved.

Local Switches

mm-dd-yy/A Moves files created this date or after. Arguments mm (month) and dd (day) may be one or two digits.

mm-dd-yy/B Moves files created before this date. Arguments mm (month) and dd (day) may be one or two digits.

name/N Excludes any files that match name from the transfer.

oldname/S newname
Assigns newname to the file specified as the oldname argument, but retains its old name in the current directory.

/A, /B, and /N local switches apply to any filename arguments specified as templates, or to all files in the current directory when no filename arguments are specified. /A, /B, and /N ignore any files that are specified explicitly (that is, whose full filenames are given).

Examples

1. *R*
   MOVE/V DJ1 TEXT-.- ⤸
   TEXTA.OB
   TEXTA.SR
   TEXTA.SV
   TEXTOBJ
   TEXTVS
   R

   Copies all nonpermanent files in the current directory that begin with the letters TEXT to directory DJ1. /V causes MOVE to list the names of the files moved at the terminal.

2. *R*
   MOVE/D/K/V SOURCE -.SR TEST-.-/N ⤸
   CHART1.SR
   CHART4.SR
   GOCHART.SR
   SIMPLOT.SR
   XCHART.SR
   R

   Copies all nonpermanent files in the current directory (except link entries (/K) and files beginning with TEST (/N)) to directory SOURCE. /V causes MOVE to list the names of the files moved at the terminal. /D causes MOVE to delete the original files from the current directory after the transfer takes place.

3. *R*
   DIR ACCTSDUE ⤸
   R
   MOVE/A/V/R DZ0:LEGALNOTES -.AD 1-2-82/B ⤸
   ADR1ANCO.AD
   JOHNSMART.AD
   MAXSMART.AD
   R

   Transfers copies of all files having the extension .AD that were created before February 24, 1984, into directory LEGALNOTES on unit DZ0. The recent (/R) switch prevents the replacement of any files in directory LEGALNOTES that are more recent than files of the same name in the current directory.

4.
```
R
CDIR DE0:JONAS ↵
R
MOVE/V JONAS NEWPROG.- ↵
FILE DOES NOT EXIST: JONAS
R
```

The CDIR command creates a directory called JONAS in primary partition DE0. For the MOVE command, RDOS could not find directory JONAS because it is not initialized. The following example corrects the problem.

```
R
INIT DE0:JONAS ↵
R
MOVE/V JONAS NEWPROG.- ↵
NEWPROG.LS
NEWPROG.OB
NEWPROG.SR
NEWPROG.SV
R
```

This moves files beginning with NEWPROG from the current directory to directory JONAS, a subdirectory of disk DE0.

NSPEED

Utility

Invoke the NOVA Supereditor

Syntax

NSPEED *[filename]*

Description

Edit 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 ECLIPSE computers, see SPEED.

To use Superedit from another directory, link to both of its files: NSPEED.SV and SPEED.ER.

For more information, see the SPEED description in this chapter, and the RDOS/DOS Superedit Text Editor.

No switches.

Example

```
NSPEED FLEXIB ↓
CREATING NEW FILE
!
.
.
.
/UE$$
/H$$ R
```

Exclamation point (*!*) is the Superedit prompt. The H ESC ESC command sequence terminates Superedit and returns control to the CLI.

OEDIT

Utility

Edit octal locations in a disk file

Syntax

OEDIT filename

Description

OEDIT invokes the octal editor to examine and modify any location in any disk file in octal, decimal, or ASCII format. (The symbolic editor SEDIT offers slightly more versatility than OEDIT.)

See *RDOS/DOS Debugging Utilities* for documentation on OEDIT.

Example

```
R
OEDIT FOO.SV ⤸
.
.
.14/001672 ⤸
.
.
.
$Z
R
```

A period (.) is the OEDIT prompt. To return to the CLI, type ESC Z (echoed as $Z).

You can find the current NMAX requirements for any .SV file by issuing the OEDIT command:

404-16/nnnnnn

and find the ZMAX value by entering:

401-16/zzzzzz

The values nnnnnn and zzzzzz are returned by OEDIT, and indicate filename's current NMAX and ZMAX requirements.

OVLDR *Utility*

Create an overlay replacement file (RDOS and DG/RDOS only)

Syntax

OVLDR prog_name old_overlay/N new_overlay(s) old_overlay/N new_overlay(s)

Description

Creates and loads an overlay replacement (.OR) file for the overlay (.OL) file of a program previously created with RLDR. OVLDR can replace up to 127 overlays. The new overlays do not replace the old ones until you use the CLI command REPLACE to do so.

The argument old_overlay/N identifies the overlay that new_overlay is to replace. The old_overlay argument 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 .ENTO pseudo-op defined the name in the root program.

See the *RDOS System Reference* for more on overlays and *RDOS/DOS Assembly Language and Program Utilities* for a description of OVLDR.

OVLDR requires a symbol table in the .SV program. You can do this when you create the .SV file with RLDR by specifying RLDR's global /D switch, or by including an .EXTN .SYM statement in a program module.

Global Switches

/A Produces an additional symbol table listing with symbols ordered alphabetically. (Use with the local /L switch.)

/E Displays error messages at the terminal 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 Displays all numeric output in hexadecimal (radix 16). By default, output is in octal.

Local Switches

name/E Sends error messages to file name.

name/L Lists the symbol table from file name. The table lists symbols in numeric order.

old_overlay/N Specifies the old overlay identifier on the command line.

Example

R
OVLDR/A/E ROOT OLD1/N NEW1 NEW2 ROOT.OM/L ↵

Creates overlay replacement file ROOT.OR. When ROOT.OR replaces the original overlay file, ROOT.OL, overlays NEW1 and NEW2 replace old overlay OLD1 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 are written to disk file ROOT.OM. Error messages appear at the terminal.

The RLDR line that previously loaded ROOT might have looked something like this:

RLDR/D ROOT [OLD1,OLD2] ROOT1 [OLD3, OLD4] ↵

PATCH

Utility

Install patchfile in a program (.SV) or overlay (.OL) file

Syntax

PATCH *[program_name/S] [patch_file/P] [loadmap_filename/L]*

Description

PATCH installs the patches you inserted into a patchfile with the ENPAT utility. To apply Data General-supplied patches to an operating system, you must have instructed SYSGEN to save the load map file. If no load map exists for a program, you can patch it by appending the global /N switch to PATCH. If you omit arguments, PATCH asks for the information.

When the global /N switch has been specified and the patchfile contains symbolic references, the PATCH program asks the operator to supply the address of each symbol it encounters. The symbol name in question appears in quotation marks. Respond by entering an octal number of no more than five digits followed by a line terminator.

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. This utility is explained in greater detail in *RDOS/DOS Debugging Utilities.*

Global Switches

/I Do not display comments from the patchfile on the terminal.

/N There is no load map available.

Local Switches

name.ex/L Load map name. You must include the extension to this name.

name/P Patchfile name, including extension; patchfile is created by ENPAT.

name/S Object program name; you can omit the .SV extension.

Examples

1. *R*
   PATCH SYS/S SYS.LM/L IPB/P ↵
   3 APPLICABLE PATCH(ES)
   3 PATCH(ES) NEEDED TO BE INSTALLED
   R

   Install the patches entered in the ENPAT example. Note that PATCH describes the number of applicable patches, and the number it installs.

2. *R*
   PATCH/N MYPROG/S MYPROG.P1 ↵
   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

Command

Return to the next higher level program

Syntax

POP

Description

POP forces a return to the next higher level program after a user program has swapped in the CLI on level two or below. Programs can use the system call .EXEC to swap in the CLI.

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 in, by issuing .RTN (or .ERTN). POP effectively issues a .RTN from the CLI.

You cannot POP from the level zero CLI.

Example

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 the .EXEC system call.

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, and it continues from the point of interruption.

PRINT

Command

Print a file on the line printer

Syntax

PRINT filename *[..filename]*

Description

PRINT outputs the contents of ASCII (text) files at the line printer (device name $LPT). PRINT is the equivalent of a series of XFER/A filename $LPT commands.

To print more than one copy of a file, use the local numeric switch to indicate the number of copies you want. For example, PRINT RIDDLER/5 gives you 5 copies of the file RIDDLER.

The file(s) may reside on any device. For a paper tape file, if the system detects a parity error, it prints a backslash (\) in place of the bad character, displays the message *PARITY ERROR* at the console, and continues printing.

To print a binary file, see the FPRINT command description.

Local Switch

filename/n n specifies the number of copies of filename to print.

Examples

1. *R*
   PRINT FOO.SR DJ1:COM.SR BLIXEN.SR/2 ↵
   R

   Prints one copy each of the source files FOO.SR and DJ1:COM.SR, and prints two copies of BLIXEN.SR.

2. *R*
   PRINT ARREARS:<JAN,FEB,MAR>.BL ↵
   R

   Prints files JAN.BL, FEB.BL, and MAR.BL from directory ARREARS.

3. *R*
   PRINT DP1:MYFILE/3 ↵
   R

   Prints three copies of file MYFILE. MYFILE resides in primary partition DP1.

4. *R*
   PRINT MT0:2 ↵
   R

   Prints the contents of file 2 of the tape on drive MT0.

PUNCH

Command

Copy text file(s) on the paper tape punch

Syntax

PUNCH filename *[...filename]*

Description

Copies the contents of ASCII file(s) to paper tape on the paper tape punch ($PTP). PUNCH is the equivalent of a series of XFER/A filename $PTP commands. To punch a binary file, use BPUNCH.

The file(s) can reside on any device. If the system detects a parity error on paper tape, it punches a backslash (\) in place of the bad character, and displays the message *PARITY ERROR* on the console; punching continues.

No switches.

Example

```
R
PUNCH DP0:AVESTAN.SR HITTITE.SR ↵
R
```

Punches files AVESTAN.SR and HITTITE.SR on the paper tape punch.

RDOSSORT *Utility*

Invoke the RDOS Sort/Merge Program (RDOS)

Syntax

RDOSSORT infile *[infile...] [outfile/O]* key *[key...] [arguments]*

Description

RDOSSORT invokes the Sort/Merge Program, which can rearrange, delete, and combine disk or tape files.

The Sort function reads records from an input file (infile) and sorts them according to the key specifications, switches, and arguments you specify in the command line. If you want a sorted output file, infile must exist on disk.

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, infile is the name of a disk or tape file.

If you omit *outfile/O*, there will be no output file; this is useful if you want only a key file or listing.

The data in infile will be sorted according to the key you specify; up to eight keys are permitted. These keys make up the Control Word, which is 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

Input file collating order

Input file field delimiters

Output files

Work files

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, use the local /R switch to enter their byte length (in decimal).

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 using the local /S switch to specify the filename that describes your sequence.

Input File Field Delimiters

To specify a lower limit, use the /B local switch to name the file containing the lower limit for the major key field; for an upper limit, use the /U local switch. If you omit either delimiter, all records input are 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 in the b.f format (that is, b:f). If you enter no output field specifier, each record is output in the same form as you input it.

Output Files

Use the /L local switch to specify a listing file; the default is the terminal. Unless you specify a file for sorted keys with /K, no key file is produced.

Work Files

During a sort operation, RDOSSORT creates up to six work files in the current directory, and names them SORTWn.TP, where n is a number from 1 to 6. In many sorts, work file 1 is most active, followed by 4, 2, 5, 3, and 6. For 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.

You can run Sort/Merge from your terminal or under Batch, on a mapped machine, or in the background of an unmapped machine.

See the *RDOS/DOS Sort/Merge and Vertical Format Utilities* manual for more information.

Global Switches

/D Sorts records in descending ASCII order. (Default is ascending.)

/M Merges records. (The default operation is Sort.)

/N Does not list sort or merge statistics. (By default, Sort/Merge lists the statistics it produces.)

Local Switches

name/B Makes file name contain the lower limit field.

name/D Deletes input file name after sorting it. The system ignores this switch if you specify no output file.

name/K Writes sorted keys to file name.

name/L Lists sorted output on file or device name (overrides global /N).

name/O Identifies name as the sorted or merged output file.

n/R Defines decimal number n as the input record size in bytes. (Default is 80.)

name/S Specifies file name as the collating sequence. (Default is ascending ASCII.)

name/U Makes file name contain the upper limit field.

name/W Makes file name a user-defined work file. You can specify up to six work files.

RDOSSORT (continued)

Example

```
R
RDOSSORT MEMBERS DA DGSORT/O COLLAT/S 61.12 141.10 ^
$LPT/L 180/R 1:30 61:60 121:30 BOTLIMIT/B TOPLIMIT/U ⤸
R
```

The summary of statistics for this command might be:

RDOS Sort/Merge	*06:43:07 07/02/82*
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

Command

Release a directory or device from system initialization

Syntax

RELEASE { directory / device_name }

Description

RELEASE prohibits access to a directory or a device by closing it to input and output, and removing it from recognition by the operating system.

You can reinitialize devices and directories that you have released, with the exception of the disk master directory under RDOS and DOS systems. The master directory is a primary or secondary partition that contains system files. On RDOS and DOS systems, when you release the master directory, the system shuts down.

CAUTION: Always release a disk or diskette before physically removing it from its drive; otherwise its files may not be updated properly and the disk or diskette will be left in an inconsistent format.

When you release

- a directory, RELEASE closes and prohibits access to the directory, its files, and any directories subordinate to that directory. The commands INIT and DIR will reopen the directory for access.
- a tape drive, RELEASE rewinds the tape and prohibits access to the device. The INIT command will reopen the device for access.
- the master directory, RELEASE closes *all* initialized directories on the disk.

  In addition, under RDOS and DOS, releasing the master directory shuts down the operating system, and leaves the computer in a halted state. The CLI displays the message *MASTER DEVICE RELEASED*. To run the CLI after releasing the master directory, you must start up the system again.

  Under DG/RDOS, you can reinitialize the master directory with the INIT or the DIR command. To shut down the operating system under DG/RDOS, use the BYE macro.

The CLI cannot release the master directory while a foreground program is running. You must first type CTRL-F (CTRL-C CTRL-F under DG/RDOS) on the background console to end the foreground program. Type one CTRL-F (or CTRL-C CTRL-F) for each foreground program running.

CAUTION: Always release the master directory before powering off your computer.

Refer to the INIT and DIR command descriptions for information on initializing directories and devices.

No switches.

RELEASE (continued)

Examples

1. *R*
   RELEASE DP1 ⤸
   R

   Releases diskette DP1 so that it can be removed from the diskette drive.

2. *R*
   RELEASE MT0 ⤸
   R

   Rewinds the tape on drive MT0, and releases tape drive MT0.

3. *R*
   RELEASE %MDIR% ⤸
   MASTER DEVICE RELEASED

   Releases the master directory (whose name is contained in the CLI variable %MDIR%), and shuts down the operating system.

RENAME

Command

Change the name of a file or directory

Syntax

RENAME oldfile newfile *[...oldfile newfile]*

Description

oldfile specifies the current filename of the file or directory; append the .DR extension when renaming a directory.

newfile assigns a new filename to the file or directory; append the .DR extension when renaming a directory.

RENAME changes a file's or directory's filename. Renaming a file does not affect the file's contents, characteristics, or attributes.

You cannot rename any file that is protected with the P (permanent) attribute. You must first change the P attribute (CHATR command) before renaming such files.

The RDOS system utilities carry the P attribute; you could rename them if you change the attribute. Note that some utilities use a special COM.CM file that uses the original utility names; these utilities will not work if you rename the files.

You can rename directories; renaming a directory does not affect the contents of the directory. However, if you have macros with pathnames that use the old name, or link files in other directories that recognize the old name, they will no longer work. Be sure to include the .DR extension for directories in the RENAME command.

No switches.

Examples

1. ```
   R
   DELETE Q.SV ↵
   R
   RENAME QTEST.SV Q.SV ↵
   R
   ```

   This example deletes the old version of program Q.SV, and replaces it with the most recent version of the program by changing the name of QTEST.SV to Q.SV. If we did not delete Q.SV prior to the RENAME command, RENAME would not have worked and we would receive the error message FILE ALREADY EXISTS.

2. ```
   R
   RENAME DP0:A1.DR DP0:A.DR B1 B ↵
   R
   ```

   Renames directory A1 to A on DP0, rename file B1 to B in the current directory.

3. ```
   R
   XFER/A $TTI FN ↵
   LONGFILE CTRL-Z
   R
   RENAME T @FN@.01 T1 @FN@.02 T2 @FN@.03 ↵
   R
   ```

   This shows one way to assign the same name string to existing files. The @ signs specify the contents of file FN, which contains the string LONGFILE. The old names T, T1, and T2 become LONGFILE.01, LONGFILE.02, and LONGFILE.03.
   ```

REPLACE *Command*

Replaces overlays in an overlay file (RDOS and DG/RDOS only)

Syntax

REPLACE program_name

Description

REPLACE replaces overlays in an overlay file named program_name.OL, with the replacement overlays created in file program_name.OR via the OVLDR command.

REPLACE is the active sequel to the OVLDR command. Actual replacement occurs as soon as there are no outstanding overlay load requests.

No switches

Example

R
REPLACE ROOT ↵

Replaces 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

Command

Display the revision level of a .SV file

Syntax

REV program_name *[.SV]*

Description

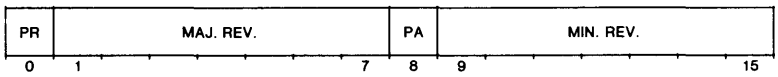
REV displays the revision level of a program (.SV file). program_name must have the S attribute. The system returns the major revision number followed by a period and a minor revision number.

Both major and minor revision levels can be in the range 0 through 99. Revision levels greater than 99 are displayed as 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 your .SV file is displayed as 00.00.

If bit 0 of the revision level word is set to 1, the code PR: is displayed after the revision level number to indicate that the save (.SV) file is a pre-release version. If bit 8 is set to 1, the word patched is appended to the revision level number display to indicate that the .SV file is a patched version.

The format of the revision level word is:



Certain compilers (e.g., FORT) assign their own revision level to the binary file, which is carried over to the .SV file.

No switches.

Example

R
REV CLI ↵
CLI.SV 07.40
R

The major revision level of this CLI is 7, and the minor revision level is 4.

RLDR

Utility

Loads relocatable binary (.RB) files to produce an executable program (.SV) file

Syntax

RLDR binary_file... [overlay_binary...] [library_file...]

NOTE: Square brackets ([]) in the RLDR command format are part of the command line; they are not notation conventions to set off options, as in other command formats.

Description

RLDR invokes the Relocatable Loader utility. RLDR takes assembled or compiled relocatable binary (.RB) files, assigns nonrelocatable addresses, and produces a program (.SV file) ready for execution. You can then execute program_name.SV by entering program_name to the CLI as a CLI command.

By default, RLDR uses the filename of the first binary file you specify in the command line to name the files it produces, including the program (.SV) file, overlay (.OL) file, and symbol table (.ST) file. The CLI searches for each binary file first with an .RB extension, then with no extension. You must include an extension, if any, on the library files you specify.

To create an overlay file for your program, specify the overlay binary files in square brackets on the command line. RLDR builds root binary files into the .SV file, and overlay binary files into the .OL file. During execution, the root binary files reside in memory all of the time, while the overlay binary files are called from the .OL file into their assigned memory nodes, according to the instructions in your program.

Each pair of square brackets ([ ]) in the RLDR command line defines a node in the .SV file and a segment in the .OL file. RLDR uses the node of the .SV file to reserve an area of memory to receive an overlay from the node's corresponding .OL file segment.

The individual overlay files reside independently in the .OL file, contiguous to one another in the order specified in the RLDR command line. Overlay files assigned to the same node (placed within the same pair of brackets) overwrite the node as the executing program loads them one by one (via the .OVLOD system call). For example, the command line

```
RLDR ABLE [A,B]
```

loads binary file ABLE as a root program, ABLE.SV. ABLE has one node for binary files A and B. The node in the disk file maintains itself in memory when ABLE executes; ABLE can load either A or B into this node without affecting the rest of ABLE.

If you want to be able to debug the program, use the global /D switch to include a debugger and symbol table in your .SV file. To include local user symbols, add the local /U switch. (You must have previously specified global /U to the assembler or compiler.)

RLDR can 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 .SV file on disk. You can print it with the global /L switch, or save it in a file with the local /L switch.

If a program runs 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 a high-level language (as opposed to assembly language), your compiler manual will tell you how to specify multiple channels and tasks.

NOTE: Mapped and unmapped systems have different system libraries (SYS.LB). A program loaded under one type of system will probably not execute under the other type of system. To load for a different kind of system, obtain the appropriate system library for the target system. You can ensure that RLDR searches the new library, and not the current library, during the load by using RLDR from a directory having links to RLDR and the target library.

To use RLDR, you need access to the following files:

RLDR.SV The RLDR program file.

RLDR.OL The RLDR program's overlay file.

SYS.LB The system library.

For more information about RLDR, see the *RDOS/DOS Assembly Language and Program Utilities* manual or the *RDOS System Reference* manual.

Global Switches

/A Produces an additional load map, with the symbols ordered alphabetically. (Use with the local /L switch.)

/B Uses short TCBs (task control blocks) in the save file (effective in unmapped NOVA multitask programs only).

/C Loads for compatibility with RTOS. Starts user NREL code at 440, and the .SV file at 0 (as with local /Z switch); does not search SYS.LB unless its name is included.

/D Includes the symbolic debugger and program symbol table with the program. RLDR does not write the program symbol table to the .SV 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). To load the interrupt-disable debugger, instead of the default symbolic debugger, include its name (xIDEB.RB) as an argument on the RLDR command line.

/E Displays error messages at the terminal 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 terminal.

/G When overlays refer to named common area, prints a warning message at each occurrence. By default, RLDR prints a warning message at the first occurrence only.

/H Prints numeric output in hexadecimal (radix 16). By default, numeric output is in octal radix.

/I Does not create UST, TCB, or other system tables; starts NREL code at octal location 445 and ZREL code at octal location 50. The .SV file cannot execute under any Data General operating system.

/K Stores RLDR's internal symbol table for this program in filename.ST. RLDR does not save this file unless you include /K.

/M Overrides the display of the load map and error messages at your terminal.

/N Overrides RLDR's search of the system library (SYS.LB). By default, RLDR searches the system library at the end of the command line to try to resolve undefined symbols.

/O Overrides the inclusion of the program symbol table as specified by the /D switch, but retains the debugger. (Use with global /D.)

/P Prints the starting NREL value of each .RB file as it loads.

/S Leaves the symbol table in high memory (you must include the global /D switch to produce the symbol table).

/X Use during SYSGEN to specify system overlays.

/Y Use during SYSGEN to specify system overlays (same as /X).

/Z Loads the .SV file to start execution at location 0. Use /Z only to load stand-alone programs that use page zero locations 0–15 (octal). You can then process the .SV file with the MKABS/Z command to produce an absolute binary file that executes in stand-alone mode, by way of the binary loader. Be careful with the global /Z switch; the resulting .SV file cannot execute under RDOS or DOS.

Local Switches

n/C Allots 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 Sends error messages to file name.

n/F Starts program's NREL address space at octal address n. Use to set NREL boundaries for a foreground program (unmapped RDOS only). (See the local /Z switch.)

n/K Allots n (octal) tasks to the program. This overrides any value in a .COMM TASK statement. By default, RLDR allots one task.

name/L Sends the load map to name, where name may be a file or a device such as the line printer ($LPT). This map lists symbols in numeric order. Without this switch, the load map will not be saved.

n/N Forces 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(octal) plus space for a User Status Table, TCBs, etc.; RLDR moves it upward as it loads each binary file.)

name/S Assigns name to the .SV file and the .OL file, overriding the RLDR default of using the name of the first binary file specified on the command line.

name/U Takes binary file name and includes its user symbols in the symbol table. Local symbols are those used exclusively within this binary file. This does not work unless you previously specified a local /U switch to the assembler.

[binaries...]/V
Loads binary files (delimited by brackets) as virtual overlay files (mapped RDOS only). You must specify all virtual overlay files before any other overlay files in the RLDR command line (e.g., RLDR ROOT [A,B]/V ROOT1 [C,D]).

n/Z Starts program's ZREL address space at octal address n. By default, ZREL code starts at location 50_8. Use to set ZREL boundaries for a foreground program (unmapped RDOS only). (See the local /F switch.)

Examples

1. *R*
   RLDR A B C DP4:D ⤸
   .
   .
   R

   Loads files A, B, and C from the current directory, and file D from directory DP4; produces an executable file called A.SV in the current directory. All messages go to the console. Neither the symbol table nor the load map is saved.

2. *R*
   RLDR/D A B C ⤸
   .
   .
   R

   Loads files A, B, and C with the symbolic debugger and a symbol table, to produce file A.SV.

3. *R*
   RLDR/E MYFILE MYFILE.LM/L ⤸
   .
   .
   R

   Loads MYFILE to produce MYFILE.SV; creates listing file MYFILE.LM and sends the load map and all messages to it. Error messages will appear at your terminal, as well (/E).

4. *R*
   RLDR MYPROG PROG1 MYLIB.LB $LPT/L 10/K 20/C ⤸
   .
   .
   R

   Loads MYPROG, PROG1, and library file MYLIB.LB to produce MYPROG.SV. All RLDR messages (plus the load map) print at the line printer ($LPT/L). MYPROG.SV is a multitask program; switch 10/K enables it to run up to 10 (octal) tasks, and switch 20/C enables it to use up to 20 (octal) channels.

5. *R*
   RLDR ROOT1 [A,B,C,D] ROOT2 ROOT3 [E,F,G,H] ROOT1.LM/L ⤸
   .
   .
   R

   Loads binary files ROOT1, ROOT2, and ROOT3 into ROOT1.SV, and creates overlay file ROOT1.OL with two segments. ROOT1.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). Switch ROOT1.LM/L sends the load map and console messages to file ROOT1.LM.

SAVE

Command

Rename a break file

Syntax

SAVE filename

Description

Rename file BREAK.SV (FBREAK.SV for the foreground) to filename.SV in the current directory. SAVE is commonly used to save the core image of a program interrupted by a CTRL-C break or by the $V debugger command. If filename already exists in the current directory, the system deletes it. You cannot precede filename with a directory pathname.

For more information on FBREAK.SV, see the *RDOS System Reference.*

No switches.

Example

```
R
DEB ALPHA ⤸
PP/LDA 2 0 LDA 2 @ 0 3 ⤸
$V
BREAK
R
SAVE ALPHA ⤸
```

Enters debugger to correct location PP. Exits from debugger and creates the break file (ESC V). Saves core image file as ALPHA.

SDAY

Command

Set the system calendar date

Syntax

SDAY mm dd year

Description

SDAY sets the system calendar to the date you specify. You can enter the year as either two or four digits (e.g., 84 or 1984). Use spaces or commas to separate each date argument.

No switches.

Example

R
SDAY 10 17 1984 ↵
R

Sets the system calendar date to October 17, 1984.

SEDIT

Utility

Analyze and edit a file on disk

Syntax

SEDIT filename

Description

Invokes 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 SEDIT does not find that file, SEDIT searches for filename. To specify an overlay file, include the .OL extension.

For more information on SEDIT, see the *RDOS/DOS Debugging Utilities* 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 nonprogram file.

/Z Start file at location 0.

Example

```
R
SEDIT MYPROG.SV ↵
SEDIT REVx.x
MYPROG% ↵
START + 10/ 106413 ↵
$; ↵
.START + 10/ SUB# 1 SNC
$Z
DONE
R
```

SEDIT finds and opens MYPROG.SV, proceeds with editing, examines a location, and returns to the CLI as directed by ESC Z (the ESC key echoes as a $).

You can find the current NMAX and ZMAX requirements for any program by typing the following SEDIT commands:

```
404/ nnnnnn ↵
401/ zzzzzz ↵
```

SEDIT returns values *nnnnnn* and *zzzzzz*, which are the User Status Table values for NMAX and ZMAX.

SMEM *Command*

Allocate memory for background and foreground (Mapped RDOS only)

Syntax

SMEM pagesize

Description

pagesize specifies the amount of memory, in pages, that SMEM allocates to the background. (One page equals 1024 words, or 2048 bytes.)

SMEM allocates the amount of memory you specify to the background, thus assigning the remainder of available user memory to the foreground. These allocations divide foreground from background, setting up separate foreground and background memory space.

When you start up RDOS, RDOS gives all available memory to the background.

You can issue SMEM only in a mapped system from the background CLI, while no foreground program is running.

No switches.

Examples

1. *R*
   GMEM ⤸
   BG: 341 FG: 0
   R
   SMEM 70 ⤸
   R
   GMEM ⤸
   BG: 70 FG: 271
   R

   In this example, the first GMEM command shows us the total amount of available memory prior to setting up memory for two grounds.

   The SMEM command allocates 70 pages of memory to the background, and allocates the remainder of available memory to the foreground. The final GMEM command confirms that the background is set up with 70 pages of memory, and the foreground is set up with the remaining 271 pages.

2. *R*
   SMEM 341 ⤸
   R
   GMEM ⤸
   BG: 341 FG: 0
   R

   This example dismantles the foreground-background setup by reallocating all available memory to the background.

SMEM (continued)

3. *R*
   SMEM 64 ↵
   R

   Allocates 64 1024-word pages of memory to the background, and the remaining available pages to the foreground.

4. *R*
   SMEM 148 ↵
   R

   Allocates 148 memory pages to the background, and all remaining user memory to the foreground.

SPDIS *Command*

Disable device spooling (RDOS and DG/RDOS only)

Syntax

SPDIS device *[...device]*

Description

Disables spooling on **device**. RDOS automatically spools data sent to any user device defined as spoolable, and to all the following devices: $LPT, $LPT1, $PTP, $PTP1, $TTP, $TTP1, and the teletypewriter versions of $TTO and $TTO1. You can disable spooling previously enabled (with SPEBL) on plotter devices ($PLT, $PLT1).

During a spool operation, RDOS stores in disk buffers data that is being sent to output devices. This frees the CPU for processing while the devices receive the data from the disk buffers.

If a spool operation requires more disk space than is available, the system itself will issue the equivalent of an SPDIS command, and will restart spooling when appropriate. This sequence is invisible to the user.

After you type the SPDIS command, the system will stop spooling data to the disabled device's disk buffer. Use SPEBL to reenable spooling to the device.

See also SPKILL for deleting a spool queue.

No switches.

Example

```
R
SPDIS $LPT ↵
R
```

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 *Command*

Enable device spooling (RDOS and DG/RDOS only)

Syntax

SPEBL device *[...device]*

Description

Re-enables spooling on a device for which spooling has been disabled by SPDIS. The device can be any user device defined as spoolable, or any of the following $DP0, $LPT, $LPT1, $PLT, $PLT1, $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.

See also the SPKILL command for deleting a spool queue.

No switches.

Example

```
R
SPEBL $LPT ↵
R
```

Re-enables spooling to the line printer.

```
R
SPEBL $PLT ↵
R
```

Enables spooling to the plotter.

SPEED

Utility

Invoke the SPEED Superedit text editor utility

Syntax

SPEED *[filename]*

Description

filename specifies the new or existing file to be edited.

The SPEED utility, also known as Superedit, allows you to create and modify random text files. The NOVA version of SPEED is NSPEED. You invoke it with the name NSPEED; it is otherwise identical to SPEED.

If the filename you specify in the command line does not already exist, SPEED creates it. You can invoke SPEED without a filename argument, and specify the file from within SPEED.

When you invoke SPEED, the utility generates an exclamation point (*!*) prompt; and you type in SPEED commands in response to this prompt. SPEED commands can add delete, or change information in a file, and store the modified file on disk.

In addition, SPEED contains features that allow use of multiple buffers, multiple I/O files, programming of macros, and use of numeric variables.

To end an editing session and return to the CLI, type H and press the ESC key twice.

SPEED is documented in the *RDOS/DOS Superedit Text Editor* manual. An introduction to SPEED can be found in the SPEED and EDIT Text Editors chapter of the *Introduction to RDOS*.

No switches.

Example

```
R
SPEED FILEX )
CREATING NEW FILE
!ITHIS IS THE STUFF.          The ! prompt indicates that SPEED
ESC ESC                       is ready to accept commands.
.
.                             You work on this file using
.                             SPEED commands.
!UE ESC ESC                   The UE command updates the file.
!H ESC ESC                    The H command terminates SPEED.
R
```

SPKILL *Command*

Stop spooling, delete the current spool queue (RDOS and DG/RDOS only)

Syntax

SPKILL device *[...device]*

Description

Cancels a spool operation by deleting the spool files queued to the device. The device may be any user device 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 that sends data to the device (e.g., PRINT MYFILE).

No switches.

Example

```
R
SPKILL $LPT ↓
R
```

Stops the spooling of data to the first line printer, and deletes the spool files.

STOD

Command

Set the system clock

Syntax

STOD hh mm ss

Description

STOD sets the system clock to the time of day you specify. The system clock is a 24-hour clock (i.e., you would specify 2 p.m. as 14 00 00). Use spaces or commas to separate the time arguments.

No switches.

Example

R
STOD 21 24 0 ⏎
R

Sets the system clock to 9:24 p.m.

SYSGEN *Utility*

Generate a new RDOS, DOS, or DG/RDOS system

Syntax

SYSGEN [*[newsysname/S] [dialogfile/V] [loadmapfile/L]*
[newsysname/S] [olddialogfile/A] [newdialogfile/V] [tuningfile/T] [loadmapfile/L]]

Description

Use the first format to generate a brand new RDOS, DOS, or DG/RDOS system, or a different version of an existing system.

In the first format, you choose 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 assigns a 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, is 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 that has the same name as a system with a tuning file.

SYSGEN saves the new version of the system under *newsysname.SV*. The name *olddialogfile* is the 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*. The name *newdialogfile* is your choice of a name for the new dialog file. The name *tuningfile* is the name of the SYSGEN tuning file (newsysname.TU); you must append the /T switch. The load map displays at 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 skips the RLDR step, and leaves the RLDR command line in (F)CLI.CM. You can invoke the RLDR phase once, before executing another SYSGEN, with the command @CLI.CM@

SYSGEN displays all questions at the terminal.

You will find the SYSGEN procedure documented in the manual *How to Load and Generate RDOS*. This manual also explains the RDOS tuning feature. To generate a DOS system, see *How to Load and Generate DOS*. For DG/RDOS, see *Using DG/RDOS on DESKTOP GENERATION™ Systems*.

Global Switch

/N Skips the usual RLDR step after the SYSGEN session, leaves the RLDR command line in (F)CLI.CM. You can invoke the RLDR phase once, before executing another SYSGEN, with the command @CLI.CM@

Local Switches

name/A Generates a system from dialog file name.

name/L Sends the load map to file name. Default is the terminal.

name/S Gives the finished system this name.

name/T Uses tuning file name (RDOS). SYSGEN will use the file to enter values that will result in a more efficient system, thus overriding the original settings, or supplementing your current SYSGEN responses.

name/V Saves the SYSGEN dialog for the system being generated in file name.

Examples

1. *R*
   SYSGEN SYS1.(SV/S SG/V LM/L) ↵

   Activates the system generation program; this allows you to generate a new operating system by answering a series of questions. The new system is named SYS1 and resides in files SYS1.SV and SYS1.OL. The dialog file is SYS1.SG. The /N switch is not specified, therefore SYS1 is built automatically; load errors and the load map go to disk file SYS.LM.

2. The following example assumes that RDOS system SYSXX was generated to include tuning, and has had tuning turned on for a while.

   R
   SYSGEN SYS2/S SYSXX.SG/A SYSXX.TU/T SYS2.SG/V $LPT/L ↵

   Generates 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 and SYS2.OL. The new dialog file is named SYS2.SG, the load map and load messages go to the line printer.

TPRINT

Command

Print the tuning file (RDOS only)

Syntax

TPRINT *[sysname]*

Description

Prints the tuning file *sysname.TU*. Note that you need not enter the *sysname* argument.

The tuning file is always in the directory that holds the current RDOS system; the TUON command creates the tuning file. The tuning file contains information on the use of 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 resource that had to be delayed because the resource was not immediately available.

If you specified a tuning overlay report at SYSGEN, you can include the global /O switch to print an overlay frequency report on each system overlay. The *RDOS System Reference Manual* describes the function of each overlay.

NOTE: The TPRINT command is meaningless unless tuning was requested at SYSGEN, and tuning was turned on (TUON) for some time during system operation.

Global Switches

/L Prints the tuning file at the line printer.

/O Prints an overlay frequency report.

Examples

1.
```
R
TPRINT/L SYS10 ↵
R
```

Prints the tuning file for the system SYS10 on the line printer.

2.
```
R
DIR %MDIR% ↵
R
LIST -.TU ↵
TOOLSYS.TU 543 D
R
TPRINT/L ↵
R
```

This sequence of commands makes the master directory (%MDIR) the current directory, lists any files having a .TU extension, and prints the file at the line printer.

TUOFF

Command

Stop recording in the tuning file (RDOS only)

Syntax

TUOFF

Description

TUOFF stops recording data concerning use of system buffer, cells, stacks, and overlays in the tuning file sysname.TU, where sysname is the name of the current operating system. TUOFF does not delete the contents of the tuning file.

No switches

Example

R
TUOFF ↵
R

Terminates recording in the tuning file.

TUON

Command

Start recording system tuning information in the tuning file (RDOS only)

Syntax

TUON

Description

TUON turns the tuning function on. Assuming that the current operating system was generated (during SYSGEN) with the tuning feature, TUON starts recording data concerning use of system stacks, cells, buffers, and overlays. For each of these, the report will include: the number of each in the system, total requests, total faults, and the percentage faulted.

If you specified an overlay report at SYSGEN, TUON will also start compiling an overlay frequency report.

TUON creates a tuning file named *sysname.TU*, where *sysname* is the name of the current operating system. TUON records data in this file whenever sysname is running, until you issue the TUOFF command. Type TUOFF before generating a new system from the current system's tuning file.

No switches.

Example

```
R
TUON ↓
R
LIST   -.TU ↓
TOOLSYS.TU 12 D
R
```

Initiates recording in the tuning file. In this example, the system name is TOOLSYS.

TYPE

Command

Display the contents of a file at the terminal

Syntax

TYPE filename *[...filename]*

Description

Displays the contents of ASCII (text) files at the terminal. This command is equivalent to a series of XFER/A filename $TTO commands.

You can suspend the display from a TYPE command by typing CTRL-S, and resume it by typing CTRL-Q. This aids in reading long files if your terminal is a CRT.

The files may reside on any device. For a paper tape file, when the system detects a parity error for a character, it types a backslash (\) and displays the message *PARITY ERROR* on the console, and continues the display.

To display the contents of a binary file, use the FPRINT command.

No switches.

Examples

1. *R*
   TYPE DP1:$$READ.ME ↵
   THIS DISKETTE CONTAINS UPDATED INFORMATION AND TESTING PROCEDURES FOR PROJECT GOLDFINGER. FEEL FREE TO USE IT.
   -JB-
   R

   Displays the contents of disk file $$READ.ME on diskette DP1, at your terminal.

2. *R*
   TYPE MT0:0 ↵
   DOCUMENT ARCHIVE RDOS CLI MANUAL REV 069-4000015-00

   LOAD TAPE FILES 1, 2, and 3.
   R

   Displays the contents of the file on tape MT0, file 0, at the terminal. In this example, the file contains information about the contents of the tape.

UNLINK

Command

Delete a link file

Syntax

UNLINK link_file *[...link_file]*

Description

Unlinks files by deleting one or more **link_files**. This command does not affect the resolution file.

CAUTION: Do not use the DELETE command to delete **link_files**. DELETE will delete the resolution file (unless the resolution file has been assigned the attribute P; see the CHATR and CHLAT commands) and the **link_file** will remain intact. If you delete the resolution file, you can restore the resolution file if you have kept a backup copy of the file on another disk or diskette, or on a tape.

See the LINK command for information about creating and using link files.

Global Switches

/C Requests confirmation of each deletion. The system repeats each specified link filename, and waits for you to confirm the deletion by typing a command line terminator. To prevent the removal of a link, press any key other than a command line terminator.

/L Produces a list of the deleted links on the line printer ($LPT). This switch overrides the /V switch.

/V Verifies each link deleted by displaying its filename at the console.

Example

```
R
UNLINK/C TEST.-↵
TEST.SR↵ *
TEST.RB↵ *
TEST.SV↵ *
R
```

Removes any link files whose names begin with TEST.

The /C switch causes UNLINK to request your confirmation before deleting each link entry from the current directory. When you confirm a deletion with a command line terminator, the system echoes an asterisk (*); when you prevent a deletion by typing any key except a command line terminator, the system echoes nothing.

VFU

Utility

Create, edit, load a format control file for producing forms on a data channel line printer (RDOS only)

Syntax

VFU *[filename]*

Description

VFU (Vertical Format Utility) creates and edits format control files for data channel line printers, and loads format control files into the printer's memory. This allows you to set up the line printer to produce forms according to your specifications.

You can specify three settings for the printer forms in each VFU file you create:

- tab stops
- form size (in number of vertical lines)
- and multiple line number/channel number pairs

When the data associated with a particular channel number finishes printing, the form skips to the next defined channel number and continues printing the data associated with that channel, and so on until a new form starts with channel 1.

VFU has four functions:

- It creates VFU files (always appending the .VF extension to your specified filename).
- It displays .VF files.
- It edits existing .VF files.
- It loads an existing .VF file into the printer's memory.

You need not enter the .VF extension to access VFU files.

VFU can enable and disable access by user programs to the printer's memory. The VFU utility program does not alter the state of spooling. It disables spooling while it is running, and restores the prior state of spooling (enabled or disabled) upon completion.

You specify each VFU function with a global switch. You must issue an enable or disable function alone on a separate command line; otherwise, you can enter multiple functions with multiple switches. In a multiple command line, the Create and Edit functions are processed first, followed by load functions, and then display functions.

Specific descriptions and examples for each of the related VFU functions are given below.

Create a VFU File (VFU/C)

To create a VFU file, 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 + + >

VFU (continued)

If you want standard tab stops (at column 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 ↵.

3. VFU CONTROL
WANT STANDARD (11 INCH)?
ENTER Y/N = = >

The answers you give to questions 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 that 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 that you will print using this file.

Next, VFU asks:

5. ENTER LINE NUMBER OR CR = = >

Specify a vertical line that you want to define as a channel, then type ↵. VFU asks:

6. ENTER CHANNEL NO. (1-12)= = >

Specify the channel number that 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 filename.VF and returns to the CLI.

Display a File (VFU/L for first line printer, VFU/V for 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 and c is the channel number. 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 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 associated with the channel you wish to set or clear, and ⟩. VFU then asks for the channel number to assign to the line number location.

4. ENTER CHANNEL NO. (1-12) = = >

Enter the channel number that you want to associate with the line number (for Set) or that is already associated with the line number (for Clear). Now, VFU asks:

5. ENTER SET(S) OR CLEAR (C) = = >

To set a new channel, type 2; to clear an existing channel setting, type C. VFU then repeats questions 3, 4, and 5 until you respond with ⟩ to 3.

VFU now updates the existing VFU file 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 filename; for the second data channel printer, type VFU/S filename, VFU then displays a prompt message. When you strike a key and the printer is ready, VFU kills spooling and transfers (XFER) the VFU file into the printer's memory. You can then print files on the forms using 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).

Global Switches

/A Enables access. Allows user programs to access the line printer's memory. Use alone, without a filename argument or additional switches.

/C Creates a VFU file (*filename.VF*). The utility prompts you for format specifications, one at a time.

/D Disables access. Disables user program access to line printer memory. Use alone, without a filename argument or additional switches.

/E Edits an existing VFU file. The utility prompts you to change format specifications.

/L Displays the format settings of an existing VFU file on the line printer ($LPT).

/S Directs output to the secondary line printer ($LPT1). Displays settings when used with the /L switch; loads settings into $LPT1 when used with the /X switch.

/V Displays the format settings of an existing VFU file at the terminal.

/X Loads the VFU file into the primary memory of $LPT (or $LPT1 when used with /S). This is the default; if you invoke the VFU utility without any switches, VFU loads the specified file into $LPT memory.

Local Switches

filename/L Outputs the format settings of the VFU file to filename.

VFU (continued)

Example

```
R
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 creates VFU file PAYROLL1.VF. The next sequence shows a combined display/edit and loading of this file.

```
R
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
TABS STOPS
3, 8, 16, 28, 50
VFU CHANNELS:
1-1, 402, 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

Command

Transfer the contents of a file to another file; Transfer input from a terminal to a disk file

Syntax

```
XFER sourcefile destinationfile
```

Description

sourcefile specifies the filename or device name (but not a directory) *from* which you transfer information

destinationfile specifies the filename or device name (but not a directory) *to* which you transfer information

Both of these arguments can be in a pathname.

XFER transfers information from a file on any device to another file on any device. Note that in RDOS, devices are logically constructed as files, and are referred to by their device names. Table 5-1 lists RDOS device names.

In RDOS and DG/RDOS, if you omit local switches and the destination file does not already exist, XFER creates the destination file as a sequential file.

In DOS, if you omit local switches, XFER organizes the the destination file randomly. (DOS does not support sequential files.)

When the destination file is an executable program (.SV) file, be sure to include the /R local switch, as RDOS executes only random files. Also, a new destination .SV file will not be assigned the S attribute. Be sure to use a CHATR filename +S command on the destination file.

When you transfer information into an existing destination file, the file retains its original characteristics.

During a transfer, the system will detect parity errors in all ASCII source files, and in binary tape files. On a parity error, RDOS display the message PARITY ERROR. For a paper tape file, RDOS displays a backslash (\) for each bad character. For magnetic or cassette tape files, RDOS aborts the command when it detects a parity error.

You can use XFER to copy text directly into a file from your terminal in the form:

```
XFER/A $TTI filename
```

where **$TTI** is the device name for your keyboard. Include the /B switch if filename is an existing file. All of the information you subsequently type at your keyboard becomes part of the destination file. To terminate the transfer and return to the CLI, type a CTRL-Z.

If you use XFER to put information on magnetic tape, then you must use XFER to retrieve the information. In other words, you can't use LOAD or another utility to read magnetic tape files written with XFER.

Global Switches

/A Specifies an ASCII (text) transfer. XFER transfers ASCII characters, line by line, taking appropriate read/write action, such as inserting line feeds and carriage returns, if that applies to the destination file.

/B Appends the source file to the end of the existing destination file. Use this switch when transferring data to a disk file or tape file that already exists; otherwise, the XFER command will abort and you will receive the error message *FILE ALREADY EXISTS.*

Local Switches

destinationfile/C Organizes the destination file contiguously. (Both the source and destination files must be disk files.)

destinationfile/R Organizes the destination file randomly. (Both the source and destination files must be disk files.) Be sure to use this switch to XFER information to a .SV file.

Examples

1. *R*
   XFER/A/B $TTI OLDFILE ↵

   We're appending this stuff to oldfile.
   RDOS transfers everything we type in until we press the CTRL key and a Z simultaneously.
   <CTRL-Z>
   R

   Appends information typed at the keyboard to file OLDFILE. OLDFILE is an existing file, and we included the /B switch to append the information to the file. We terminated the transfer with a CTRL-Z. OLDFILE retains its original characteristics and attributes.

2. *R*
   XFER/A DP4:OLDNOTES NEWFILE/R ↵
   R

   Creates a random file, NEWFILE, in the current directory, and transfers the information from the file OLDNOTES in directory DP4 to the new file.

3. *R*
   XFER MYPROG.SV MT0:0 ↵
   R

   Transfers the file MYPROG.SV to tape file 0 on tape MT0.

4. *R*
   XFER MT0:1 NEWPROG.SV/R ↵
   R
   CHATR NEWPROG.SV +S ↵
   R

   Creates a random file, NEWPROG.SV, from tape file 1 on tape MT0. The CHATR command assigns the new file an S (executable) attribute.

5. *R*
   XFER/A/B QTY:7 DP4:MUXNOTES ↵
   R

   Appends the text input from QTY line 7 to disk file MUXNOTES in directory DP4. The user on QTY:7 can terminate the transfer with CTRL-Z.

XFER (continued)

6. *R*
   XFER $PTR $PTP ⤸
   R

   Copies the tape in the paper tape reader to the paper tape punch.

7. *R*
   XFER $PTR Q ⤸
   LOAD $PTR, STRIKE ANY KEY
   R

   Creates disk file Q, waits for the operator to load the paper tape reader and strike a key, then copies the tape in the reader to disk file Q.

Appendix A
CLI Command Summary

Command	Function
ALGOL filename...	Compile an ALGOL source file (RDOS).
APPEND outputfilename inputfilename *[... inputfilename]*	Combine two or more files.
ASM filename...	Assemble a source file, producing an .RB file.
BASIC	Invoke the BASIC interpreter.
BATCH *[jobfile(s) ...]* *[outfile/O] [logfile/G]*	Invoke the Batch monitor, to execute Batch job streams (RDOS).
BOOT {disk / *[directory:]*sysname}	Bootstrap a system from disk.
BPUNCH filename...	Punch a binary file.
BUILD outputfilename *[... inputfilename]*	Build a file of filenames.
CCONT filename blockcount	Create a contiguous file of the specified number of blocks.
CDIR directoryname	Create an RDOS subdirectory or DOS directory.
CHAIN program_name	Load and run a program in place of the CLI.
CHATR filename *[sign]* attributes *[... filename [sign] attributes]*	Change a file's attributes.
CHLAT filename *[sign]* attributes *[... filename [sign] attributes]*	Change a file's link access attributes.
CLEAR *[filename ...]*	Set file or device use count to 0.
CLG filename...	Compile, load, and execute a FORTRAN IV source file.
COPY sourcediskette destinationdiskette	Copy diskette$_1$ to diskette$_2$ (DOS).
CPART partion_name blockcount	Create a secondary partition (RDOS, DG/RDOS).
CRAND filename	Create a random file.
CREATE filename	Create a sequential file (RDOS, DG/RDOS).
CSSORT infile outfile key arguments	Run the CSSORT Sort/Merge program.
DDUMP *[devname]*	Fast dump a disk, partition or directory to diskettes.

(continues)

Command	Function
DEB program_name	Debug a program.
DELETE filename ...	Delete a file or directory.
DIR directory_name	Change the current directory.
DISK	Display the number of blocks used, free, and free contiguous in the current.
DLOAD *[devname]*	Load diskettes that were fast dumped (DDUMP).
DO macroname *[argument...]*	Execute the CLI macro, replacing dummy arguments with arguments specified in the DO command line.
DUMP *[[destination:]dumpfilename] [filename] [argument]*	Copy the contents of the current directory to *dumpfilename.*
EDIT *[filename]*	Invoke the Text Editor.
ENDLOG *[password]*	Stop recording in the LOG.CM file.
ENPAT patchfile	Encode patches in patchfile. (Install patches with PATCH.)
EQUIV newname oldname	Temporarily rename a disk or magnetic tape (RDOS).
EXFG programname	Execute program in the foreground (RDOS).
FCOPY *[source destination]*	Duplicate a diskette or copy a file (DG/RDOS).
FDUMP MTn:f	Fast dump the current directory to magnetic tape file.
FGND	Describe the foregound program status.
FILCOM $filename_1$ $filename_2$ *[outputfile/L]*	Compare the contents of two files.
FLOAD MTn:f	Fast load a fast dumped (FDUMP) file into the current directory.
FORT filename...	Compile FORTRAN IV source program.
FORTRAN filename...	Compile FORTRAN 5 source program.
FPRINT filename	Print a file in octal, or other specified format.
GDIR	Display the current directory name.
GMEM	Display background/foreground memory areas (mapped RDOS).
GSYS	Display the current system name.
GTOD	Display the current system time.

(continued)

Command	Function
ICOBOL source_file *[list_file] [error_file]*	Compile an ICOBOL source program.
ICX *[program_name/x]*	Invoke the Interactive Cobol Runtime Environment.
IMOVE devname *[filename ...]*	Dump and load files between disk, diskette, or magnetic tape.
INIT { directory_name / device_name }	Initialize a directory or device for I/O.
LDIF	Display the last current directory name.
LFE	Create or edit .RB library files.
LINK link_file *[resolution_file]*	Create a link entry to resolution filename.
LIST *[[directory_name:] file ...]*	List file information.
LOAD *[source:]*dumpfilename... *[filename ...]*	Reload dumped (DUMP) files.
LOG *[password] [directory/O]*	Start recording in the log file LOG.CM.
MAC filename...	Assemble a source file into a relocatable binary (.RB) file with the Macroassembler.
MCABOOT MCAT:n *[arguments ...]*	Transmit a system over an MCA line (RDOS).
MDIR	Display the master name.
MEDIT *[filename]*	Invoke the Multiuser Text Editor (RDOS).
MESSAGE *["]*textstring*["] [...["]textstring["]]*	Display text message.
MKABS program_name absolute_filename	Make an absolute file from a save file.
MKSAVE absolute_filename program_name	Make a save file from an absolute file.
MOVE destination_dir *[... filename]*	Copy filename to destination_dir.
NSPEED *[filename]*	Invoke the NOVA SPEED text editor.
OEDIT *filename*	Edit disk file locations with the Octal Editor.

(continued)

Command	Function
OVLDR prog_name old-overlay/N new-overlay(s)	Create an overlay replacement file.
PATCH *[program_name/S] [patch_file/P] [loadmap_filename/L]*	Install patchfile created by ENPAT.
POP	Return to the program on the next higher level.
PRINT filename...	Print a file on the line printer.
PUNCH filename...	Punch an ASCII file on paper tape punch.
RDOSSORT infile *[outfile/O]* key *[arguments]*	Sort a file or merge files with the Sort/Merge program (RDOS).
RELEASE {directory / device_name}	Release a directory or device from system initialization.
RENAME oldfile newfile	Rename a file.
REPLACE program_name	Replace overlays in an overlay file (RDOS).
REV program_name*[.SV]*	Display the revision level of program filename.
RLDR binary_file *[overlay_binary] [library-file]*	Process relocatable binary files to form an executable program.
SAVE filename	Rename a breakfile.
SDAY mm dd yy	Set the system calendar.
SEDIT filename	Edit disk file locations with the Symbolic Editor.
SMEM pagesize	Set the foreground-background memory areas (mapped RDOS).
SPDIS device	Disable spooling to device (RDOS).
SPEBL device	Enable spooling to device (RDOS).
SPEED *[filename]*	Invoke the ECLIPSE SPEED text editor.
SPKILL device	Delete data spooled to device (RDOS).
STOD hh mm ss	Set the system clock.
SYSGEN *[newsysname/S] [dialogfile/V] [loadmapfile/L]*	Generate a new operating system.
TPRINT *[sysname]*	Print the tuning file (RDOS).
TUOFF	Stop recording in the tuning file (RDOS).
TUON	Start recording in the tuning file (RDOS).
TYPE filename ...	Type a file at the terminal.
UNLINK link_file	Remove a link entry.
VFU *[filename]*	Create or load a VFU file for a data channel line printer (RDOS).
XFER sourcefile destinationfile	Copy the contents of one file to another file.

(concluded)

End of Appendix

Appendix B
Error Messages

Message	Source, Possible Causes, and Action
ATTEMPT TO CREATE A ZERO LENGTH CONTIGUOUS FILE	You've neglected to specify blockcount to the CCONT command.
ATTEMPT TO RELEASE AN OPEN DEVICE	From the operating system. You've attempted to release a magnetic tape unit while a file on the tape is open.
ATTEMPT TO RESTORE A NON-EXISTENT IMAGE	You issued the POP command from a level 0 CLI. The system has no program to restore.
ATTEMPT TO WRITE AN EXISTENT FILE	From the operating system. The program tried to write to a file that already exists.
ATTRIBUTE PROTECTED	You can't change the attributes, rename, or delete the file.
BLANK TAPE	From the operating system. Probably the tape is new and has not been initialized with INIT/F.
CHECKSUM ERROR	From the operating system or a support program. The drive hardware couldn't read the diskette or tape. Retry. If the error recurs, try another diskette or tape or a different drive. For a tape drive, sometimes cleaning the tape or cleaning the unit's read-write heads will help.
DEVICE ALREADY IN SYSTEM	From the CLI. You tried to initialize (INIT) a directory that is already initialized. The system can't initialize it again.
DEVICE NOT IN SYSTEM	From the CLI. You tried to access a tape or disk unit without initializing it. For tape, type **INIT MTn**; for a disk(ette) based directory, you can use either the INIT or DIR command.
DEVICE PREVIOUSLY OPENED	From the operating system. You've tried to open a magnetic tape unit that is already open.
DEVICE TIMEOUT: device	From the CLI. The system has tried for 20 seconds to access the device, but it cannot do so. For a diskette, this probably means that the diskette is not inserted in the unit properly or not inserted at all. Check the diskette in the drive.
DIRECT I/O ACCESS ONLY	From the CLI. A program tried to perform nondirect-block I/O on a file that requires direct-block I/O.

Message	Source, Possible Causes, and Action
DIRECTORY DEPTH EXCEEDED	From the CLI. You (or a LOAD command) tried to create a subdirectory within a subdirectory, or a secondary partition within a secondary partition. Change directories or commands as appropriate, and retry.
DIRECTORY IN USE	From the CLI. It thinks that the directory you tried to delete or rename is in use. First, try releasing the directory with the RELEASE command. If the message recurs, the foreground program may be using the directory. If so, you might not want to delete or rename the directory. If you do want to delete or rename it, have the foreground release it, then retry. If the message recurs again, there may be a file open in the directory. For example, you can't release the directory from which you started logging (LOG command) until you type ENDLOG. Check for open files with the LIST/U/S command. Any use count other than 0 indicates a file in use. If there are any use counts above 0 (aside from SYS.DR), try the following commands to clear the use counts to 0. DIR dir_name CLEAR/A/V/D; CLEAR LOG.CM *CLEARED SYS.DR* *R* RELEASE dir_name *R* Then try again to delete or rename the directory. If the foreground is running, the system won't let you use the CLEAR command. So, if the foreground is running, you'll need to wait until it's terminated, or terminate it yourself, before clearing use counts.
DIRECTORY NOT INITIALIZED:	You've tried to access an uninitialized directory. Initialize with the INIT command then retry the command.
DIRECTORY SHARED	From the CLI. After you release a directory (RELEASE), this message reminds you that the other ground has the directory initialized. The message is informational only. Take no action.
DIRECTORY SIZE INSUFFICIENT	You (or a program) tried to create a secondary partition smaller than 48 blocks long. Repeat with an argument of 48 or more.

Message	Source, Possible Causes, and Action
DISK FORMAT ERROR	From the CLI. The disk(ette) is not in RDOS, DOS, or DG/RDOS directory format. It has been hardware formatted, but not software formatted (it hasn't had a DKINIT FULL command run on it and INIT/F typed to it). The next step really depends on what you want to do. The problem may be the wrong diskette or disk (check the label). Or, if you want to use INIT or DIR, you must run a DKINIT FULL and INIT/F command to make the diskette into a directory.
DISK SPACE EXHAUSTED: file	From the CLI. This message means that your command cannot be completed because the current disk (or secondary partition) is full. If file is a secondary partition or nonmaster disk, you will probably want to delete some files to make it usable; use the LIST date/B command to identify old files. If file is the master directory, you must free some space for system operations to continue. Check for files to delete with the LIST date/B command. With a hard disk, you must plan some procedure for backing up and deleting many files — there should be a minimum of 500–1000 blocks free on a hard disk. If this is a multiuser system, ask users which files they don't need, and delete them.
END OF FILE	From the CLI. The system hit an end of file when it tried to execute your command. Often, with tape, this means you specified a nonexistent file number. Try a lower number, for example, MT0:0.
ERROR message	Try to find the message in this table.

EXPANDED ERROR TEXT NOT AVAILABLE	From SPEED text editor. SPEED needs two files for its error message text: CLI.ER and SPEED.ER. Either execute SPEED from the directory in which these files reside, or create links to the files.
	
	If you're running SPEED from a nonmaster directory, and CLI.ER and SPEED.ER reside in the master directory, create needed links by typing
	
	`LINK CLI.ER %MDIR%:CLI.ER`
	`LINK SPEED.ER %MDIR%:SPEED.ER`

FG TERM	From the background CLI. The foreground program has terminated. Either you did it from the system console with CTRL-C CTRL-F, or the program terminated normally, or the program hit a fatal error condition.
FILE ALREADY EXISTS	From the CLI. You tried to create a file whose name already exists in the pertinent directory. Or, you tried to use the MOVE, IMOVE, LOAD, or XFER commands on such a file without appropriate switches.

Message	Source, Possible Causes, and Action
FILE ATTRIBUTE PROTECTED	You tried to change the attributes of a file whose attributes were fixed with the .CHATR system call. The attributes can't be changed.
FILE DATA ERROR: file	From the CLI. There's an inconsistency in the file you're trying to access. It displays this error on most read or write errors to disk files. If file is a diskette name, this message probably means that the diskette you tried to initialize (INIT or DIR) has not been hardware formatted. (The message is SYS.DR ERROR if the diskette has been hardware formatted but not software formatted.) If file is SYS.DR or MAP.DR, this means a system directory has inconsistent information in it. Probably, an INIT/F command is needed on the disk(ette) to fix things. You can try to back up existing files before doing the INIT/F, but the backup may itself be inconsistent.

FILE DOES NOT EXIST: name	From the CLI. The system can't find the file in the current or specified directory. The problem may be one of the following:
	
	· You made a typing error.
	
	· The file exists, but you forgot a needed extension; for example, save files have the .SV extension, which you must type (except to execute). CLI macro files have the extension .MC; and directory names end in .DR (for IMOVE commands or DUMP commands).
	
	· The file exists, but in another directory, and there is no link to the file in its parent directory. This often happens after you create a new directory, move to it with the DIR command, and try to execute a program (like SPEED.SV). Use the LINK command to create a link, preferably with the same name, to the directory and file. The link to a save file must have the .SV extension — it won't work otherwise. For example
	
	`LINK SPEED.SV %MDIR%:SPEED.SV`
	`LINK SPEED.ER %MDIR%:SPEED.ER`
	
	Generally, system files that many people use are installed in a specific directory (often the master directory). A pathname (directory:file), link entry, or macro is needed to access the file.
	
	A link allows you to access the file from the directory at almost no cost in disk space.
	
	· If *name* is DJ0 or another disk name, this means that the disk has not been fully initialized with INIT/F. Verify that this is the right disk(ette). Then, if you still want to access it as a directory, type INIT/F to it, and confirm with Y.
	
	· Lastly, the file really doesn't exist.

Message	Source, Possible Causes, and Action
FILE IN USE: file	From the CLI. You tried to read, rename, or delete a file that the system thinks is in use. If a file is open at abnormal shutdown, its use count remains nonzero when the operating system comes up again; RDOS thus assumes that someone is still using the file. You can check a file's use count by typing LIST/U filename. To zero the use count, type CLEAR/A/V/D. If this doesn't help, type CLEAR/V filename. Note that CLEAR does not fix any data inconsistency that may have developed because the file was left open.
FILE NOT FOUND: xxxx.SV	From the bootstrap program, at startup. It can't find the file you specified. Programs you can start from the Filename? question include the operating system (e.g. DGRDOS.SV), or a stand alone utility (e.g. DKINIT.SV, MBOOT.SV, YBURST.SV).
FILE READ PROTECTED	From the CLI. You can't read the file because it has the R (read-protect) attribute. If you need to read it, type CHATR filename −R, then retry the command.
FILE SPACE EXHAUSTED	From the CLI. All tape has been used; yet more disk-based material remains to be copied. The backup to this tape file is incomplete; retype the command to a tape with more space left. For disk, see message *DISK SPACE EXHAUSTED.*
FILE WRITE PROTECTED	From a text editor or CLI. If a disk file, you can't change the file because it has the W (write-protect) attribute. If you need to change it, type CHATR filename −W. If it is a tape file, insert a write ring. If it's a diskette file, make sure the diskette is write enabled.
FILES MUST EXIST IN THE SAME DIRECTORY	From the CLI. Your command (probably RENAME) requires both files to be in the same directory.
FOREGROUND ALREADY RUNNING	From the CLI. While a foreground program is running, you can't: use SMEM to change memory allotments; CLEAR file use counts to 0; execute a program in the foreground; or shut down (BYE). Before you can do any of these things, you must terminate the foreground program with CTRL-C CTRL-F.
ILLEGAL ARGUMENT	From the CLI. You specified an illegal argument.
ILLEGAL ATTRIBUTE	From the CLI. You specified an illegal attribute (with CHATR). See the CHATR command for more detail.
ILLEGAL BLOCK TYPE	From the CLI. You tried to LOAD a file that was copied with XFER or to use XFER on a file that was copied with DUMP. Use the appropriate command.

Message	Source, Possible Causes, and Action
ILLEGAL CHARACTER	From system, at startup, when you're trying to type the date and time. Turn computer power off and on, and try again. On DG/RDOS systems, try the numeric keypad, to the right of the main keypad, to type numbers.
ILLEGAL DIRECTORY NAME	From the CLI. The directory name you specified was illegal. Legal filename characters are a–z, A–Z, 0–9, and $.
ILLEGAL FILE NAME	From the CLI. The filename you specified was illegal. Legal filename characters are a–z, A–Z, 0–9, and $.
ILLEGAL INDIRECT FILENAME	From the CLI. The indirect filename you specified was illegal. Retype the command, using the form @filename@ (legal filename characters are A–Z, 0–9, and $).
ILLEGAL NUMERIC ARGUMENT	From the CLI. The problem may be that there is a nonnumeric character in a numeric argument; the numeric argument is too large; or the radix is wrong.
ILLEGAL PARTITION VALUE	From the CLI, after your SMEM command. You tried to: allocate more memory to the background than is available to both grounds, or allocate too little memory for the background CLI (less than 20 pages). Retry the SMEM command.
ILLEGAL TEXT ARGUMENT	From the CLI. You omitted a matching quotation mark (") in a MESSAGE command.
ILLEGAL VARIABLE	From the CLI. The variable you typed doesn't exist, or is illegal. Retype the command, using the form %variable%.
INSUFFICIENT CONTIGUOUS BLOCKS	From the CLI. There are not enough contiguous free blocks on the disk to hold the contiguous file you're trying to LOAD (MOVE) onto it. The disk is probably nearly full; take action described under message *DISK SPACE EXHAUSTED.*
INSUFFICIENT MEMORY TO EXECUTE PROGRAM	From the CLI. There isn't enough memory available in the foreground to execute this program. Give the foreground more memory. Use GMEM to get memory allotment (n). If the foreground (FG) doesn't have at least 16 pages, give it 16 pages (SMEM 16), and retry the EXFG command. Keep giving the foreground more memory, in increments of 2 pages, until the EXFG command works or you get an error message. Another option is to check the product Release Notice for the amount of memory required.

Message	Source, Possible Causes, and Action
INSUFFICIENT NUMBER OF CONTIGUOUS FREE BLOCKS	There isn't enough contiguous space for you to create the file. Take action described under DISK SPACE EXHAUSTED.
INT	From the CLI. You interrupted the CLI command with CTRL-C CTRL-A.
INVALID BAD BLOCK TABLE	From the CLI. The disk(ette) cannot be initialized in its current state. Run a DKINIT PARTIAL (if this fails, DKINIT FULL) command on the diskette.
INVALID TIME OR DATE	From the CLI. You tried to specify an illegal time or date, or used an illegal character. Type GTOD for an example of the correct format (you can use spaces as separators).
LINE TOO LONG	From the CLI. The command line and system limit for line-oriented text is 132 characters; you typed more than this number. Retype the command.
LINK ACCESS NOT ALLOWED	From the CLI. The file's link attributes (CHLAT command) prevent access by way of a link entry.
LINK DEPTH EXCEEDED	From the CLI. This probably means the file was linked to itself. Remove the link (UNLINK linkname) and recreate it specifying the correct resolution file pathname.
LOG FILE ERROR	From the CLI. Logging was on (LOG), but the CLI could not write to the log file. This may mean you released the directory where the log file was started. If so, move to the directory with the DIR command, restart logging, and continue. The original log file has been deleted.
MAP.DR ERROR	From the CLI. The space allocation directory has inconsistent information. Probably, an INIT/F is needed to fix things. You can try to back up existing files before doing the INIT/F, but the backup may itself be inconsistent.

Message	Source, Possible Causes, and Action
NO DEBUG ADDRESS	From the CLI. You tried to debug (DEB command) a program that does not include a debugger (RLDR/D switch).
NO DEFAULT DEVICE	From the CLI. No directory is current. Set a new current directory with the DIR command.
NO FILES MATCH SPECIFIER	From the CLI. There are no files that match your filename template (characters - or *).
NO MORE DCBS	You (or another user or program) tried to initialize a directory (via INIT or DIR) — but the number of directories in use has reached the maximum allowed (as set in SYSGEN or CONFIG). For the INIT or DIR command to work, an initialized directory must be released (RELEASE command). If you get this error message often, you may want to increase the number of directories accessible at one time with SYSGEN or CONFIG. Or, you may want to reduce the number of directories you use in your system.
NO ROOM FOR UFTS	From the CLI. There are not enough channels configured for the program ground. The CLI requires 16 channels and other applications may require more. Run SYSGEN or CONFIG to specify more channels. This message can also mean that a program you built doesn't have enough channels reserved (with RLDR /C switch). This message also can result when you try to execute rather than boot a stand-alone program from the CLI.
NO SOURCE FILE SPECIFIED	From a utility like the MAC assembler. You omitted a source filename from the command line.
NO SUCH DIRECTORY: file	The operating system can't find the directory. This may mean that the directory, or its parent directory, isn't initialized. You can check directory names within the current directory by typing LIST -.DR. Use the INIT or DIR command to initialize the directory, then try again.
NOT A LINK ENTRY	You tried to UNLINK a nonlink file. To remove nonlink files, use DELETE.
NOT A SAVE FILE	You can't execute any program that's not a save (.SV) file. It must also be a randomly organized file and have the S attribute. Check with LIST and give the S attribute (CHATR filename + S) if needed. If the file isn't randomly organized (R), copy it to a random file by typing XFER filename newname/R
NOT ENOUGH ARGUMENTS	From the CLI. More arguments are needed; see the entry in the Command Dictionary for needed arguments.
OUT OF TCB'S	From the CLI. Your user program needs to have more tasks defined. Run RLDR on it again and specify more tasks.

Message	Source, Possible Causes, and Action
PARITY ERROR	From the CLI. For action, see message *CHECKSUM ERROR.*
PERMANENT FILE: name	From the CLI. You tried to delete a permanent file (P attribute). This can happen after a DELETE command or after a MOVE, IMOVE or LOAD command that specifies deletion (the /R or /O switch). If you really want to delete the file, remove its P characteristic (CHATR name −P). (Some system files are attribute protected, which means you can't remove a P attribute.)
QTY ERROR	From the CLI. This means a USAM line error — probably a simultaneous read or write to the same line.
RDOS ERROR: n	From a system utility. It could not interpret the error code returned. This can happen if the program lacks disk space to create a needed file. It can also indicate a revision clash (incompatible revisions of the utility and the operating system).
SPOOL FILES ACTIVE	From the CLI, when you try to shutdown or boot. There is material in a spool queue, for example the queue for the line printer. Either wait for the queue to empty, for example for printing to stop, or kill the spool queue by typing, for example SPKILL $LPT then shut down.
STACK OVERFLOW	From the CLI. There isn't enough memory available to execute your command. For some commands, the CLI needs a minimum of 21K words. Check with GMEM, give the CLI at least 21K words with SMEM, and try again. (If the foreground is already running, you'll need to wait until it ends to change memory with SMEM.)
SYS.DR ERROR: file	From the CLI. There's an inconsistency in the system file directory. If file is a diskette or disk name, this message probably means that the one you tried to initialize (INIT or DIR) has been hardware formatted, but not software formatted, or that it has been written to by a program like IMOVE or a different operating system.
TEXT ARGUMENT TOO LONG	From the CLI. You typed too many characters (more than 72) between quotation marks. Retype with a shorter text string.
TOO MANY ARGUMENTS	From the CLI. You typed too many arguments (maybe you accidentally inserted an extra space).
UNIT IMPROPERLY SELECTED	From the CLI. You tried to access a magnetic tape drive, but the drive is not turned on, or not on line.
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 computer.

End of Appendix

Appendix C
ASCII Character Set

DECIMAL	OCTAL	HEX	KEY SYMBOL	MNEMONIC
0	000	00	↑@	NUL
1	001	01	↑A	SOH
2	002	02	↑B	STX
3	003	03	↑C	ETX
4	004	04	↑D	EOT
5	005	05	↑E	ENQ
6	006	06	↑F	ACK
7	007	07	↑G	BEL
8	010	08	↑H	BS (BACKSPACE)
9	011	09	↑I	TAB
10	012	0A	↑J	NEW LINE
11	013	0B	↑K	VT (VERT. TAB)
12	014	0C	↑L	FORM FEED
13	015	0D	↑M	CARRIAGE RETURN
14	016	0E	↑N	SO
15	017	0F	↑O	SI
16	020	10	↑P	DLE
17	021	11	↑Q	DC1
18	022	12	↑R	DC2
19	023	13	↑S	DC3
20	024	14	↑T	DC4
21	025	15	↑U	NAK
22	026	16	↑V	SYN
23	027	17	↑W	ETB
24	030	18	↑X	CAN
25	031	19	↑Y	EM
26	032	1A	↑Z	SUB
27	033	1B	ESC	ESCAPE
28	034	1C	↑\	FS
29	035	1D	↑]	GS
30	036	1E	↑↑	RS
31	037	1F	↑—	US

DECIMAL	OCTAL	HEX	KEY SYMBOL
32	040	20	SPACE
33	041	21	!
34	042	22	" (QUOTE)
35	043	23	#
36	044	24	$
37	045	25	%
38	046	26	&
39	047	27	' (APOS)
40	050	28	(
41	051	29	)
42	052	2A	*
43	053	2B	+
44	054	2C	, (COMMA)
45	055	2D	-
46	056	2E	. (PERIOD)
47	057	2F	/
48	060	30	0
49	061	31	1
50	062	32	2
51	063	33	3
52	064	34	4
53	065	35	5
54	066	36	6
55	067	37	7
56	070	38	8
57	071	39	9
58	072	3A	:
59	073	3B	;
60	074	3C	<
61	075	3D	=
62	076	3E	>
63	077	3F	?
64	100	40	@

DECIMAL	OCTAL	HEX	KEY SYMBOL
65	101	41	A
66	102	42	B
67	103	43	C
68	104	44	D
69	105	45	E
70	106	46	F
71	107	47	G
72	110	48	H
73	111	49	I
74	112	4A	J
75	113	4B	K
76	114	4C	L
77	115	4D	M
78	116	4E	N
79	117	4F	O
80	120	50	P
81	121	51	Q
82	122	52	R
83	123	53	S
84	124	54	T
85	125	55	U
86	126	56	V
87	127	57	W
88	130	58	X
89	131	59	Y
90	132	5A	Z
91	133	5B	[
92	134	5C	\
93	135	5D	]
94	136	5E	↑ OR ^
95	137	5F	← OR _
96	140	60	` (GRAVE)

DECIMAL	OCTAL	HEX	KEY SYMBOL
97	141	61	a
98	142	62	b
99	143	63	c
100	144	64	d
101	145	65	e
102	146	66	f
103	147	67	g
104	150	68	h
105	151	69	i
106	152	6A	j
107	153	6B	k
108	154	6C	l
109	155	6D	m
110	156	6E	n
111	157	6F	o
112	160	70	p
113	161	71	q
114	162	72	r
115	163	73	s
116	164	74	t
117	165	75	u
118	166	76	v
119	167	77	w
120	170	78	x
121	171	79	y
122	172	7A	z
123	173	7B	{
124	174	7C	|
125	175	7D	}
126	176	7E	~ (TILDE)
127	177	7F	DEL (RUBOUT)

End of Appendix

Index

Within this index, "f" or "ff" after a page number means "and the following page" or "pages". In addition, primary page references for each topic are listed first. Commands, calls, and acronyms are in uppercase letters (e.g., CREATE); all others are lowercase.

D

G

H

M

Q

R

DataGeneral

users group

Installation Membership Form

Name _____ Position _____ Date _____

Company, Organization or School _____

Address _____ City _____ State _____ Zip _____

Telephone: Area Code _____ No. _____ Ext. _____

1. Account Category

- ☐ OEM
- ☐ End User
- ☐ System House
- ☐ Government

2. Hardware

	Qty. Installed	Qty. On Order
M/600		
MV/Series ECLIPSE®		
Commercial ECLIPSE		
Scientific ECLIPSE		
Array Processors		
CS Series		
NOVA®4 Family		
Other NOVAs		
microNOVA® Family		
MPT Family		

Other _____
(Specify) _____

3. Software

- ☐ AOS
- ☐ AOS/VS
- ☐ AOS/RT32
- ☐ MP/OS
- ☐ MP/AOS
- ☐ RDOS
- ☐ DOS
- ☐ RTOS
- ☐ Other

Specify _____

4. Languages

- ☐ ALGOL
- ☐ DG/L
- ☐ COBOL
- ☐ Interactive COBOL
- ☐ PASCAL
- ☐ Business BASIC
- ☐ BASIC
- ☐ Assembler
- ☐ FORTRAN 77
- ☐ FORTRAN 5
- ☐ RPG II
- ☐ PL/1
- ☐ APL
- ☐ Other

Specify _____

5. Mode of Operation

- ☐ Batch (Central)
- ☐ Batch (Via RJE)
- ☐ On-Line Interactive

6. Communication

- ☐ HASP
- ☐ HASP II
- ☐ RJE80
- ☐ RCX 70
- ☐ RSTCP
- ☐ 4025
- ☐ X.25
- ☐ SAM
- ☐ CAM
- ☐ XODIAC™
- ☐ DG/SNA
- ☐ 3270
- ☐ Other

Specify _____

7. Application Description

○ _____

8. Purchase

From whom was your machine(s) purchased?

- ☐ **Data General Corp.**
- ☐ Other

Specify _____

9. Users Group

Are you interested in joining a special interest or regional Data General Users Group?

○ _____

CUT ALONG DOTTED LINE

DataGeneral

Data General Corporation, Westboro, Massachusetts 01580, (617) 366-8911

FOLD FOLD

TAPE TAPE

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

Postage will be paid by addressee:

Data General

ATTN: Users Group Coordinator (C-228)
4400 Computer Drive
Westboro, MA 01581

Data General

TP_____

TIPS ORDER FORM
Technical Information & Publications Service

BILL TO:
COMPANY NAME_____
ADDRESS _____
CITY _____
STATE_____ ZIP _____
ATTN: _____

SHIP TO: (if different)
COMPANY NAME_____
ADDRESS _____
CITY _____
STATE_____ ZIP _____
ATTN: _____

QTY	MODEL #	DESCRIPTION	UNIT PRICE	LINE DISC	TOTAL PRICE

(Additional items can be included on second order form)

[Minimum order is $50.00]

Tax Exempt #_____
or Sales Tax (if applicable)

TOTAL	
Sales Tax	
Shipping	
TOTAL	

METHOD OF PAYMENT

☐ Check or money order enclosed
For orders less than $100.00

☐ Charge my ☐ Visa ☐ MasterCard
Acc't No._____ Expiration Date_____

☐ Purchase Order Number:_____

SHIP VIA

☐ DGC will select best way (U.P.S or Postal)

☐ Other:
☐ U.P.S. Blue Label
☐ Air Freight
☐ Other _____

NOTE: ORDERS LESS THAN $100, INCLUDE $5.00 FOR SHIPPING AND HANDLING.

Person to contact about this order _____ Phone _____ Extension _____

Mail Orders to:

Data General Corporation
Attn: Educational Services/TIPS F019
4400 Computer Drive
Westboro, MA 01580
Tel. (617) 366-8911 ext. 4032

Buyer's Authorized Signature Date
(agrees to terms & conditions on reverse side)

Title

DGC Sales Representative (If Known) Badge #

DISCOUNTS APPLY TO MAIL ORDERS ONLY

012-1780

educational services

CUT ALONG DOTTED LINE

DATA GENERAL CORPORATION
TECHNICAL INFORMATION AND PUBLICATIONS SERVICE
TERMS AND CONDITIONS

Data General Corporation ("DGC") provides its Technical Information and Publications Service (TIPS) solely in accordance with the following terms and conditions and more specifically to the Customer signing the Educational Services TIPS Order Form shown on the reverse hereof which is accepted by DGC.

1. **PRICES**
   Prices for DGC publications will be as stated in the Educational Services Literature Catalog in effect at the time DGC accepts Buyer's order or as specified on an authorized DGC quotation in force at the time of receipt by DGC of the Order Form shown on the reverse hereof. Prices are exclusive of all excise, sales, use or similar taxes and, therefore are subject to an increase equal in amount to any tax DGC may be required to collect or pay on the sale, license or delivery of the materials provided hereunder.

2. **PAYMENT**
   Terms are net cash on or prior to delivery except where satisfactory open account credit is established, in which case terms are net thirty (30) days from date of invoice.

3. **SHIPMENT**
   Shipment will be made F.O.B. Point of Origin. DGC normally ships either by UPS or U.S. Mail or other appropriate method depending upon weight, unless Customer designates a specific method and/or carrier on the Order Form. In any case, DGC assumes no liability with regard to loss, damage or delay during shipment.

4. **TERM**
   Upon execution by Buyer and acceptance by DGC, this agreement shall continue to remain in effect until terminated by either party upon thirty (30) days prior written notice. It is the intent of the parties to leave this Agreement in effect so that all subsequent orders for DGC publications will be governed by the terms and conditions of this Agreement.

5. **CUSTOMER CERTIFICATION**
   Customer hereby certifies that it is the owner or lessee of the DGC equipment and/or licensee/sub-licensee of the software which is the subject matter of the publication(s) ordered hereunder.

6. **DATA AND PROPRIETARY RIGHTS**
   Portions of the publications and materials supplied under this Agreement are proprietary and will be so marked. Customer shall abide by such markings. DGC retains for itself exclusively all proprietary rights (including manufacturing rights) in and to all designs, engineering details and other data pertaining to the products described in such publication. Licensed software materials are provided pursuant to the terms and conditions of the Program License Agreement (PLA) between the Customer and DGC and such PLA is made a part of and incorporated into this Agreement by reference. A copyright notice on any data by itself does not constitute or evidence a publication or public disclosure.

7. **DISCLAIMER OF WARRANTY**
   DGC MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY AND FITNESS FOR PARTICULAR PURPOSE ON ANY OF THE PUBLICATIONS SUPPLIED HEREUNDER.

8. **LIMITATIONS OF LIABILITY**
   IN NO EVENT SHALL DGC BE LIABLE FOR (I) ANY COSTS, DAMAGES OR EXPENSES ARISING OUT OF OR IN CONNECTION WITH ANY CLAIM BY ANY PERSON THAT USE OF THE PUBLICATION OF INFORMATION CONTAINED THEREIN INFRINGES ANY COPYRIGHT OR TRADE SECRET RIGHT OR (II) ANY INCIDENTIAL, SPECIAL, DIRECT OR CONSEQUENTIAL DAMAGES WHATSOEVER, INCLUDING BUT NOT LIMITED TO LOSS OF DATA, PROGRAMS OR LOST PROFITS.

9. **GENERAL**
   A valid contract binding upon DGC will come into being only at the time of DGC's acceptance of the referenced Educational Services Order Form. Such contract is governed by the laws of the Commonwealth of Massachusetts. Such contract is not assignable. These terms and conditions constitute the entire agreement between the parties with respect to the subject matter hereof and supersedes all prior oral or written communications, agreements and understandings. These terms and conditions shall prevail notwithstanding any different, conflicting or additional terms and conditions which may appear on any order submitted by Customer.

DISCOUNT SCHEDULES

DISCOUNTS APPLY TO MAIL ORDERS ONLY.

LINE ITEM DISCOUNT

5-14 manuals of the same part number - 20%
15 or more manuals of the same part number - 30%

DISCOUNTS APPLY TO PRICES SHOWN IN THE CURRENT TIPS CATALOG ONLY.

DataGeneral

TIPS ORDERING PROCEDURE:

Technical literature may be ordered through the Customer Education Service's Technical Information and Publications Service (TIPS).

1. Turn to the TIPS Order Form.
2. Fill in the requested information. If you need more space to list the items you are ordering, use an additional form. Transfer the subtotal from any additional sheet to the space marked "subtotal" on the form.
3. Do not forget to include your MAIL ORDER ONLY discount. (See discount schedules on the back of the TIPS Order Form.)
4. Total your order. (MINIMUM ORDER/CHARGE after discounts of $50.00.)

   If your order totals less than 100.00, enclose a certified check or money order for the total (include sales tax, or your tax exempt number, if applicable) plus $5.00 for shipping and handling.
5. Please indicate on the Order Form if you have any special shipping requirements. Unless specified, orders are normally shipped U.P.S.
6. Read carefully the terms and conditions of the TIPS program on the reverse side of the Order Form.
7. Sign on the line provided on the form and enclose with payment. Mail to:

   TIPS
   Educational Services – M.S. F019
   Data General Corporation
   4400 Computer Drive
   Westboro, MA 01580
8. We'll take care of the rest!

educational services

User Documentation Remarks Form

Your Name _____ Your Title _____

Company _____

Street _____

City _____ State _____ Zip _____

We wrote this book for you, and we made certain assumptions about who you are and how you would use it. Your comments will help us correct our assumptions and improve the manual. Please take a few minutes to respond. Thank you.

Manual Title _____ Manual No. _____

Who are you? ☐EDP Manager ☐Analyst/Programmer ☐Other _____
☐Senior Systems Analyst ☐Operator _____

What programming language(s) do you use? _____

How do you use this manual? *(List in order: 1 = Primary Use)* _____

___ Introduction to the product ___ Tutorial Text ___ Other
___ Reference ___ Operating Guide _____

About the manual:	Yes	Somewhat	No
Is it easy to read?	☐	☐	☐
Is it easy to understand?	☐	☐	☐
Are the topics logically organized?	☐	☐	☐
Is the technical information accurate?	☐	☐	☐
Can you easily find what you want?	☐	☐	☐
Does it tell you everything you need to know	☐	☐	☐
Do the illustrations help you?	☐	☐	☐

If you have any comments on the software itself, please contact Data General Systems Engineering.
If you wish to order manuals, use the enclosed TIPS Order Form (USA only).

Remarks:

Date

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

POSTAGE WILL BE PAID BY ADDRESSEE

DataGeneral

User Documentation, M.S. E-111
4400 Computer Drive
Westborough, Massachusetts 01581

Data General Corporation, Westboro, MA 01580

069-400015-01