Introduction to Micro/RSX

Order No. AA-Y538B-TC

Micro/RSX Version 3.0

Digital Equipment Corporation Maynard, Massachusetts

First Printing, December 1983
Revised, June 1985

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by Digital Equipment Corporation or its affiliated companies.

The postpaid USER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DEC	EduSystem	UNIBUS
DEC/CMS	IAS	VAX
DEC/MMS	MASSBUS	VAXcluster
DECnet	MicroPDP-11	VMS
DECsystem-10	Micro/RSX	VT
DECSYSTEM-20	PDP	
DECUS	PDT	
DECwriter	RSTS	digital
DIBOL	RSX	

ZK-2601

This document was prepared using an in-house documentation production system. All page composition and make-up was performed by TEX, the typesetting system developed by Donald E. Knuth at Stanford University. TEX is a registered trademark of the American Mathematical Society.

Contents

Chapter 2 Creating Files

Chapter 3 Using DCL Commands

Chapter 4 Automatic Command Entry

Glossary

Index

Figures

Preface

Purpose of this Manual

This manual is intended to help new users of the Micro/RSX operating system get started using their system. It provides examples of commonly used commands, as well as instructions for creating and editing files. Micro/RSX is a complex system used for many different purposes, but it can be quite simple for an everyday user. If you are a new user, you should read this manual first and try the examples. By the time you finish reading and following the instructions in this book, you will be familiar with most of the normal procedures needed for using Micro/RSX on the MicroPDP-11 computer.

Intended Audience

This book is designed for any new user of Micro/RSX, whether familiar with computers or not.

If you are accustomed to using a computer, you may be able to get enough information from the examples and captions. However, most users will want to read the explanation before trying the examples.

Structure of This Manual

The *Introduction to Micro/RSX* consists of five parts:

- The Micro/RSX warm-up session. This uses a terminal on a Micro/RSX system. You will learn how to log in and log out, how to issue commands, and how to correct mistakes.
- An introduction to the EDT editor. You will learn how to create and edit files.
- A summary of the commonly used Micro/RSX commands, with examples of how to use them.
- An introduction to the Indirect Command Processor and to batch processing, two methods of "hands-off" use of the computer.
- A glossary of commonly used terms. All terms introduced in *italics* in this book are defined in the Glossary, as are many other terms.

Associated Manuals

You do not have to read all the books supplied with your Micro/RSX system to get started. The other books are meant to be read in order over a period of time. You should read the first two chapters of Volume 1 of the *Micro/RSX User's Guide* next. After that, you will be able to find your way to any further information you need, either through continued reading or through the help files.

Four other manuals are included in your Micro/RSX Base Kit.

The *Micro/RSX User's Guide,* Volume 1, provides a further introduction to using the Micro/RSX system and DCL commands. All the most common commands and their most common qualifiers are described. Chapter 4 of Volume 1 provides documentation of the EDT editor. Volume 1 also provides complete documentation of batch processing and the Indirect Command Processor. It concludes with an annotated, alphabetical list of all DCL commands.

The *Micro/RSX User's Guide,* Volume 2, is more detailed and includes full reference information on all DCL (DIGITAL Command Language) commands and all qualifiers. There is also information on the LINK and LIBRARY commands for programmers in Chapter 15 and a list of Micro/RSX error messages in Chapter 16.

The *Micro/RSX System Manager's Guide* is for system managers and system programmers. It includes information on day-to-day system maintenance, such as rebooting the system, creating and deleting accounts, backing up files, and setting system-wide attributes. It also includes information on more advanced concepts of system management and special commands for system programmers.

Programming on Micro/RSX surveys the programming possibilities on Micro/RSX systems and also includes an annotated list of other books about Micro/RSX and the RSX family of operating systems. It also describes the Micro/RSX Advanced Programmer's Kit, which is available separately.

Conventions Used in This Manual

Convention	Meaning
red ink	In examples, what you type is printed in red. What the system types is printed in black.
RET	Any symbol with a 2- to 6-character abbreviation indicates that you press the corresponding key on your terminal. For example, RET indicates that you press the RETURN key.
CTRL/a	The symbol CTRL/a means that you hold down the key labeled CTRL while pressing another key. For example, CTRL/Z indicates that you should press the CTRL key and the Z key together.

A Note for New System Managers

A new system manager needs to know more than this introduction includes. Your next step is to look at the *Micro/RSX User's Guide,* Volume 1, which gives you more introductory information on Micro/RSX. Once you have looked at Chapters 1 and 2 of the *Micro/RSX User's Guide,* you're ready to move on to the *Micro/RSX System Manager's Guide.*

Your new Micro/RSX system comes with two user accounts already in place. One is a nonprivileged account, meaning that anyone can use it. It is called the USER/USER account because the account name is USER and the password is USER. The other is a privileged account, meaning that only those who control the system should use it. It is called the MICRO/RSX account because the account name is MICRO and the password is RSX. One of the first things you'll learn to do as a system manager is to change the name and password of that account to something less obvious. You'll

also learn how to set up accounts for your other users, establish phone-in terminal lines, install programs, and, in general, manage your system.

The Micro/RSX operating system can be as simple, or as complex, as you want to make it. If you're patient, you'll find that everything about the system is written down, either in the *Introduction*, *User's Guide*, and *System Manager's Guide*, or in the manuals described in the final book of the Micro/RSX Base Kit, *Programming on Micro/RSX*. (These manuals are available separately, or as part of the Micro/RSX Advanced Programmer's Kit.)

All these books, by the way, include User's Comments forms. We invite you to let us know what you like or do not like about your Micro/RSX system.

Chapter 1
Getting Started

Micro/RSX is an *operating system.* The Micro/RSX operating system is a collection of *software* designed to make it easy to use the MicroPDP-11 *hardware.*

The *terminal* is used to communicate with the operating system. From your terminal you can issue *commands* that put the system to work for you. Whenever you issue a command, the system acknowledges and acts upon your command. Because you are interacting with the system, Micro/RSX is called an *interactive system.*

Generally, any mistakes you make in using a Micro/RSX system result in an *error message* that tells you what you did wrong. For the most part, there is nothing you can do to harm the system, and there are only a few things you can do that harm your own use of the system. This book warns you of most of the possible mistakes you might make.

You will also be learning *DCL,* the DIGITAL Command Language. DCL is used on many DIGITAL operating systems, so what you learn about using DCL on a Micro/RSX system is also applicable on other DIGITAL computers, large and small.

How to Use Your Terminal

Most Micro/RSX systems use *video terminals* consisiting of a typewriter keyboard and a numerical keypad like that on a calculator, and a separate screen. In this book, we assume you are using a VT100 (Figure 1–1) or VT220 (Figure 1–2) video terminal. These are DIGITAL's standard video terminals. If you don't have one or the other, you should check with someone to find out anything that is special and different about your terminal before you do the exercises in this book.

Before You Start

Make sure both the terminal and the computer are turned on.

A switch on the back of the VT100 turns it off and on. When you turn the VT100 on, the screen lights up and beeps; the ON LINE light at the top of the keyboard is lit. If the VT100 is already turned on, the screen will be lit, at least with a blinking line or square called the cursor.

A switch on the back of the VT220 turns it off and on. When you turn the VT220 on, it beeps; the lights on the keyboard flash; and the screen displays the message "VT220 OK". The VT220 may be turned on even if the screen is not lit, as this terminal rests the screen if no keys have been touched for a few minutes. If the power switch is on, but the screen is not lit, touch any key on the VT220 keyboard. The screen relights.

The large rocker switch on the MicroPDP–11 turns it off and on. When the MicroPDP–11 is on, the switch is lit. There are four small buttons labeled Halt, Restart, Write Protect, and Ready. Only the button labeled Ready should be lit. Simply press these buttons to turn the lights on or off, as needed.

Figure 1–1: VT100 Video Terminal

Figure 1–2: VT220 Video Terminal

Startup Procedure

When you turn on the MicroPDP-11, it goes through a startup procedure, also called a *bootstrap*. The startup procedure checks the hardware and brings the operating system from the fixed disk, where it is stored, into the computer's main memory. During the startup procedure, you are asked to type the time and date and press the key labeled RETURN. Use the 24-hour clock for the time; for example, use 13:00 for 1:00 P.M. Use a three-letter abbreviaton for the month; for example, 27-SEP-85.

When the startup procedure is completed, a dollar sign ($) *prompt* appears on the left-hand margin of your screen, indicating that the system is available to you.

Now you are ready to begin.

Before You Log In to the System

Press the key labeled RETURN a few times. (In examples, the symbol [RET] indicates that you press the RETURN key.) You should get a series of dollar sign ($) prompts, each on a line by itself. These prompts inform you that the system is ready to accept *input*. Input is a computer-industry word meaning whatever you type into the computer.

```
$ [RET]
$ [RET]
$
```

The RETURN key is the simplest form of command you can give the system; it enters the commands that you type and tests the system. Pressing the RETURN key is a way of saying to the system, "Are you ready?" If the system responds with the $ prompt, you can be reasonably sure that the terminal, the Micro/RSX operating system, and the MicroPDP-11 computer are ready for use.

A blinking indicator, called the *cursor*, should appear next to the $ prompt. It is called a cursor because it points out the "course" you will follow, that is, where the next character you type will appear.

Now you can log in.

Logging In

Logging in gains you access to the system.

LOGIN is the DCL command that logs you in.

Micro/RSX is a *multiuser* system. This means there can be more than one terminal and more than one person using the system at a time. In fact, there may be more people authorized to use the system than there are terminals. All these things mean that the system must have some means of keeping track of who's who. Logging in does this.

Type the LOGIN command. Enter the command by pressing the RETURN key. The system responds by asking you to identify yourself. Type USER and press the RETURN key. (Later you will use your own name or some other name assigned by the system manager, but you must log in to the account named USER to follow the examples in this book.)

The system asks you for your password. Type the password USER, but do not press RETURN yet.

Until now, everything you have typed has appeared on the terminal, but the password does not. Passwords are supposed to be secret, so that unauthorized users cannot get on the system. For this reason, passwords do not appear on your screen. In the following example, invisibility is indicated by the angle brackets.

```
$ LOGIN [RET]
Account or name: USER [RET]
Password: <USER> [RET]
```

The login procedure illustrates an important point about terminals. Each terminal is really two *devices* in one. The keyboard is an input device for sending messages to the system. The screen is an output device for receiving messages from the system.

Everything you have typed so far has been a message from you to the system. Everything that has appeared on the screen has been a message from the system to you. Until you reached the password, it looked like you were typing directly on the screen. You were not.

What you may have thought you were typing on the screen was really an *echo* from the system, confirming that you typed what you thought you did. For the password, however, the echo is suppressed for security purposes. Occasionally, when the system is busy, you may notice that the echo takes a little longer than usual.

There can be many different input or output devices on your Micro/RSX system, but the terminal is one of the most common. Later, you'll learn about others.

Now enter the password by pressing the RETURN key.

Note that the system allows one minute for logging in. If you don't complete the login procedure within one minute, this message appears on your terminal:

```
HEL - Timeout on response
```

Simply begin the login again by typing LOGIN and pressing the RETURN key.

Login Messages from the System

During the *login* procedure, Micro/RSX checks your identification and password to make sure you should be allowed on the system. If you are identified correctly, the system makes your terminal available to you, as indicated by the return of the $ prompt. A number of messages from the system may appear before you see the prompt. For example, you may see the following messages:

```
$ LOGIN  [RET]
Account or name: USER  [RET]
Password: <USER>  [RET]

Micro/RSX V3.0 Multiuser Operating System
14-MAR-85 08:15 Logged in on terminal TT2:

Good Morning

**Softball team practice today.**
```

```
@LB:[1,2]LOGIN.CMD
$ SET DEFAULT [USER]/NAMED
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
|                                                      |
|    Hello.                                            |
|                                                      |
|     You are now logged in on the                     |
|     Micro/RSX Operating System.                      |
|                                                      |
|     This is the USER account.  Nothing               |
|     that you do in this account can do               |
|     any harm either to the system, or to             |
|     this account.                                    |
|                                                      |
|     Use this account with the terminal               |
|     warm-up session described in the book            |
|     Introduction to Micro/RSX.                       |
|                                                      |
|     Take it easy.                                    |
|                                                      |
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

$
```

Whenever you log in, you may get certain messages from the system, such as "Good Morning." In addition, you may get informational messages, such as the note about softball practice. In the USER account, you also get the special greeting shown. At any rate, after everything has happened that is going to happen, the system returns the $ prompt, signifying that it is ready for more input.

If the prompt does not appear, press the RETURN key.

Correcting Typing Mistakes

Typing mistakes are by far the most common errors made by Micro/RSX users. If you make a typing mistake, press the DELETE key once for each character you want to delete. The DELETE key always deletes the character immediately to the left of the cursor. In this book, the symbol [DEL] means that you press this key. The DELETE key on the VT200-series terminals looks like this: <X] .

Beware the BACK SPACE key. This key is not used in DCL. Any command line with a BACK SPACE will either be rejected or misinterpreted by Micro/RSX. (This key can be used when you use the EDT editor, but never when you enter commands to Micro/RSX.)

The most common errors made by terminal users involve confusing the zero (0) and the capital O and confusing the number one (1) and the lowercase L (l) or I (i). Type these characters to see how they differ in appearance on your terminal. After you have typed them, delete them.

When you press the DELETE key, the character immediately to the left of the cursor disappears and the cursor takes its place. The next character you type appears in the vacated location. You can continue deleting until you reach the left margin, but you cannot delete prompts.

Lowercase letters are not used in the examples in this book, but Micro/RSX generally accepts either lowercase or uppercase commands. You may want to use the SHIFT or CAPS LOCK key to make your examples look like ours. Note that on most DIGITAL-manufactured keyboards the CAPS LOCK key has no effect on the number and symbol keys.

Deleting Whole Lines

CTRL/U deletes an entire line.

If you wish, you can delete an entire line using the CTRL/U command. Press the keys marked CTRL and U at the same time, just as you use the SHIFT key and a letter together to type an uppercase letter. See the following example.

```
$ INADEQUATE COMMAND^U
$
```

When you press CTRL/U on your terminal, it is echoed as a *circumflex* (^), also called an up-arrow, and a U. The line you typed is ignored.

Ending Input

CTRL/Z indicates End-of-File or End-of-Input.

```
More input? ^Z
$
```

Pressing CTRL/Z tells the system that you have finished supplying input. CTRL/Z is a command you can try when your terminal appears to be *hanging* or otherwise behaving in some confusing way. You also use CTRL/Z to terminate input to many *system tasks*, but more about that later.

Clearing Your Screen

If, after you've issued several DCL commands, you get tired of seeing everything you've typed on the screen, you can clear the screen completely by typing the following:

```
$ CLR [RET]
```

This command clears the screen. It has no other effect.

Displaying Information on Your Terminal

The SHOW command displays system information.

The SHOW TIME command displays the time.

Type SHOW and enter it. DCL prompts you for the next portion of the command. DCL always prompts you if you have not given a complete command. Some commands prompt you more than once.

TIME is one function you can display. Type TIME in response to the **Function?** prompt. Micro/RSX displays the current system time.

```
$ SHOW [RET]
Function? TIME [RET]
12:37:33 10-JUL-85
```

Now type SHOW TIME on one line.

```
$ SHOW TIME [RET]
12:37:33 10-JUL-85
```

Micro/RSX displays the current system time. Both forms of the command—with or without the prompt—return the same information. The **Function?** prompt identifies the next command element DCL is expecting. As you will see, you can use SHOW to display a variety of system information.

Prompts like **Function?** are most useful when you are learning a command, or using a rarely used command. As you see in the two forms in the example, DCL prompts you only when you omit a necessary part of the command. Once you have learned a command format, you probably will not need the prompts, but they will always be available.

If you decide not to enter a command, you can always press CTRL/Z at the end of the line or in response to the prompt, and the command is not executed.

For example, type SHOW, with or without pressing RETURN. Then press CTRL/Z.

```
$ SHOW^Z
```

or

```
$ SHOW [RET]
Function? ^Z
$
```

As you can see, nothing happens. Pressing CTRL/Z cancels any command.

Shortening Commands

DCL does not require that you type the full command.

Try dropping letters from the SHOW TIME command. You will find that S TI is sufficient to display the time.

```
$ S TI [RET]
12:37:33 10-JUL-85
$ S T [RET]
SHOW -- Illegal function
S T
  ^
$
```

The full command, such as SHOW TIME, identifies the action of the command. As you will see later, DCL commands can be quite complex, and the full form of the command is useful to help you see what action you are performing. For everyday convenience, however, DCL accepts abbreviations. You can abbreviate any DCL command.

In the case of SHOW, the abbreviation is S. There are several commands with one-letter abbreviations. For most commands, three letters will be enough, and four will always be enough. You can experiment with new commands as you learn them, shortening the command until you get an error message like the one just shown.

In the example, the abbreviation S T didn't work because there is another DCL command, SHOW TERMINAL. Therefore, S TI is as short as SHOW TIME can be. S TE is as short as SHOW TERMINAL can be.

For the sake of clarity, this book uses only full commands in examples.

Help from Micro/RSX

Type the command HELP SHOW TIME and enter it. Text explaining the SHOW TIME command appears on your screen. (You should remember that the time displayed is the time as it is set on the system, and is no more accurate than any other clock.)

```
$ HELP SHOW TIME [RET]

SHOW DAY[TIME] or SHOW TIME

The SHOW DAYTIME command displays the current time and date. The time
is in 24-hour format and the date is formatted as dd-mmm-yy.
$
```

You may get an error message, such as the following:

```
$ HELP SHOW TIME [RET]
HEL -- HELP file error -26
$
```

This means the *files* needed to make the HELP command work are temporarily unavailable. (The next section tells you more about files.) If this message appears, see your system manager. It is not difficult to restore the help files whenever you need them.

As you go through this book, you should use the help available for each command that you learn.

More Help from Micro/RSX

You can also get help from Micro/RSX while entering commands.

Type SHOW and enter it. When the **Function?** prompt appears, type a question mark (?). Help text appears, followed by another **Function?** prompt.

```
$ SHOW [RET]
Function? ? [RET]
SHOW thing

The SHOW command can be used to show something. The following things
can be shown with this command:

ACCOUNTING     [DAY]TIME     LIBRARY       QUEUE     TERMINAL
ASSIGNMENTS    DEFAULT       PARTITIONS    SYSTEM    UIC
CLOCK_QUEUE    DEVICE        PROCESSOR     TASKS     USERS
COMMON         ERROR_LOG     PROTECTION

For further help on the qualifiers, type HELP SHOW qualifier.

Function? ? TIME [RET]
```

As you see, you can display a great deal of information with the SHOW command. The display you see is the same you would see if you typed HELP SHOW.

Now type ? TIME in response to the **Function?** prompt. This time, the help text you see is the same you would see if you typed HELP SHOW TIME.

```
SHOW DAY[TIME] or SHOW TIME

The SHOW DAYTIME command displays the current time and date. The time
is in 24-hour format and the date is formatted as dd-mmm-yy.

Function? TIME [RET]
14:53:48 10-JUL-85
$
```

This time, type TIME in response to the **Function?** prompt. The time is displayed.

You can get help on any DCL command in this way. If you have typed a command but you aren't sure what the prompt means, type in a question mark (?) to get further information. Everything you had typed before you typed the ? is preserved for you while you get help.

A Directory of Your Files

The DIRECTORY command displays information about stored files.

Type DIRECTORY and enter it. What you see on your terminal should resemble the example, but your list of files may be somewhat different.

```
$ DIRECTORY [RET]

Directory DUO:[USER]
14-MAR-85  14:56

WHATSHERE.TXT;1          3.          27-JAN-85 16:24
HELLO.TXT;1              2.          27-JAN-85 16:24
LONG.TXT;1               25.         27-JAN-85 16:24
FLY.TXT;3                1.          27-JAN-85 16:24
FLY.TXT;2                1.          27-JAN-85 16:24
FLY.TXT;1                1.          27-JAN-85 16:24
MYDISK.CMD;1             4.          27-JAN-85 16:24
LOGIN.CMD;1              1.          27-JAN-85 16:24
FLU.TXT;1                1.          27-JAN-85 16:24
SHOW.CMD;1               1.          27-JAN-85 16:24

Total of 40./40. blocks in 10. files
```

Files are one of the basic units of storage on Micro/RSX systems. Everything on a Micro/RSX system either starts out or ends up as a file. Put simply, a file is the means used to separate one significant collection of material from another. Files can contain text, runnable programs (called *tasks*), or various kinds of data.

The DIRECTORY command provides you with a list of all the files stored on a particular *device* in a particular *directory*. Later you will have a directory and an *account* under your own name.

Devices on Your System

Most Micro/RSX systems have a limited number of devices, usually two diskette drives, or a tape drive, and one or more fixed disks. The diskettes and tape can be removed and changed, but the fixed disk is permanently in place, or nonremovable. You will generally use the fixed disk to store files that you are currently working on or that you use all the time. You will generally use diskettes and tapes to store files that need not always be available.

All Micro/RSX devices have names such as DU2:, which is a diskette drive. Notice that the name has two letters, a number, and ends with a colon (:). The fixed disk on your system is named DU0:. If you have a tape drive, it is probably named MU0:. The device name is important in helping you find your files.

There are many other kinds of devices besides mass storage devices. *Line printers* and terminals are both devices, for instance. Your system may also have other kinds of devices not mentioned here.

File Specifications

Each file stored on a Micro/RSX system has a unique identification, or specification, also called a filespec. The device name and directory name are important parts of the *file specification.*

At the head of the directory listing, you will see the device name (DU0:) and directory name ([USER]) where the files are stored. Within the listing, each file is identified by a file name (WHATSHERE), a file type (.TXT), and a version number (;1). These five elements fully distinguish one file from all the others on the system and also permit you to locate it.

Thus, the full file specification of the first file listed in the directory example is:

DU0:[USER]WHATSHERE.TXT;1

A complete file specification consists of a device name, a directory name, a file name, a file type, and a version number. There can be only one file with this full specification; there may be others with similar, but not identical, specifications.

The *syntax* rules for all *fields* of the file specification are really quite simple:

- Device names have two letters and a number, followed by a colon (:).
- Directory names have two possible formats. The named directory format has one to nine of the following characters: the 26 letters A through Z and the numbers 0 through 9, or it has two numbers separated by a comma and enclosed in brackets ([ ]). The numbered directory format has two numbers separated by a comma and enclosed in brackets ([ ]).
- File names have one to nine letters or numbers.
- File types have one to three letters or numbers and begin with a period (.).
- Version numbers start at 1 and go up; they are separated from the file type by a semicolon (;).

As you go through the exercises in this book, you will see that each field of the file specification has a part to play in the smooth functioning of the system.

But don't worry. Most of the time, all you'll have to type is the file name and type.

Displaying Files on Your Terminal

The TYPE command displays selected files on your terminal.

Type TYPE and enter it. Give the full file specification as shown in response to the **File(s)?** prompt. A short file is printed on your terminal.

```
$ TYPE RET
File(s)? DU0:[USER]FLY.TXT;3 RET
Time flies like an arrow.
Space flies like a bow.
Fruit flies like a banana.
```

Now try the one-line form of the command; leave out the device and directory names and the version number.

```
$ TYPE FLY.TXT [RET]
Time flies like an arrow.
Space flies like a bow.
Fruit flies like a banana.
```

You do not always have to include the full file specification to specify a given file. Some parts are included by *default* if you do not specify them.

The use of the word "default" may be slightly confusing. In general, "default" means "for lack of competition." If all but one runner drops out of a foot race, the last runner wins by default. On Micro/RSX, if you do not supply a value, the system supplies a value of its own, for lack of competition. If you supply a value, your value always "wins."

Defaults in File Specifications

The SHOW DEFAULT command displays the default device and directory for your terminal.

```
$ SHOW [RET]
Function? DEFAULT [RET]
   DU0:[USER]  Named  TT2:
   Protection UIC:    [200,1]
$
```

The default device and directory are automatically included in every file specification, unless you have included some other device or directory. You can find out your default device and directory, which is "where you are" on Micro/RSX, with the SHOW DEFAULT command. Every file in DU0:[USER] has that disk name and directory name as part of its filespec. (SHOW DEFAULT also tells you that your directory can use a name instead of numbers, which terminal you are on, and your User Identification Code (UIC), but that isn't part of the file specification default.)

There may be several files on your system called FLY.TXT;3, but there is only one called DU0:[USER]FLY.TXT;3. If you wanted to see one of the others, you'd have to include its directory name and disk name when you typed its name, for example, DU1:[LEMEN]FLY.TXT;3.

One important default does not show. If you do not supply a version number, Micro/RSX defaults to the highest-numbered version.

Type and enter the following command:

```
$ TYPE FLY.TXT [RET]
```

The file FLY.TXT;3 is printed on your terminal.

Now type and enter this command:

```
$ TYPE FLY.TXT;1 [RET]
```

As you see, two files with the same name and type but different version numbers can differ greatly.

If you want to see a copy of a file from another device or directory, or both, include the device name and directory name in the file specification; the defaults of DU0: and [USER] are overridden.

Type and enter the following command:

```
$ TYPE DU0:[1,2]LOGIN.TXT [RET]
```

You should see most of the same text that is printed on your terminal when you log in, if any.

Defaults are designed for your convenience, but you can always override them. Usually, defaults are set to produce the most commonly used form of the command or file specification.

Controlling Output to Your Terminal

The NO SCROLL and HOLD SCREEN keys delay output to your terminal.

CTRL/O skips over output to your terminal.

Type and enter the following command:

```
$ TYPE [RET]
File(s)? LONG.TXT [RET]
```

The file begins to appear on your screen. Immediately press the NO SCROLL (VT100) or HOLD SCREEN (VT220) key on your terminal. The file you are displaying stops right where it is.

Often on video terminals, the output from a command may *scroll* past on the screen too fast for you to read. (Scroll means unroll like a scroll on your screen.)

Now press NO SCROLL or HOLD SCREEN again. The file starts scrolling past again.

The NO SCROLL and HOLD SCREEN keys do the same thing: they stop the scrolling, or, if you prefer, hold the screen. You are not missing any output when the key is in effect. These keys allow you to read output at your own pace, rather than the terminal's.

Note

If your terminal appears to be hanging (not accepting new input), you may have accidentally pressed the NO SCROLL or HOLD SCREEN key.

Again, type and enter the following command:

```
$ TYPE LONG.TXT [RET]
```

This time, immediately press CTRL/O. The output display from the TYPE command stops immediately. Press CTRL/O again. The output display from the TYPE command starts again, but not at the same place.

CTRL/O works like the fast-forward switch on a tape recorder. Use CTRL/O to skip over output you do not want to see.

You can either skip the rest of the file entirely or skip down rapidly. You are only skipping what you would see on the screen. You are doing nothing to the file. Type the file again to be sure.

Stopping the Action Entirely

Again, type and enter the command:

```
$ TYPE LONG.TXT [RET]
```

This time, press CTRL/C. The file stops printing on your terminal almost immediately and the $ prompt returns.

You have *aborted* the TYPE command. Therefore, the printing of LONG.TXT on your terminal ceases. CTRL/C is one of the most useful commands on Micro/RSX systems. Whenever something is happening at your terminal that you do not like, or, if you want the system to stop doing something you told it to, all you have to do is press CTRL/C and it will stop happening.

Pressing CTRL/C causes a Micro/RSX *task* to abort. That is not as drastic as it sounds. It means only that pressing CTRL/C stops the current program from running. Everything you see happening on Micro/RSX is caused by a task, or program. An operating system is a complex collection of tasks. Even the commands that cause tasks to run are themselves tasks. This means that it is handy to be able to stop a task from running. No harm

comes to the task when it is aborted; it simply goes away. You can run it again right away. In this example, the TYPE command still works and LONG.TXT is intact. (Of course, you shouldn't abort programs willy-nilly; some tasks may be affected if they are aborted.)

Setting and Showing

The SET and SHOW commands can be used to change and display system attributes.

The SET TERMINAL command sets, and resets, the attributes of your terminal.

For example, SET TERMINAL/LOWERCASE causes your terminal to leave lowercase input unchanged. SET TERMINAL/UPPERCASE causes your terminal to convert lowercase input to uppercase.

The SHOW TERMINAL command displays attributes of your terminal and other terminals on the system.

Type and enter the following command:

```
$ SET TERMINAL/LOWERCASE [RET]
```

Remember, you can use either the prompting form or the single-line form for DCL commands.

Now type and enter the following command:

```
$ SHOW TERMINAL [RET]
TT2:     [USER]  [200,1]  14-MAR-85 14:47   1  A. USER
CLI    = DCL     BUF    = 80.     HFILL = 0
LINES  = 24.     TERM   = VT100   OWNER = none     BRO      NOABAUD
LOWER    NOPRIV  NOHOLD   NOSLAVE NOESC   CRT      NOFORM   NOREMOTE
ECHO     NOVFILL HHT      NOFDX   WRAP    NORPA    NOEBC    TYPEAHEAD
CTRLC    NOAVO   ANSI     DEC     EDIT    NOREGIS NOSOFT    NOBLKMOD
SERIAL   NOHSYNC NOPASTHRU        TTSYNC  NOPRINTER_PORT
```

Micro/RSX displays all the attributes set for your terminal. Most of these attributes look very technical, and they are. Most terminal attributes are more of interest to programmers than users. If a particular terminal attribute is important, your terminal will probably already have it set. We are using SET TERMINAL and SHOW TERMINAL for this example because the effects are simple, obvious, and harmless.

In the list of attributes of your terminal, you will see LOWER. LOWER means that whatever you type in lowercase is sent to the system in lowercase. If your terminal is not set LOWER, any lowercase character you type is echoed in uppercase. The echo lets you see exactly what the system received. (Actually, in most cases, the system doesn't care whether the characters are uppercase or lowercase.)

Not all the terminal attributes listed by SHOW TERMINAL have obvious meanings, but they include your terminal number, the width of your screen (under BUF), and the length of your screen (under LINES). The other terminal attributes are explained in the *Micro/RSX User's Guide* and the help files.

Now, type and enter the following command:

```
$ SET TERMINAL/UPPERCASE RET
```

Your terminal is now set to translate any lowercase characters you type into uppercase before transmitting them. Any lowercase character you type will be echoed in uppercase. Why would you want to do this? Well, some computer programs don't understand lowercase. But the only reason for doing it in this example is for practice in setting something.

Now type and enter the SHOW TERMINAL command again. Where the display listed LOWER, you now see NOLOWER (which is a computer way of saying UPPER).

```
$ SHOW TERMINAL RET
TT2:     [USER]  [200,1]  14-MAR-85 14:47   1  A. USER
CLI   = DCL      BUF    = 80.      HFILL = 0
LINES = 24.      TERM   = VT100    OWNER = none     BRO      NOABAUD
NOLOWER NOPRIV   NOHOLD   NOSLAVE NOESC    CRT      NOFORM   NOREMOTE
ECHO     NOVFILL HHT      NOFDX    WRAP     NORPA    NOEBC    TYPEAHEAD
CTRLC    NOAVO   ANSI     DEC      EDIT     NOREGIS NOSOFT    NOBLKMOD
SERIAL   NOHSYNC NOPASTHRU         TTSYNC   NOPRINTER_PORT
```

Try typing something in lowercase. It comes out in uppercase. You will probably want to set your terminal back to LOWER before continuing with this warm-up session.

Displaying System Information

The SHOW USERS command displays a list of logged-in users.

Type SHOW USERS and enter it. Micro/RSX displays a list of everyone currently logged in on the system. For each user, the list includes the terminal number, the default directory, the protection UIC (User Identification Code), the login time, the number of active tasks, and the user's name.

```
$ SHOW [RET]
Function? USERS [RET]
TT1:     [JOEL]      [303,4]    19-FEB-85 11:27   0      D. JOEL
TT2:     [USER]      [200,1]    19-FEB-85 13:05   1      A. USER
TT4:     [7,11]      [7,11]     19-FEB-85 09:19   1      H. TAVANI
TT5:     [MCCARTHY]  [7,57]     19-FEB-85 08:15   1       . MCCARTHY
```

Note that the terminal number has the same form as other device names: two letters, a number, and a colon. The two letters identify the device type, in this case, a terminal.

Logging Out

The LOGOUT command logs you off the system.

When you are through using Micro/RSX, you must log yourself out using the LOGOUT command.

```
$ LOGOUT [RET]
Connect time:   0 hrs  30 minutes   8 secs
CPU time used:  0 hrs   0 minutes  23 secs
Task total:       84
Have a Good Afternoon
14-MAR-85 14:48   TT2: logged off
```

Micro/RSX gives you some statistics on your system use when you log out. Notice the disparity between CONNECT TIME, which shows you how long you were logged in, and CPU TIME USED, which shows you how much of that time you were actually using the central processing unit or *CPU*. The CPU is the part of the computer that actually computes.

For most of the time that you were logged in, the system was waiting for input from you. This is one of the main reasons why it is possible for several users to do work simultaneously on the system. While the system is waiting for input from you—even between the time you type one character and the next, which is a long time to a computer—some other user's needs can be served.

As part of logging you out, the LOGOUT command cleans up behind you, aborting any active tasks and returning resources to the system.

There's just one more point you should know about the Micro/RSX system: sometimes systems *crash*.

Sometimes, nothing seems to work, not even CTRL/C or the LOGOUT command. In fact, you can type without getting any response from the system. On these occasions, the system may have crashed. Crashing is not as serious as it sounds either. If the system crashes, it is probably not your fault. A crash is the system's response to an unstable condition, usually caused by a *privileged* user, or a privileged task, or a hardware problem. If the system should crash, the system manager or some other responsible person (maybe you) will probably be bringing the system back in a few minutes.

After a crash, the system must be *bootstrapped*. This can be done simply by pressing the Restart button on the computer (or turning the power switch off and on) and waiting for a confirmation message. This message may be nothing more than the $ prompt.

No one is logged in if the system has crashed and then been restarted.

Summary

In this chapter, you have learned the most common ways of eliminating confusion from your terminal and restoring tranquility:

- CTRL/Z for ending input
- CTRL/C to stop a program
- CLR to clear the screen
- HELP or question mark (?) to get help
- LOGOUT to end your use of the system

You now know enough about using your terminal and the system to move on to some of the more complex, and useful, facilities of the Micro/RSX system.

You have had to learn some computer jargon and some Micro/RSX jargon, and you will be learning some more as you continue working with this book. Micro/RSX is part of the RSX family of computer operating systems. These systems can be quite complex. If you learn the correct terms from the beginning, your chances of understanding what is happening on the system are improved.

You should understand that all the commands discussed so far have been commands to the operating system itself. The system also includes many *utilities* to help you.

The next chapter of this book demonstrates the use of one of these utilities, an *editor,* which is used to create files.

Chapter 2
Creating Files

A file is a collection of data significant to a user.

This definition covers a lot of territory. Files in Micro/RSX systems can be of many types. *Text files,* like FLY.TXT, are in a form you can read. Other files, such as *task image files* (files that contain a runnable program), are in a form that only the computer and operating system can read.

The file type—the three-letter identification after the file name, such as .TXT—usually gives a clue as to the contents of a file. There is a list of common file types in Chapter 5 of the *Micro/RSX User's Guide.*

This chapter tells you how to create and edit files on a Micro/RSX system.

You will create and edit a text file called DOCTOR.FEL. The instructions in this chapter have been designed to introduce you to many of the commands you need to know to create and edit text files. Follow the examples closely, so you can see how everything works.

Creating a File

The CREATE command creates files.

Type CREATE and enter it. In response to the **File?** prompt supply the file name DOCTOR and the file type .FEL and enter it.

The prompt does not appear. Type the line of text shown in the example below, including the RETURN. Note that the RETURN key works like a typewriter carriage return. Then press CTRL/Z. The $ prompt returns.

```
$ CREATE [RET]
File? DOCTOR.FEL [RET]
I do not like you, Doctor Fell, [RET]
^Z
$
```

While you were typing, you were "in" the CREATE task. CREATE is not only a DCL command; it is also a system task used to create files. While you were in this task, you were not in DCL. DCL commands have no effect inside the CREATE task.

Pressing CTRL/Z indicated the end of your input to the CREATE task. It took you "out" of the CREATE task and returned you to DCL level.

You have now created a file called DOCTOR.FEL. It is automatically numbered version 1 and placed in the default directory, which is [USER]. Thus, the file has the file specification of DU0:[USER]DOCTOR.FEL;1. You can use the DIRECTORY command to see that the file is there.

Note that although you have created a text file, the file type is not .TXT. Micro/RSX permits you to give whatever name and file type you like to your files. On the other hand, Micro/RSX systems also provide default file types for various purposes; these are discussed in Chapter 5 of the *Micro/RSX User's Guide*. (There is no default file type for the CREATE command.)

The CREATE command is handy for making notes, but you can't really do much in CREATE. You can't go back up a line to change something, for instance. For fancier *functionality*, Micro/RSX provides an editor, a system task designed to make creating and changing files easier.

EDT, the DIGITAL Standard Editor

The EDIT command invokes EDT, the DIGITAL standard editor.

EDT, the editor included on Micro/RSX systems, is also found on a number of other DIGITAL operating systems. It is a general-purpose interactive text editor with two modes of operation. *Line mode* has English-like commands and works on either video or *hardcopy* terminals. As the name implies, line mode editing is done on one line of text at a time. *Character mode* works on video terminals only. When you use character mode editing, you can work on a whole file.

In this book there are brief introductions to both modes of EDT. However, the emphasis is on character mode editing, because it is the more common mode of editing used on video terminals. More information on both types of editing can be found in the *Micro/RSX User's Guide*.

Startup Command Files

EDT allows you to set the characteristics of your editing session by using a file called a startup command file. For example, you can create a startup command file that specifies how many characters per line appear on the screen and whether you want character mode editing. The default name for this file is EDTINI.EDT, for EDT initialization.

EDT looks for a file named EDTINI.EDT each time you begin an editing session. If it finds one, EDT executes the commands in it. If there is no such file, EDT simply uses its default characteristics, including line mode editing.

The following sample EDTINI.EDT file does only two things. It sets the editing mode to character and sets the wrap, that is, when the line breaks, to 75 characters.

```
SET MODE CHANGE
SET WRAP 75
```

You will probably want to create an EDTINI.EDT file in your own account. However, the [USER] account simply uses the default characteristics of EDT.

Character Mode Editing

EDT character mode allows you to edit at any position in your text. Your screen always contains an accurate picture of the part of the file you are working on. The cursor shows exactly where you are at all times.

Character editing uses the keypad on your terminal. If your terminal has no keypad, see Chapter 4 of the *Micro/RSX User's Guide* for information on using character mode.

The Keypad

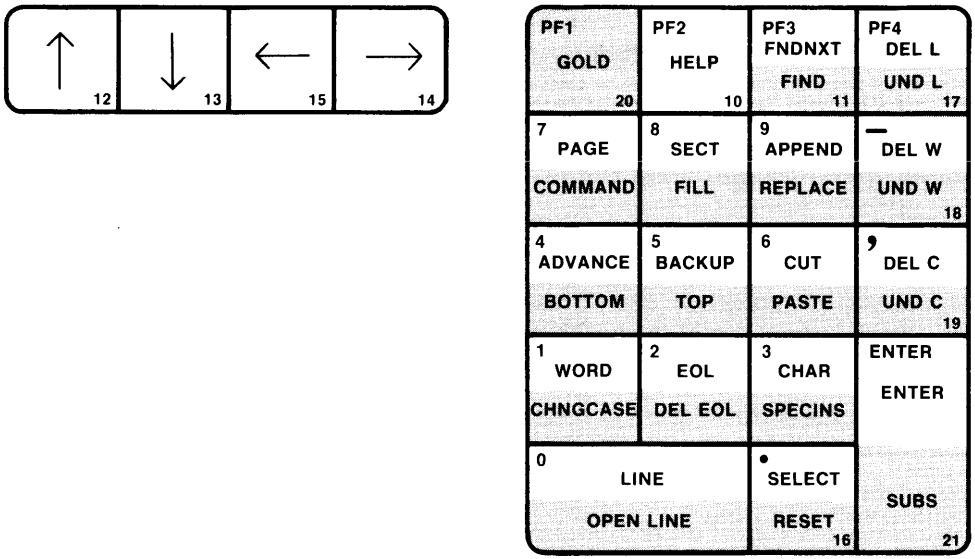
Character (or keypad) editing works on the VT100 and VT200-series video terminals and other terminals with a numerical keypad. In character editing, you request editor functions by pressing keys on the keypad. No RETURN is required to enter the command. Anything you type on the regular keyboard, including RETURN, is inserted into the file as text.

It is a good idea to keep a copy of the keypad diagram handy while you are learning character editing. The keys on your terminal are not labeled with EDT commands, but with numbers and some other characters. Figures 2–1 and 2–2 show the meaning of each key on the keypad for the VT100 and VT220. The numbers or characters shown in the upper right of each key correspond to what you see on the key.

In this chapter, the keypad key number is noted in parentheses the first time the key is mentioned, but not afterward. For instance, GOLD (PF1) indicates that the GOLD function uses the key labeled PF1.

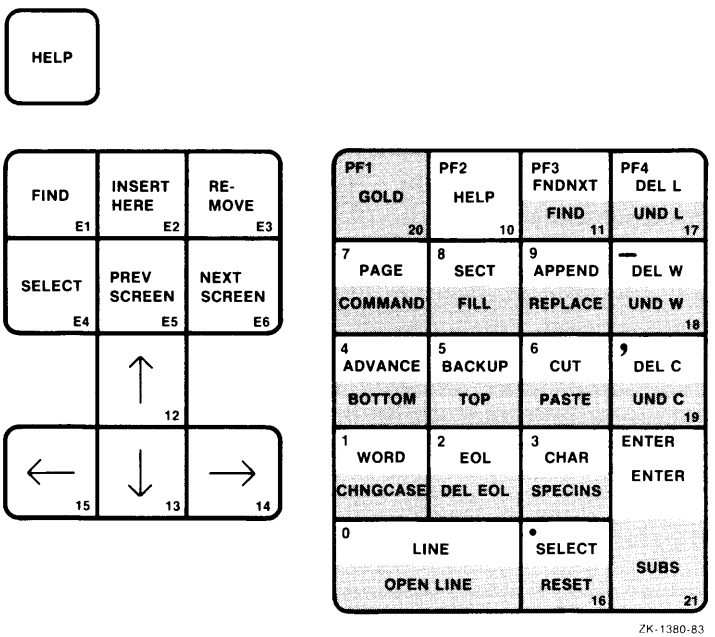
As shown in the diagrams, most keys perform two functions. When you want to use the upper of the two functions listed, simply press the key. To use the lower (shaded) function, first press and release the GOLD key (PF1) and then press the key you want to use.

Figure 2–1: VT100 Keypad



ZK-1377-83

Figure 2–2: VT220 Keypad



ZK-1380-83

Beginning an Editing Session

Type and enter the command EDIT. At the **File?** prompt, type and enter the file name DOCTOR.FEL.

```
$ EDIT
File? DOCTOR.FEL
1 I do not like you, Doctor Fell,
* [RET]
[EOB]
*
```

The line appears on the screen, followed by an asterisk (*), which is the EDT line mode prompt. The asterisk signifies that you are "in" EDT and that EDT is ready to accept your line mode commands.

Note that the line has the *line number* 1. EDT automatically numbers lines. Line numbers are one way of locating text in your file.

Now press RETURN. You see the symbol [EOB], which means End-of-Buffer, and another asterisk prompt. In EDT, a *buffer* is a workspace used in editing files. In this case, the buffer contains one line of text. The [EOB] tells you that you have reached the end, or bottom, of the buffer.

As you will see, the end of the buffer keeps moving down as you add lines to your file. For this practice session, you will be working mostly in a buffer called MAIN. EDT permits you to create more buffers if you need them. See Chapter 4 in the *Micro/RSX User's Guide* for more information.

Entering Character Mode

At the asterisk prompt, type and enter the CHANGE command (abbreviation C) to enter character mode.

When you issue the CHANGE command, the one-line file and the [EOB] symbol appear at the top of the screen. You are ready to begin editing.

Type the following lines, including RETURN:

```
The reason why I cannot tell. [RET]
But this I know, and know full well, [RET]
I do not like you, Doctor Fell.
```

Getting Help on EDT

You can get help on the keypad functions any time during an editing session. Simply press the HELP (PF2) key. This prints a copy of the keypad diagram on your screen. While the diagram is showing, you can press any keypad key to get help on using that key. When you are through with the help, press the space bar to return to editing. Try it out now.

Moving the Cursor

The cursor always appears where the next character will appear or the next action will take place. You can move the cursor in many different ways. Experience will teach you which is best in a given situation.

Using the Arrow Keys

The easiest way to move the cursor is by using the arrow keys. On a VT100 terminal the four arrow keys are located at the top of the keyboard. On the VT200-series terminals, they are located between the keyboard and the keypad.

The LEFT and RIGHT arrows move the cursor one character to the left or right. If the cursor is at the end of a line, the RIGHT arrow moves it to the beginning of the next line. Conversely, if the cursor is at the beginning of a line, the LEFT arrow moves it to the end of the previous line.

The UP and DOWN arrows move the cursor one line up or down. The column position of the cursor does not change, unless there is no text in the corresponding column above or below. In that case, the cursor moves to the end of the preceding or following line.

Try using the arrow keys. Note that the cursor will not move beyond the limits of the buffer. If you try to go beyond the limits, the message "Advance past bottom of buffer" or "Backup past top of buffer" appears on the screen.

Changing the Direction of Cursor Movement

The keypad commands ADVANCE (4) and BACKUP (5) change the direction in which the cursor moves. The ADVANCE key causes the cursor to move forward, toward the end of the buffer. The BACKUP key causes the cursor to move back through the text, toward the top of the buffer.

When you start editing, the cursor moves forward by default. You may want it to move backward if you are searching for a word or making a series of changes. However, it can be confusing to edit a file with BACKUP in effect. In general, you should follow any use of ADVANCE immediately with BACKUP.

The TOP (GOLD + 5) and BOTTOM (GOLD + 4) functions move the cursor to the top and to the bottom of the file, respectively. Note that these two functions use the GOLD key plus BACKUP or ADVANCE.

Use TOP to move the cursor to the top of your file.

```
I do not like you, Doctor Fell,
The reason why I cannot tell.
But this I know and know full well,
I do not like you, Doctor Fell.
```

Moving the Cursor by Character, Word, and Line

You can move the cursor one character at a time with the CHAR (3) function and one word at a time with the WORD (1) function.

The LINE (0) function moves the cursor to the beginning of the next line; the EOL (2) function moves the cursor to the end of the line.

Remember that if BACKUP is in effect, these functions will move the cursor back instead of forward.

Use the LINE and WORD functions to move the cursor to the word "why".

```
I do not like you, Doctor Fell,
The reason why I cannot tell.
But this I know and know full well,
I do not like you, Doctor Fell.
```

You can also move the cursor through larger entities of text with the PAGE (7) and SECT (8) functions. However, this file is not long enough to try them out. For more information, see Chapter 4 in the *Micro/RSX User's Guide*.

Inserting Text

There are two ways to insert text.

One way to insert text is simply to move the cursor to where you want the new text to appear and begin typing.

To try this out, move the cursor to "tell" and type the words "and will not". The screen looks like this:

```
I do not like you, Doctor Fell,
The reason why I cannot and will not tell.
But this I know and know full well,
I do not like you, Doctor Fell.
```

The other way to insert text is to open a line first. The OPEN LINE (GOLD + 0) function opens a line immediately above the cursor.

Move the cursor to the beginning of the second line and try it out. The screen will look like this:

```
I do not like you, Doctor Fell,

The reason why I cannot and will not tell.
But this I know and know full well,
I do not like you, Doctor Fell.
```

Deleting text

The DELETE key functions in EDT the same way it does at the system level. It deletes a character at a time to the left of the cursor.

In EDT, however, you can also delete text by the character, by the word, and by the line.

DEL C (,) deletes the character the cursor is on.

DEL W (-) deletes a word, starting with the letter the cursor is on and going to the next space.

DEL L (PF4) deletes a line, starting with the letter or space the cursor is on and going to the next RETURN.

Use DEL L to delete the blank line.

Use DEL W three times to delete the words "and will not".

```
I do not like you, Doctor Fell,
The reason why I cannot tell.
But this I know and know full well,
I do not like you, Doctor Fell.
```

Undeleting Text

You can restore text you have deleted using the UNDELETE functions. These functions use the GOLD key plus the DEL L, DEL W, and DEL C functions.

To try this out press DEL W; "tell" is deleted. Now press UND W; "tell" is restored.

Try each of these functions by deleting and undeleting a line, a word, and a character. Note that you can restore only the last piece of text you have deleted. For example, if you use DEL C twice, you can only restore the second character that you deleted.

Locating Text

The FIND (GOLD + PF3) function lets you search through a file for a specific *string*.

To follow this example, first move the cursor to the top of the file.

Press the two keys of the FIND function. The words "Search for:" appear highlighted at the bottom of the screen. Type the word "not". To search forward through the file, press ADVANCE.

The screen looks like this:

```
I do not like you, Doctor Fell,
The reason why I cannot tell.
But this I know and know full well,
I do not like you, Doctor Fell.
```

The FIND NEXT function lets you search for repeated occurrences of a string. To find the next occurrence of "not", simply press FNDNXT (PF3). Continue to press FNDNXT until the message "String was not found" appears at the bottom of the screen.

Note that to search back through the text you use the BACKUP function in place of ADVANCE. Try searching back through the text for the word "like".

Moving Text

You can move sections of contiguous text from one location in the file to another with the CUT and PASTE functions.

The SELECT (.) function marks a range of text, in this case the text you are going to move. The text you put in the select range is highlighted on the screen.

The following example moves the third line to the first line.

First, move the cursor to the first letter of "But". Press SELECT. Then press LINE. The third line appears highlighted, indicating that it has been marked by SELECT. (If you select the wrong text, use RESET (GOLD + .) to undo it.)

CUT (6) moves the text in the select range from the buffer you are working in to a buffer named PASTE. The PASTE buffer does not appear on the screen.

Press CUT. The selected text disappears from the screen:

```
I do not like you, Doctor Fell,
The reason why I cannot tell.
I do not like you, Doctor Fell.
```

PASTE (GOLD + 6) moves the text from the PASTE buffer back to the main buffer, locating it at the cursor.

For this example, move the cursor to the top of the file. Press PASTE.

```
But this I know and know full well,
I do not like you, Doctor Fell,
The reason why I cannot tell.
I do not like you, Doctor Fell.
```

The text in the PASTE buffer is not deleted until you replace it with new text or end your editing session. Move the cursor to the beginning of the last line and press PASTE again. The same line appears. Use TOP and DEL L to delete the first line and go back to the original format.

Leaving the Editor

The EXIT command takes you out of EDT and makes a new file.

The QUIT command takes you out of EDT but does not make a new file.

EXIT and QUIT must be issued from line mode. To get back to line mode, press CTRL/Z.

At the asterisk prompt, type EXIT and enter it by pressing RETURN. You are out of EDT. The file name appears on the screen along with the number of lines in the file. The dollar sign ($) prompt returns, signifying that EDT is no longer active at your terminal.

Note

You can use the COMMAND (GOLD + 7) function to issue line mode commands while still in character mode. Line mode commands issued from character mode are entered with the ENTER key.

Now type and enter the command TYPE DOCTOR.FEL. The file is printed on your terminal.

Look at your directory. You now have two versions of DOCTOR.FEL. You created version 1 with the CREATE command. When you edited version 1, you automatically created version 2.

Return to EDT using the EDIT command and naming DOCTOR.FEL as the file to be edited.

Use the CHANGE command enter character mode. Delete the first two lines of the file.

Again type CTRL/Z and, at the asterisk, type QUIT and enter it. The dollar sign ($) prompt returns immediately. EDT prints no messages on your terminal. The QUIT command directs EDT not to create a new version of the file.

Type and enter TYPE DOCTOR.FEL. The full paragraph is printed on your terminal. Deleting the two lines had no permanent effect, because you left EDT using the QUIT command instead of the EXIT command. If you had used the EXIT command, the new version of the file would have been only two lines long and numbered version 3. (Version 1, the one-line version, and version 2, the four-line version, would still be in existence.)

Why would you use the QUIT command? If you wish to use EDT just to read a file, to search for some lines, for instance, you can QUIT when you are through because you haven't really done any permanent editing. If you start to edit and then change your mind, QUIT is also useful. Or, if you make a major error, such as deleting a large number of lines that you wish to keep, it may be simpler to QUIT and start over.

Remember though, if you type QUIT accidentally, you will lose all the editing work that you have performed. The EDT command QUIT is one EDT command that can cause you to lose work if you use it when you don't mean it.

Line Mode Editing

Line mode has all the editing functions that character mode does. Instead of using keypad functions, line mode uses English-like commands. For example, INSERT, DELETE, FIND, and MOVE are all line mode commands. Line mode commands are typed at the asterisk prompt. They are entered by pressing the RETURN key or the keypad key labeled ENTER. You can find more information about all line mode editing commands in Chapter 4 of the *Micro/RSX User's Guide*.

To begin, type and enter EDIT DOCTOR.FEL. The first line of text appears on the screen; notice that it is numbered.

```
$ EDIT DOCTOR.FEL [RET]
1   I do not like you, Doctor Fell,
*
```

When you created DOCTOR.FEL, you supplied no line numbers, but when you brought the file into EDT, line numbers were assigned. These line numbers are used only by EDT. They are not a part of your file. When you return to DCL, you leave the line numbers behind. Because keypad editing operates on the whole file, it doesn't use the line numbers EDT assigns. However, because line mode editing operates on the file a line at a time, it depends on them.

Type TYPE 1 and enter it. EDT prints line 1 and returns the prompt.

Now type T 1 and enter it. Line 1 is printed again.

Note

EDT has stricter rules for shortening commands than DCL. Type commands exactly as shown in the examples.

Range Specifications

Line mode can display all or part of a buffer. The lines displayed are selected by range specifications; range specifications can be line numbers or descriptive words.

Type T WHOLE and enter it. The entire buffer (four lines) is printed on your terminal, with line numbers.

Now type T BEGIN and enter it. EDT returns to the beginning of the buffer and prints line 1. EDT uses an invisible *line pointer* to keep track of where you are in a buffer. When you move from one place to another in a buffer, you are moving the line pointer.

Now type TYPE END and enter it. The line pointer moves to the end of the buffer and EDT displays [EOB].

Now type T END -1 and enter it. The line pointer moves "up" one line and EDT displays the last line.

The expressions BEGIN, END, and END -1 are all ways of specifying a *range* for the TYPE command.

Getting Help in Line Mode

The EDT HELP command provides help on EDT. Typing HELP on a line by itself gives information on the HELP command and lists other EDT help available. Since you've just used some simple forms of EDT ranges, you can now try some other ways to specify ranges in EDT.

Type HELP RANGE and enter it. After a pause, EDT displays quite a bit of text explaining the different ways of expressing a range. What you have been doing is combining range specifications with the TYPE command to specify the lines you wish listed. Notice that there are many more forms of range expression besides those you have already used.

```
* HELP RANGE RET
```

```
RANGE

Range specifications are used on most line editing commands to select
the exact lines of text on which the command will operate.

There are several general classes of range specifications:

    1.  Single line ranges specify a single line of text.

    2.  Multiple line ranges specify blocks of text, such as an
        entire buffer or all lines from the current line to the end
        of the buffer.

    3.  Compound ranges combine single line ranges with operators
        to specify multiple lines of text.

    4.  Noncontiguous ranges specify multiple lines that are not
        necessarily adjacent to one another.

Additional information available:

ALL         AND         BEGIN       BEFORE      BUFFER      DOT         END
FOR         LAST        MINUS       NUMBER      ORIGINAL    PLUS        REST
SELECT      STRING      THRU        WHOLE
*
```

Now type HELP RANGE MINUS and enter it. EDT displays text explaining how the minus (-) is used in range specifications. Similar help is available for all EDT line mode commands, as well as for concepts such as RANGE.

```
* HELP RANGE MINUS RET

RANGE

MINUS

The minus sign in ranges selects a single line that is a specified
number of lines before a specified line.

Format: [range] - [n]

Range is a single line range, and n is an integer.  The line selected
is the line that is n lines before the line specified by range.  If
you omit range, the current line is used; if you omit n, 1 is used.

Ex:     TYPE 15 - 3     Type the third line before the line numbered 15.
        TYPE END -1     Type the last line in the buffer.
        TYPE -          Type the previous line.
*
```

As you go through this editing exercise, use the HELP command whenever you want further information. Type HELP on a line by itself for information on what help is available from EDT.

Displaying Lines of Text

As you have seen, the TYPE command moves the line pointer to the beginning of a range. There are many ways of expressing ranges.

Type T 1 and enter it. Line 1 is printed on your terminal. The TYPE command moved the line pointer to line 1 and printed it.

Now type 1 by itself and enter it. Once again, line 1 is printed on your terminal. This has the same effect as the previous command. If the range expression for a type command begins with a number, you need not type TYPE or T.

```
* T 1 [RET]
    1          I do not like you, Doctor Fell,
* 1 [RET]
    1          I do not like you, Doctor Fell,
```

Press RETURN on a line by itself. Line 2 is printed. Line 2 is the next line past the line pointer.

Type T . and enter it. Line 2 is printed. The dot (.) is a range expression meaning "where the line pointer is."

Now type a dot (.) and enter it. Line 2 is printed. The line pointer has not moved. The dot is considered a line number. Thus, you did not have to type the T.

```
* [RET]
    2          The reason why I cannot tell,
*T . [RET]
    2          The reason why I cannot tell,
*. [RET]
    2          The reason why I cannot tell,
```

This time type TYPE 1 THRU 2 and enter it. Both lines are printed.

Type 1 THRU 2 and enter it. The range begins with a line number. No TYPE command is needed.

Again, type and enter the dot. Although EDT printed both lines in the range, the line pointer is still pointing at the first line in the range. The dot will always tell you where the line pointer is.

```
*T 1 THRU 2 [RET]
    1        I do not like you, Doctor Fell,
    2        The reason why I cannot tell,
*1 THRU 2 [RET]
    1        I do not like you, Doctor Fell,
    2        The reason why I cannot tell,
*. [RET]
    1        I do not like, Doctor Fell,
```

Now type TYPE WH and enter it. The entire buffer is printed.

Finally, type WH and enter it. You get an error message, "Unrecognized command." WH is a range expression that does not begin with a number, so it does not work without the TYPE (or T) command. Since you entered an illegal command, nothing happened. You received the message and the asterisk (*) prompt returned. Your text is unaffected, and the line pointer stays in place.

```
*T WH [RET]
    1        I do not like you, Doctor Fell,
    2        The reason why I cannot tell,
    3        But this I know and know full well,
    4        I do not like you, Doctor Fell.
[EOB]
*WH [RET]
 ^
Unrecognized command
*
```

Inserting Text

The INSERT command (abbreviation I) inserts text ahead of the line pointer.

The RESEQUENCE command (abbreviation RES) renumbers lines.

Type 1 and enter it. Line 1 is printed on your terminal.

Now type I (for INSERT) and enter it. Type the new line of text shown in the example, and then end the insertion by pressing CTRL/Z on a line by itself.

```
*1 [RET]
    1        I do not like you, Doctor Fell,
*I [RET]
             All around the mulberry bush, [RET]
             [CTRL/Z]
    1        I do not like you, Doctor Fell,
```

Now type T WH and enter it. The new line you entered appears ahead of line 1. The line pointer was pointing to line 1 when you issued the INSERT command. The INSERT command inserts text ahead of the line pointer. (If the line pointer had been pointing at the end of the buffer, the new text would have been inserted at the end of the file.)

```
*T WH [RET]
    0.1       All around the mulberry bush,
    1         I do not like you, Doctor Fell,
    2         The reason why I cannot tell,
    3         But this I know and know full well,
    4         I do not like you, Doctor Fell.
[EOB]
```

Notice that the new line number is line 0.1. EDT keeps your lines in numerical order by using numbers with decimal points when you insert new material between existing lines. Since these numbers can become confusing after a complicated series of inserts, EDT provides a means of resequencing line numbers.

Renumbering Text Lines

Type RESEQUENCE and enter it. Now type T WH and enter it. The lines have been renumbered.

```
*RESEQUENCE [RET]
5 lines resequenced
* T WH [RET]
    1         All around the mulberry bush,
    2         I do not like you, Doctor Fell,
    3         The reason why I cannot tell,
    4         But this I know and know full well,
    5         I do not like you, Doctor Fell.
[EOB]
*
```

Deleting Lines from Text

The DELETE command eliminates text.

Move the line pointer to line 1 and print it.

```
*1 [RET]
    1       All around the mulberry bush,
```

Now type DELETE 1 and enter it. EDT informs you of the deletion and prints the next line on your terminal.

Type T WH again and enter it, and you will see that the excess line is gone.

```
*DELETE 1 [RET]
1 line deleted
    2       I do not like you, Doctor Fell,
*T WH [RET]
    2       I do not like you, Doctor Fell,
    3       The reason why I cannot tell,
    4       But this I know and know full well,
    5       I do not like you, Doctor Fell.
    [EOB]
*
```

Type and enter RESEQUENCE to renumber the lines correctly.

Searching through Text

The FIND command moves the line pointer past the string you are searching for.

You can search for a string by quoting it. You can search again for the same string by typing just the "quotes."

You can search for a group of lines by quoting from the beginning of the first line and the end of the last line.

You can move the line pointer with plus (+) and minus (-) commands.

Type FIND BEGIN and enter it. The EDT prompt returns, but nothing else is printed on your terminal. The line pointer is now at the beginning of the buffer. The FIND command moves the line pointer without printing the line. The TYPE command moves the line pointer and also prints the line.

Type "Fell," including the quotes and enter it. The line containing the quoted string is printed on your terminal. Now do the same thing again. EDT reports that the string was not found and reprints the line. When EDT finds a string, the line pointer moves past that string. When EDT cannot find a quoted string, it reprints the last line pointed to. This tells you that the line pointer has not moved. This may seem confusing, but as you use EDT, particularly with larger files, you will find it less so.

```
*FIND BEGIN [RET]
*"Fell," [RET]
    1        I do not like you, Doctor Fell,
*"Fell," [RET]
String was not found
    1        I do not like you, Doctor Fell,
```

Now type F BE and enter it. F is the abbreviation for the FIND command and BE is the abbreviation for the BEGIN range expression. The EDT prompt returns, but you see nothing else. The line pointer is at the beginning of the buffer again.

Type two quotation marks ("") with nothing between them and press RETURN. The first line is printed. EDT remembers the last string that you searched for and searches for it again when you type the "quotes" with nothing between them.

Return to the beginning of the buffer with F BE.

Now type "I do" THRU "Fell." and enter it. Notice that this time the period is included in the second string. The entire poem is printed on your terminal. The line pointer has not moved, however, as you can confirm by typing a dot (.) and entering it. The dot means "the current line."

```
*F BE [RET]
*"" [RET]
*"I do" THRU "Fell." [RET]
    1        I do not like you, Doctor Fell,
    2        The reason why I cannot tell,
    3        But this I know and know full well,
    4        I do not like you, Doctor Fell.
*. [RET]
    1        I do not like you, Doctor Fell,
```

You can also move the line pointer down using the plus (+) command or up using the minus (-) command as shown in the example. You can also combine the plus or minus with other range expressions in commands, such as TYPE BEGIN +1, or TYPE "reason" + 1, or TYPE END -1. END-1, by the way, is the last line in the buffer, since END is the actual end of the buffer, meaning there is no line there.

```
*+2 [RET]
                But this I know and know full well,
*-1 [RET]
                The reason why I cannot tell,
*
```

Moving and Copying Text within the File

The MOVE command moves text.

The COPY command copies text.

Type 1 and enter it. Line 1 is printed.

Now type MOVE 1 TO END and enter it. EDT informs you that one line has been moved.

Type RES (RESEQUENCE) and enter it.

Print the whole buffer by typing T WH and entering it.

```
*1 [RET]
    1       I do not like you, Doctor Fell,
*MOVE 1 TO END [RET]
1 line moved
*RES [RET]
4 lines resequenced
*T WH [RET]
    1       The reason why I cannot tell,
    2       But this I know and know full well,
    3       I do not like you, Doctor Fell.
    4       I do not like you, Doctor Fell,
[EOB]
*
```

Now type COPY 4 TO 1 and enter it. The command means "Make a copy of line 4 and place it just ahead of line 1." Resequence again and print the whole buffer using the T WH command.

```
*COPY 4 TO 1 [RET]
1 line copied
*RES [RET]
4 lines resequenced
*T WH [RET]
    1       I do not like you, Doctor Fell,
    2       The reason why I cannot tell,
    3       But this I know and know full well,
    4       I do not like you, Doctor Fell.
    5       I do not like you, Doctor Fell,
[EOB]
*
```

Notice that the COPY command leaves the copied line in place, while the MOVE command deletes the line from one location and places it in another location.

Type and enter DELETE 5 to restore the poem to its original form.

Replacing Words

The SUBSTITUTE command replaces text. The SUBSTITUTE command makes its substitution on the first matching string it encounters on the current line. You can also make substitutions throughout a range or throughout a whole file.

The SUBSTITUTE command searches for a *string* of text and replaces that string with a new string. The old and new strings are marked by slashes (/), or *delimiters.*

Begin this example by moving the line pointer to line 1. Type and enter S/ell/umble/. EDT searches the line for the string "ell", makes the substitution, and prints the new line.

```
1 [RET]
* S/ell/umble/ [RET]
    1       I do not like you, Doctor Fumble,
    1 substitution
```

Now type S/ell/umble/WHOLE and enter it. WHOLE means the whole buffer, as usual. EDT searches for the string "ell" throughout the buffer, making the substitution and printing each new line. When EDT has completed its operations, it informs you of the number of substitutions made.

```
*S/ell/umble/WHOLE RET
    2       The reason why I cannot tumble,
    3       But this I know and know full wumble,
    4       I do not like you, Doctor Fumble.
3  substitutions
*
```

Return to line 1 and substitute "ell" for "umble," to restore the original poem.

Line Mode Commands Also Used in Character Mode

WRITE and INCLUDE are two line mode commands that have no direct character mode equivalents. They are used in both line and character editing.

Remember that you can use the COMMAND (PF1 + 7) function in character mode to enter line mode commands.

Creating a Second File

The WRITE command takes a specified range of text and creates a new file containing that text. This command can be useful if you are going to use a section of a file in more than one piece of writing, for example in several different reports.

This example creates a file named LIKE.DOC that contains the first two lines of DOCTOR.FEL. The two lines are not deleted from DOCTOR.FEL.

```
* WRITE LIKE.DOC 1 THRU 2 RET
DUO:[USER]LIKE.DOC;1  2 lines
```

Including a Second File in Your Text

The INCLUDE command inserts a file into your existing file at the point you specify. The default position is at the line pointer. This example moves the line pointer to the end of your file and inserts the file LIKE.DOC.

```
*END [RET]
[EOB]
* INCLUDE LIKE.DOC
*T WH [RET]
    1       I do not like you, Doctor Fell,
    2       The reason why I cannot tell,
    3       But this I know and know full well,
    4       I do not like you, Doctor Fell.
    5       I do not like you, Doctor Fell,
    6       The reason why I canot tell,
[EOB]
*
```

The file LIKE.DOC still exists; you could include it in other files.

Summary

This concludes your introduction to EDT. You have learned to use the most common editing functions in both character and line mode:

- Creating new files and new versions of files
- Moving around in a text file
- Inserting and deleting text
- Moving and copying text
- Making local and *global* substitutions
- Using range specifications
- Getting help

EDT has additional capabilities that have not been explained here. See Chapter 4 of the *Micro/RSX User's Guide* for more information. This chapter details all these advanced features:

1. Multiple buffers that permit you to work on smaller blocks of text while you build a large file in the main buffer. If, for instance, you have text that must be repeatedly inserted in a file—"boilerplate" of some kind—you can store that text in an alternate buffer and use the COPY command to insert it wherever you need it.

2. A *journaling* facility that protects you against losing your files should the system crash while you are editing.

3. A means of defining new commands in line mode and new keys in character mode. Any time you need to do one series of editing commands over and over again, you should consider defining keys or commands.

4. More information on EDTINI.EDT files for saving those defined commands and using them over and over without having to redefine them every time.

EDT is quite versatile, and this introduction is only a beginning. Once you get used to using EDT as taught here, you should explore the EDT help files and Chapter 4 of the *Micro/RSX User's Guide* for ideas and suggestions on how you can use the advanced features of EDT.

Chapter 3
Using DCL Commands

Micro/RSX has hundreds of available commands and qualifiers. These commands enable you to specify in precise detail exactly what you want the system to do. In most cases, however, complex (and confusing) commands are not necessary. This chapter describes the Micro/RSX DCL commands most likely to be of everyday use to an average system user. The Micro/RSX help files and the *Micro/RSX User's Guide* have detailed descriptions of all the DCL commands, qualifiers, and concepts.

So far you have not used any qualifiers with DCL commands. Qualifiers are attached to a command by a slash (/) and are used to change the effect of the command. For instance, if you type DIRECTORY, you'll see a three-column listing of all the files in the USER account, but if you type DIRECTORY/BRIEF, you'll see a one-column listing of all the files.

Most DCL commands require you to name a file. Just as a book can contain anything that can be printed, a file can contain anything that can be put on a computer disk or put into computer memory. A file can contain a memo, or it can contain a computer program.

All files, regardless of their contents, have a lot in common. In particular, all files have names in the same form. The file name can be no more than nine characters and is usually followed by a three-letter file type. The name of a file generally tells you quite a bit about what is in the file. For instance, there is a file in the USER account called WHATSHERE.TXT. This text file is an annotated list of all the files included in the USER account.

This chapter explains more about how to use DCL and summarizes the most often-used DCL commands. The chapter opens with a discussion of *wildcards*, which are tools for specifying files in groups.

The next section discusses file management. DCL provides many commands for managing your files. You have done some file management already with the DIRECTORY and TYPE commands. You have also created files using the EDIT and CREATE commands. Other DCL commands for managing files include RENAME, COPY, DELETE, PURGE, and PRINT.

Following the file management section are descriptions of general system commands like BROADCAST, RUN, SET DEFAULT, SHOW DEFAULT, SHOW DEVICES, SHOW TIME, SHOW USERS, SET PASSWORD, and HELP.

The next section explains the commands for using disks and tapes, including how to prepare new diskettes for use. Commands in the section include MOUNT, ANALYZE/MEDIA, DISMOUNT, and INITIALIZE.

The final section describes briefly how to use the Queue Manager, including the PRINT command, as well as SHOW QUEUE, DELETE/ENTRY, and SET QUEUE.

Wildcards and Other Wild Things

Directories can be quite large. Most of the time when you look in a directory, you don't want to see the whole list, but only to see if certain files are available. If you want to check on a single file, you can type a command like this:

```
$ DIRECTORY FLY.TXT [RET]
```

The command lists only the most recent version of FLY.TXT.

The following command line lists all versions of FLY.TXT:

```
$ DIR FLY.TXT;* [RET]
```

The asterisk (*) is called a wildcard and functions exactly as a wild card does in card games. It can stand for anything. In this case, the asterisk means "all version numbers." Without the asterisk, you would have had to type this to list all three:

```
$ DIR FLY.TXT;1,FLY.TXT;2,FLY.TXT;3 [RET]
```

And even then you couldn't be sure you'd listed all versions.

For many commands, you can use wildcards in place of most fields in the file specification. The examples in this section use the DIRECTORY command, but you can use wildcards on all the commands discussed under file management in the next section, as well as some other commands.

Some of this is going to look complicated, but once you get used to it, you'll find it very useful. For instance, the following command asks for "the latest versions of all files with the file type .TXT":

```
$ DIR *.TXT [RET]
```

The next command asks for "all files in all directories on the disk." Try it. You'll probably want to press CTRL/C to stop it.

```
$ DIR [*]*.*;* [RET]
```

You cannot use wildcards for device names, but you can use them for directory names, file names, file types, and version numbers.

Wildcards can get wilder. If you type the following command line, you're asking for "the latest version of all .TXT files whose names start with F":

```
$ DIR F*.TXT [RET]
```

You're asking for "the latest version of all .TXT files whose names end with F", if you type this:

```
$ DIR *F.TXT [RET]
```

And, if you type this command line, you're asking for "the latest version of all .TXT files whose names include an F":

```
$ DIR *F*.TXT [RET]
```

Combining letters and the asterisk wildcard only works on file names and file types.

Another wildcard for file names and file types is the percent sign (%). The percent sign is a wildcard that stands for a single character. Therefore, to ask for "the latest version of all .TXT files whose names start with FL and end with one other character," you use this command:

```
$ DIR FL%.TXT [RET]
```

As you can see, wildcards are a way of specifying lots of files, or narrowing down the number of files you have to look at, without doing lots of typing or thinking. You'll find more information on wildcards in Chapter 5 of the *Micro/RSX User's Guide*.

There are also some DCL command qualifiers that help you specify groups of files without knowing anything about their exact names. These qualifiers can be combined with wildcards in commands.

Many of these qualifiers relate to when the files in your directory were created. You can show all the files created on or since a specified date, or during a specified time period.

To ask for all the files you created today, type the following:

```
$ DIR/TODAY [RET]
```

To ask for "all the files I created on April 12, 1985," type:

```
$ DIR/DATE:12-APR-85 [RET]
```

The qualifier /SINCE:28-FEB-85 means you're asking for "all the files I created on or after February 28, 1985."

The /THROUGH qualifier allows you to use /THROUGH:03-SEP-85 to mean "all files created on or before September 3, 1985."

You can use /SINCE and /THROUGH together to ask for files created between two dates:

```
$ DIR/SINCE:14-MAR-85/THROUGH:01-NOV-85 [RET]
```

And, finally, there is the /EXCLUDE qualifier. Use this if you want to see all your files *except* the file you specify. For example, to ask for "all files in the directory except those named FLU.TXT," type:

```
$ DIR/EXCLUDE:FLU.TXT;* [RET]
```

These qualifiers can be combined with wildcards. You can ask for "all the .TXT files I created today" by typing this:

```
$ DIR/TODAY *.TXT;* [RET]
```

The more systematic you are about naming your files, the more useful you'll find wildcards and these special DCL qualifiers.

Note

The next section discusses other file management commands that accept wildcards and special qualifiers. Wildcards and special qualifiers are easiest to use on DIRECTORY and TYPE. You should wait until you are confident of your ability to use them before trying them out on more complex commands, such as RENAME or COPY. This is especially true for DELETE and PURGE.

DCL Commands for File Management

This section describes Micro/RSX file management commands in alphabetical order: COPY, CREATE, CREATE/DIRECTORY, DELETE, DIRECTORY, EDIT, PURGE, RENAME and TYPE.

COPY

The COPY command makes a duplicate of one or more files. The file that you create with COPY does not have to have the same name as the original file.

Use COPY to move a file from the fixed disk to a diskette or tape, or from one diskette to another, or from one directory to another. You may also want to copy a file that you want to edit and use for a different purpose, such as a form letter.

This example duplicates the file HELLO.TXT; the new copy of the file has the name MESSAGE.TXT:

```
$ COPY [RET]
From?  HELLO.TXT [RET]
To? MESSAGE.TXT [RET]
$
```

If you have just looked at a file from the fixed disk and wish to make a copy on your diskette, use a command like this:

```
$ COPY [RET]
From? GOODIE.TXT [RET]
To? DU1:[JOHNSON]GOODIE.TXT [RET]
$
```

You will then have two copies of the file, one on the fixed disk and one on DU1: in the directory named [JOHNSON]. Naturally, there must be a directory named [JOHNSON] on the diskette volume in DU1: for this command to work.

(Later in this chapter you'll find a description of the CREATE/DIRECTORY command that shows you how to create a directory to do this.)

Two qualifiers to COPY are quite useful:

/OWN
/REPLACE

The /OWN qualifier specifies that the new copy of the file belongs to the directory you sent it to, and not to the directory you copied it from. This helps keep the *protection* of the file straight. If you encounter confusion about the protection of copied files, copy the file again using this qualifier, and you'll probably be all right.

There is a full discussion of file protection in Chapter 5 of the *Micro/RSX User's Guide*. For the time being, all you need to know is that if you are making a copy of the file for yourself, you won't need to use the /OWN qualifier, but if you are making a copy for someone else, you will need to use it. Otherwise, the person you're making the file for may have trouble editing or deleting the file.

The /REPLACE qualifier specifies that if the directory you are sending the copy to already has a file of that name, the old file will disappear and be replaced by the new copy you have just sent.

CREATE

The CREATE command lets you create a text file without using the editor. After you issue the command, type the text of the file on your terminal. You can delete a line with CTRL/U. You can close the file by pressing CTRL/Z.

For example:

```
$ CREATE [RET]
File? MAMOU.TXT
Why did you go away and leave me in Big Mamou?
^Z
$
```

You can create a file on a fixed disk or on a diskette. For instance, if your default directory is on the fixed disk, you could create a file on a diskette either by changing your default, or by specifying the diskette drive as part of the response to the **File?** prompt.

You will almost always have a directory on the fixed disk, but you will often have to create a directory on a diskette before you can create a file to put in it. See the description of CREATE/DIRECTORY, next.

CREATE/DIRECTORY

The CREATE/DIRECTORY command adds a directory to a fixed disk or to a diskette. Nonprivileged users can create a directory on a diskette drive if it is mounted with the qualifier /NOSHAREABLE. Privileged users can create a directory on any mounted volume. See the discussion of MOUNT later in this chapter.

If you name no disk, the directory is created on your current default disk. If you name no directory, the directory will have the same name as your default directory.

For example, if your default device is DU0: and your default directory is [USER], the following command creates a directory called [USER] on DU2:. UFD (User File Directory) is a synonym for directory.

```
$ CREATE/DIRECTORY [RET]
Device and UFD? DU2: [RET]
$
```

Note that for this command to work you must be a privileged user or the device DU2: must be mounted using the /NOSHAREABLE qualifier.

Remember that a directory must exist before you can copy files into it. Later in this chapter you'll read more about how to move your work around from the fixed disk to diskettes or tape. You should read this discussion, which appears in the section "DCL Commands for Disks, Diskettes, and Tapes" in this chapter, before creating any directories.

DELETE

The DELETE command erases one or more files from a directory. Once you delete a file, it is gone forever. That makes DELETE the most dangerous command you've learned so far. If you are deleting a whole list of files, pressing CTRL/C may stop the last ones on the list from being deleted, but it will not save the first ones.

DELETE works fine with wildcards and the special qualifiers, but you should be sure you know what you're deleting. You may want to look at a directory before deleting from it. For instance, type:

```
$ DIRECTORY/TODAY *.TXT;* [RET]
```

before typing:

```
$ DELETE/TODAY *.TXT;* [RET]
```

You may have forgotten about something that you want to keep.

You have to specify a version number with the name of the file you are deleting. You can use a wildcard, if you are deleting all versions of a file. If you don't specify a version number, Micro/RSX prompts you with each version of the file and asks whether you want to delete it.

There are two useful qualifiers to DELETE.

/LOG
/QUERY

The /LOG qualifier lists the names of files as they are deleted. This gives you a list of what you deleted, but no chance to change your mind.

The /QUERY qualifier gives you a second chance. It allows you to delete selectively after specifying a group of files. You are prompted with a list of file names, based on your original command. As each file in the list is named, you are asked whether you want to delete it.

The possible responses are as follows:

Y—delete the file
N—save the file
Q—save the file and quit
G—go ahead and delete all remaining candidates
RET —save the file

Y, N, and RETURN will continue with the next possible file, unless you press CTRL/Z, which stops all further deleting.

Here's an example, issued with the defaults of DU1:[BERRY]:

```
$ DELETE/QUERY *.TMP RET
Delete file   DU1:[BERRY]NADINE.TMP;1          [Y/N/G/Q]? Y RET
Delete file   DU1:[BERRY]16.TMP;1              [Y/N/G/Q]? Y RET
Delete file   DU1:[BERRY]CHUCK.TMP;1           [Y/N/G/Q]? N RET
Delete file   DU1:[BERRY]DUCK.TMP;1            [Y/N/G/Q]? G RET

The following files have been deleted:
DU1:[BERRY]HAVANA.TMP;1
DU1:[BERRY]MOON.TMP;1
DU1:[BERRY]ROLLOVER.TMP;1
DU1:[BERRY]BEETHOVEN.TMP;1
$
```

The user was able to delete the first two files one at a time with the Y response, save the third file with the N response, and delete all remaining files with the G response.

CAUTION

Don't use DELETE unless you mean it. In particular, don't use DELETE with wildcards or special qualifiers until you're sure of what you're doing.

DIRECTORY

The DIRECTORY command lists files in your directory. Remember that if you do not name a particular file or group of files with the DIRECTORY command, you will see a listing of all the files in the directory. If you name one file, then the directory listing is limited to that file.

Several qualifiers are available for DIRECTORY:

/BRIEF
/FREE
/FULL
/OUTPUT:filespec
/PRINTER
/SUMMARY

The DIRECTORY command lists the files in your directory. Normally, this is a three-column listing. The three columns show you the complete file names, the number of blocks used by the file, and the creation date of the file. At the end of the directory listing is a summary listing of the total number and space requirements of all files listed. If you use the /BRIEF qualifier, you get a one-column listing, showing only the file names. If you use the /FULL qualifier, you get a great deal of information about each file.

The /FREE qualifier shows you the amount of free space on the fixed disk or a diskette you specify. For example:

```
$ DIRECTORY/FREE DU2: [RET]
DU2: has 405. blocks free,  395. blocks used out of 800.
Largest contiguous space = 210. blocks
12. file headers are free, 36. headers used out of 48.
$
```

Task image files, which are runnable programs, need contiguous space, as do certain other kinds of files. Since each file has a header, the number of free headers is the number of additional files you can make.

Since diskettes have limited space on them compared to the fixed disk, you should check occasionally to make sure you're not about to overrun the capacity of the diskette. Each *block* on a disk can hold 512 characters, or about 80 words of text. With 800 blocks, each diskette can hold about 64,000 words of text.

Use DIRECTORY/FREE to compare the size of the diskettes with the size of the fixed disk.

The /OUTPUT qualifier allows you to create a file containing the directory listing, instead of printing it on your terminal. Include the name you want the file to have with the /OUTPUT qualifier, such as /OUTPUT:FOLEY.LST. You can use this qualifier to keep directories on the fixed disk for each diskette you use.

The following example creates a file on the fixed disk; the file contains a copy of the directory DU1:[FOLEY].

```
$ DIR/OUTPUT:FOLEY.LST DU1:[FOLEY]
```

The /PRINTER qualifier allows you to print the directory on your printer, if you have one.

The /SUMMARY qualifier tells you how many files you have in your directory and how much space they take up.

EDIT

The EDIT command starts up EDT, the DIGITAL standard editor, which is used to create and edit text files. Editing files is discussed in Chapter 2 of this book.

Several qualifiers to EDIT are described in Chapter 4 of the *Micro/RSX User's Guide*; they can increase the flexibility and convenience of EDT.

PURGE

The PURGE command is very similar to the DELETE command, except that PURGE always leaves one or more copies of the file around. Normally, PURGE will delete all but the highest-numbered copy of a file. This command is very useful for cleaning up your disks.

For instance, if you edit a file, look at it, edit it again, look at it, change your mind, edit it again, and so forth, pretty soon you'll have a large number of files with the same name and different version numbers. Usually, you'll only want to keep the latest version. The following command allows you to delete all but the last version:

```
$ PURGE GROUCHO.TXT [RET]
```

There are two qualifiers to PURGE that you may want to use:

/KEEP:n
/LOG

The /KEEP qualifier allows you to specify that the last n versions (by number) be saved. In other words, instead of keeping just the latest copy and deleting all the others, you can keep two or more of the latest versions by using /KEEP.

The /LOG qualifier lists the names of the files deleted on your terminal.

The following command purges all the files in your directory:

```
$ PURGE *.* [RET]
```

This is a good command to issue at the end of the day if you've done a lot of editing or other file creating.

CAUTION

Don't use PURGE unless you mean it. In particular, don't use PURGE with wildcards or special qualifiers until you're sure of what you're doing.

RENAME

The RENAME command changes a file's name.

For example,

```
$ RENAME [RET]
Old file name? BROWNS.STL [RET]
New file name? ORIOLES.BLT [RET]
$
```

This changes BROWNS.STL (old file name) to ORIOLES.BLT (new file name).

Here's another example:

```
$ RENAME WRONG.TXT;* SONG.TXT;* [RET]
```

This changes all files named WRONG.TXT to the name SONG.TXT. All the version numbers stay in order. Other wildcards and special qualifiers will work with RENAME, but you should generally only rename one file at a time until you are confident of your ability to handle wildcards and special qualifiers.

TYPE

The TYPE command prints files on your terminal. You can use any combination of file specifications, wildcards, and special qualifiers with the TYPE command.

For example:

```
$ TYPE/EXCLUDE:FLU.TXT;* *.TXT;* [RET]
```

The TYPE command prints on your terminal all .TXT files in your directory, excluding all versions of FLU.TXT.

DCL Commands for General System Use

This section describes, in alphabetical order, a number of commands for general system use. These include BROADCAST, HELP, RUN, SET DEFAULT, SET PASSWORD, SHOW DEFAULT, SHOW DEVICES, SHOW TIME, and SHOW USERS.

BROADCAST

The BROADCAST command sends a one-line message to one or more terminals.

For instance, the following command sends a message to terminal TT2:.

```
$ BROADCAST  RET
To? TT2:  RET
Message? "Time for lunch."  RET
```

You can also broadcast to another user by user name:

```
$ BROADCAST  RET
To? BRANDO  RET
Message? "Let's go out for popcorn."  RET
```

In this case, the broadcast goes to all the terminals that user Brando happens to be logged in on.

If you leave the "quotes" off the message, it appears on the receiving terminal in UPPERCASE. With the "quotes," the message appears exactly as you sent it.

Privileged users can send messages to all terminals, or to all logged-in terminals, by using the following qualifiers:

/ALL
/LOGGED_IN

The following command sends the broadcast to every terminal that has power on, whether logged in or not:

```
$ BROADCAST/ALL  RET
Message? "Emergency meeting in main lobby."  RET
```

But this command sends the broadcast only to terminals with a user logged in:

```
$ BROADCAST/LOGGED_IN  RET
Message? "Everybody log out."  RET
```

HELP

The HELP command gives you information about using the system. Most Micro/RSX systems have help files available for users, although sometimes they may be found on a separate diskette instead of on the fixed disk.

H and ? are both abbreviations for the HELP command. You can also get help by typing ? in response to a prompt from a DCL command. In that case, after the help file appears on the screen, the prompt returns. You can either answer, or you can ask for more help by typing another question mark.

If you simply issue the HELP command at the $ prompt, you'll see a list of the available help files for your system. You can also ask for information on a specific command or qualifier by typing commands such as these:

```
$ HELP TYPE [RET]
$ HELP DIRECTORY BRIEF [RET]
```

Usually, each screen of help text will point you to further help text when it is available.

The EDT HELP command works similarly. (See Chapter 2.)

SET DEFAULT

The SET DEFAULT command sets either your default directory or device, or both. When you log in, you log in on a particular device, probably the fixed disk (DU0:), and in a particular directory, for example [KILROY]. Therefore, DU0:[KILROY] is your default. You can use the SHOW DEFAULT command to find out what your defaults are.

Whenever you name a file in a command, such as this:

```
$ PRINT  [RET]
File(s)? SLOE.GIN [RET]
```

the system assumes you mean:

```
$ PRINT [RET]
File(s)? DU0:[KILROY]SLOE.GIN [RET]
```

If you want to print a file from another disk or directory, you have to include the disk or directory name in your command. For instance:

```
$ PRINT [RET]
File(s)? DU2:[JANE]TOP.CAT [RET]
```

If you want to do a number of things with files from DU2:[JANE], use the SET DEFAULT command, as in this example:

```
$ SET DEFAULT DU2:[JANE] RET
$ PRINT TOP.CAT RET
```

When you're through with DU2:[JANE], you can go back to your original disk and directory with another SET DEFAULT command.

SET PASSWORD

The SET PASSWORD command changes your password. You must enter your old password before you are allowed to change it. You must enter your new password twice, the second time for verification. Type carefully when entering the password information, because it is not echoed. You wouldn't want to have an unknown password because your finger slipped. (If this does happen, the system manager can straighten it out.)

In the following example, the passwords are shown in brackets, but remember that they do not appear on your terminal.

```
$ SET PASSWORD RET
Old password: <FUDD>
New password: <WABBIT>
Verification: <WABBIT>
```

The next time you log in, you will have a new password.

SHOW DEFAULT

The SHOW DEFAULT command tells you your current default device and directory. It also tells you the type of directory you have and which terminal you are logged in on.

For example:

```
$SHOW DEFAULT RET
DUO:[USER] Named TT2:
Protection UIC:  [200,1]
$
```

Only the device and directory are important to know in order to use defaults in file specifications. The directory type, NAMED, is the default directory type. It indicates that your directory can use either the name or number format. The protection UIC (User Identification Code) identifies you to the system and controls what system privileges you have.

SHOW DEVICES

The SHOW DEVICES command tells you which devices are on your system and which are available. If you name a device type, only information about devices of that type is shown. For example, the following command gives information about the DU: devices on a system with one fixed disk and two diskette drives:

```
$ SHOW DEVICES DU: RET
DU0:     Public Mounted Loaded Type=RD51
DU1:     TT0: - Private Loaded Type=RX50
DU2:     Loaded Type=RX50
DU3:     Offline Loaded Type=unknown
$
```

DU0:, the fixed disk, is mounted public, which makes it accessible to all users. The device is an RD51, a type of hard disk; its driver is *loaded*, meaning it is present in memory so that it can be used at any time. This means the fixed disk is available for use by anyone.

DU1:, a diskette drive, is mounted nonshareable, or private. Only the user logged in on TT0: can use it. The device is an RX50, a type of diskette, and the driver is loaded.

DU2: is available for use with an RX50 diskette. It has not been mounted, which indicates it is not presently being used.

DU3: is reported to be off line and loaded, type unknown. Loaded means that the software is present to handle such a device if it exists. However, since the device is off line, the system knows nothing about it. In fact, this system may have no physical DU3: device.

If you issue the SHOW DEVICES command without naming any device, you'll see a list of all devices on the system, including terminals and *pseudo devices*. Pseudo devices are not physical devices. They are names used by the system as stand-ins for real device names. This makes it possible to refer to a device on any RSX system without knowing its name and number. On any RSX system, the operating system itself is always on pseudo device LB:, regardless of which physical device it might be. Similarly, your terminal is always pseudo device TI:, regardless of its number or model.

SHOW TIME

The SHOW TIME command displays the current time and date. The time is in 24-hour format, and the date is formatted as dd-mmm-yy.

For example:

```
$ SHOW TIME [RET]
15:27 03-SEP-85
$
```

SHOW USERS

The SHOW USERS command tells you which terminals are logged in, as well as providing some information about the user logged in to the terminal. The following display shows the terminal number, the directory, and the protection UIC (User Identification Code) for each user, followed by the login time, the number of active tasks, and the user's name.

```
$ SHOW USERS [RET]
TT2:      [FREDDY]     [7,40]    18-MAY-85 10:57 0      F. SANFORD
TT6:      [WAREHOUSE] [303,5]    18-MAY-85 15:04 3      R. ROGERS
$
```

RUN

The RUN command starts a task (or working program) executing.

For example, this command runs a program called QIX:

```
$ RUN QIX [RET]
```

Chapter 7 in the *Micro/RSX User's Guide* provides more information about running tasks.

DCL Commands for Disks, Diskettes, and Tapes

The commands in the next section help you move your work around from one disk to another, or from disk to tape. The section opens with a description of how to use the MOUNT command with diskettes and how to prepare a blank diskette for use. It then describes how to use the MOUNT command with a tape and how to prepare a blank tape for use.

Disk drives and tape drives on the MicroPDP-11 are *peripheral devices* used for information storage. Many MicroPDP-11s have at least three disk drives available, one with *fixed media* and two diskette drives with *removable media.* Other MicroPDP-11s have a drive for a fixed disk and a drive for a removable tape cartridge. You can't see the fixed disk, because it's inside the box, and since it is fixed in place, you can't change it. The diskettes and tape, on the other hand, are removable and readily available.

The fixed disk has a greater capacity and operates at a higher speed than the removable media. The operating system itself resides on the fixed disk. Whether you use the fixed disk or the removable media usually depends on how you are using the Micro/RSX system.

In general, the fixed disk holds information that needs to be immediately available, or *on line,* at all times. The operating system obviously falls in this category, but the fixed disk probably has most of the user directories on it as well.

The diskettes are slower and smaller than the fixed disk, but you can easily remove them or replace them. The diskettes generally hold information that you want to have on line for the moment, but that need not be on line all the time. One common operation is to edit a file on the fixed disk, where the speed is important, and then copy it to a diskette and delete it from the fixed disk once you've finished with it.

Tapes are not as fast as either diskettes or the fixed disk, but they have a greater storage capacity and are more economical. Tapes are commonly used to back up the files on a system. Then, if files on the fixed disk are accidentally deleted or destroyed, copies can be recovered from the tape.

You'll probably grow quite accustomed to moving files around from one medium to another. On Micro/RSX systems, a disk, diskette, or tape must be mounted before you can do anything with it.

MOUNT with Diskettes

The MOUNT command gives you access to a diskette in its drive. After placing the diskette in the drive, you must mount the volume. Every volume has a label. Often, this label is written right on the diskette package along with other identification. The volume label is like a password to allow you to see the information that is on the diskette.

There are four qualifiers to MOUNT that you may need:

/NOSHAREABLE
/SHAREABLE
/PUBLIC
/FOREIGN

If you want to keep the information on the volume to yourself, you should use the /NOSHAREABLE qualifier to MOUNT. You must mount the volume /NOSHAREABLE if you are going to create a directory.

If you need to share the information on the volume, use the /SHAREABLE qualifier. Diskette volumes are mounted /SHAREABLE by default. Everyone who wants to share in the information on the mounted volume will have to know the volume label, often called the Volume-ID.

Privileged users can use the /PUBLIC qualifier to MOUNT. When a disk is mounted public, everyone can use it. The fixed disk is usually mounted public.

The /FOREIGN qualifier allows you to access diskettes that are not in the format required by Micro/RSX. This qualifier is most commonly used when you are preparing a blank diskette for use. As supplied, the diskettes are not in Micro/RSX format. Therefore, when you mount them, you must use the /FOREIGN qualifier to the MOUNT command. See the following discussion for an example of this. You are always warned when you need to use the /FOREIGN qualifier with any Micro/RSX command.

You can also use the /FOREIGN qualifier to mount any disk or diskette that is not in Micro/RSX format. The /FOREIGN qualifier bypasses the Micro/RSX file system.

Note

The term volume is often used as a synonym for diskette. Strictly speaking, the diskette is the actual piece of plastic and the volume is the organized information on the diskette. In most cases, the difference is unimportant because you need both hardware and software to gain

access to the information on the diskette. Once the diskette is placed in a drive, you will probably think of the device as holding the information. Again, this is a loose definition, but it is important to remember that they are not the same thing.

Preparing a Blank Diskette for Use

Preparing a blank diskette for use involves two steps: checking for *bad blocks* and putting the diskette in the format Micro/RSX requires to read and write files. Bad blocks are usually caused by damage to the surface of the diskette. In general, you should not use a blank diskette with a bad block on it.

To prepare a blank diskette for use follow these steps:

1. Place the diskette in a drive. Remember that diskette drive numbers vary according to whether you have an optional fixed disk on your system. This example assumes you're using DU1:.

2. Issue the following command:

```
$ MOUNT/FOREIGN/NOSHAREABLE [RET]
Device? DU1: [RET]
$
```

3. Now, to check for bad blocks, issue the following command and wait for the message to return:

```
$ ANALYZE/MEDIA [RET]
Device? DU1: [RET]
BAD -- DU1: Total bad blocks = 0.
$
```

   If ANALYZE/MEDIA informs you it has found more than two bad blocks on the diskette, you will probably not want to use the diskette for anything important. In general, diskettes either have no bad blocks at all, or have too many to use. If you do not use a diskette, you should make a note of it on the diskette envelope.

   If the diskette has bad blocks, issue the following command:

```
$ DISMOUNT [RET]
Device? DU1: [RET]
14:41:30 *** DU1: -- Dismount complete
DMO -- TT0: dismounted from DU1: *** Final dismount initiated ***
$
```

Put the rejected diskette back in its envelope and set it aside. Start over with a new diskette.

4. When ANALYZE/MEDIA displays the message announcing zero bad blocks, issue the following command:

```
$ INITIALIZE [RET]
Device? DU1: [RET]
Label? CHAPTER1 [RET]
$
```

Wait for the $ prompt to return. Initializing a diskette makes it ready to accept files and directories. The label, sometimes called the Volume-ID, is, in effect, a password that you and others who want to use the diskette will use. The label can be up to 12 characters long. Unless the information on the diskette is meant to be kept secret, you can write the label on the envelope so you won't forget it. (Don't write on the envelope with the diskette inside it, or you may create bad blocks.)

5. Now issue the following two commands:

```
$ DISMOUNT [RET]
Device? DU1: [RET]
14:41:30 *** DU1: -- Dismount complete
DMO -- TTO: dismounted from DU1: *** Final dismount initiated ***
$ MOUNT/NOSHAREABLE [RET]
Device? DU1: [RET]
Label? CHAPTER1 [RET]
$
```

Now the diskette in DU1: is ready to use. You had to issue a DISMOUNT command before the MOUNT command, because once you have initialized a diskette, it is no longer foreign and must be remounted correctly.

6. The final step for readying a blank diskette is to create a directory for yourself on it. In the next example, the system prompts you for device and UFD. UFD (User File Directory) is a synonym for directory. Issue the following command:

```
$ CREATE/DIRECTORY [RET]
Device and UFD? DU1:[HAYES] [RET]
$
```

Now you can copy files into the [HAYES] directory on that diskette, and, in fact, do anything you can do with any other directory.

7. When you're finished with the diskette, issue this command:

```
$ DISMOUNT [RET]
Device? DU1: [RET]
14:41:30 *** DU1: -- Dismount complete
DM0 -- TT0: dismounted from DU1: *** Final dismount initiated ***
$
```

   Remove the diskette from the drive so that others can use it.

You may be thinking, "That certainly is complicated!" It is complicated, but there is an easy way around it. Whenever you have to use a complicated series of commands, you can simply create a file containing all the commands and let Micro/RSX issue the commands for you. See Chapter 4 on Indirect command processing for an explanation of how to do this.

Using a Diskette with Files on It

To prepare a diskette that already has files on it, follow these steps:

1. Find out the volume label for the diskette you wish to use. Then issue the following command:

```
$ MOUNT [RET]
Device? DU1: [RET]
Label? SPIDERS [RET]
$
```

2. As soon as the $ prompt returns, you can use the mounted diskette.
3. When you're finished with the diskette, issue this command:

```
$ DISMOUNT [RET]
Device? DU1: [RET]
14:41:30 *** DU1: -- Dismount complete
DM0 -- TT0: dismounted from DU1: *** Final dismount initiated ***
$
```

   Remove the diskette from the drive so that others can use it.

MOUNT with a Tape

The MOUNT command gives you access to a tape in its drive. After placing the tape in the drive, you must mount the volume. Every volume has a label of no more than six characters. Often, this label is written right on the tape cartridge along with other identification. The volume label is like a password to allow you to see the information that is on the tape. Tape volumes are organized into file sets. See Chapter 3 of the *Micro/RSX System Manager's Guide* for more information about file sets and volumes.

There is one qualifier to MOUNT that you will need:

/FOREIGN

The /FOREIGN qualifier allows you to access tapes that are not in ANSI format, the format that Micro/RSX generally uses. This qualifier is used when you are preparing a blank tape for use. As supplied, the tapes are not in ANSI format. Therefore, when you mount them for the first time, you must use the /FOREIGN qualifier to the MOUNT command.

You can also use the /FOREIGN qualifier to mount any tape that is not blank, but is not in ANSI format. The /FOREIGN qualifier bypasses the file system that Micro/RSX usually requires.

Note

The term volume is often used as a synonym for tape. Strictly speaking, the tape is the actual piece of plastic and the volume is the organized information on the tape. In most cases, the difference is unimportant because you need both hardware and software to gain access to the information on the tape. Once the tape is placed in a drive, you will probably think of the device as holding the information. Again, this is a loose definition, but it is important to remember that they are not the same thing.

Preparing a Blank Tape for Use

To prepare a blank tape for use, follow these steps:

1. Make sure that the tape cartridge is write-enabled. This means that the write-protect switch is pushed all the way to the right, allowing you to write files on the tape.

2. Place the tape in the drive. You will need to know the name of the drive you are using. If your system uses a TK50 tape, the name of your tape drive is MU0:. If your system uses any other kind of tape, your tape drive is MS0:. If you do not know which kind of tape you have, use the SHOW DEVICES command to get a list of the devices on your system.

3. Issue the following commands:

```
$MOUNT/FOREIGN [RET]
Device?MUO: [RET]
$INITIALIZE [RET]
Device? MUO: [RET]
Label? MELVIN [RET]
```

   Wait for the $ prompt to return. Initializing a tape prepares it to accept files. The label, sometimes called the Volume-ID, is, in effect, a password that you and others who want access to the tape will use.

4. Now issue the following two commands:

```
$ DISMOUNT [RET]
Device? MUO: [RET]
$ MOUNT [RET]
Device? MUO: [RET]
Label? MELVIN [RET]
$
```

   Now the tape in MU0: is ready to use. You had to issue a DISMOUNT command before the MOUNT command, because once you have initialized the tape, it is no longer foreign and must be remounted correctly.

5. You can now use the tape in many of the same ways you would a diskette. For example, you can copy files to the tape using the same commands that you use to copy files to a diskette.

6. When you are finished using the tape drive, issue the following command:

```
$ DISMOUNT [RET]
Device? MUO: [RET]
```

Remove the tape cartridge from the drive. Be sure to label the cartridge, so you will remember what is on it.

Note that backing up files is a common use of tapes. To use the Micro/RSX backup procedure, you do not need to have the tape in ANSI format. For information on backing up files, read Chapter 3 in the *Micro/RSX System Manager's Guide*.

DCL Commands for the Queue Manager

The Queue Manager, or QMG, is the system task that keeps track of jobs that have been directed to batch processors, printers, or other output devices, making sure that they are separate and in order.

Commands that involve the Queue Manager include the following: PRINT, SHOW QUEUE, SET QUEUE, DELETE/ENTRY, HOLD/ENTRY, RELEASE/ENTRY, and STOP/ABORT.

PRINT

The PRINT command prints files on a printer, if your system has one.

```
$PRINT [RET]
File(s)? WHATSHERE.TXT [RET]
PRI - Job 141, name "WHATSHERE", submitted to queue "PRINT "
$
```

Once you receive the message that the print job has been submitted to a queue, you can go on with your other work. The job name comes from the name of the first, or only, file. The job number is unique and can be used in other commands to the Queue Manager, such as SHOW QUEUE, SET QUEUE, DELETE/ENTRY, and STOP/PRINTER.

There are two particularly useful qualifiers to PRINT:

/AFTER:(dd-mmm-yy hh:mm)
/COPIES:n

The /AFTER qualifier allows you to print your job after a time you specify, perhaps at a time when no one is around. Without this qualifier, the job would go directly to the printer.

You may specify either the date, or the time, or both. If you do not specify a date, the current date is assumed. To specify the date without the time, omit the hh and mm values.

The date must be in the format shown here:

```
18-MAY-85
```

The month is indicated by the first three letters of its name.

The time must be in the 24-hour format.

Here are some examples.

Print the file after 6 P.M.:

```
$ PRINT/AFTER:(18:00) [RET]
File(s)? LONG.TXT [RET]
```

Print the file on April 1, 1985:

```
$ PRINT/AFTER:(1-APR-85) [RET]
File(s)? JOKE.TXT [RET]
```

The /COPIES qualifier lets you print more than one copy of the file.

Print two copies of the file:

```
$ PRINT/COPIES:2 [RET]
File(s)? RESUME.TXT [RET]
```

If you are printing more than one file at the same time, but only want extra copies of one, put the /COPIES qualifier after that file name.

Print two copies of the file RICE.TXT and one copy of the other two files:

```
$ PRINT [RET]
File(s)? BASEBALL.TXT, RICE.TXT/COPIES:2, YAZ.TXT [RET]
```

Or, if necessary, you can carry this a bit further.

Print one copy of the first file, two of the second, and three of the third:

```
$ PRINT [RET]
File(s)? HUEY.TXT,DEWEY.TXT/COPIES:2,LOUIE.TXT/COPIES:3 [RET]
```

The following commands can be used for both print and batch jobs. The SUBMIT command, described in the next chapter, places jobs in batch queues just as the PRINT command places jobs in print queues.

SHOW QUEUE

SHOW QUEUE displays information about print or batch jobs in queues, such as where they are and what their entry numbers are.

The SHOW QUEUE command by itself lists full information on all jobs in all queues.

If you want brief information on all jobs in all queues, use the following command:

```
$ SHOW QUEUE/BRIEF [RET]
```

If you want information on a particular job, type a command like the following:

```
$ SHOW QUEUE/ENTRY:141 [RET]
```

SET QUEUE

SET QUEUE allows you to change attributes of print or batch jobs after they have been placed in a queue.

The following example shows you how you can print two copies of the file HOLIDAY.LIS, even though you did not specify two copies in the initial PRINT command. The /FILE_POSITION qualifier refers to the position of the file within this job.

```
$ PRINT [RET]
File(s)? WHATSHERE.TXT,HOLIDAY.LIS,JUNE.DAT [RET]
PRI - Job 141, name "WHATSHERE", submitted to queue "PRINT "
$

$ SET QUEUE/ENTRY:141/FILE_POSITION:2/COPIES:2 [RET]
$
```

The SET QUEUE command works for most of the qualifiers to the PRINT and SUBMIT commands.

DELETE/ENTRY

DELETE/ENTRY removes an entry from a queue. If you change your mind after issuing a PRINT or SUBMIT command, use DELETE/ENTRY. The following example shows how you can delete a print job, even though you have already submitted it to the print queue.

```
$ PRINT  [RET]
File(s)? WHATSHERE.TXT [RET]
PRI - Job 141, name "WHATSHERE", submitted to queue "PRINT "
$

$ DELETE/ENTRY:141 [RET]
$
```

HOLD/ENTRY

If you submit a print or batch job to a queue and then wish to delay processing for some reason, use the HOLD/ENTRY command. The following example shows how you can delay processing of a job after you have submitted it to the print queue.

```
$ PRINT  [RET]
File(s)? WHATSHERE.TXT [RET]
PRI - Job 141, name "WHATSHERE", submitted to queue "PRINT "
$

$ HOLD/ENTRY:141 [RET]
$
```

RELEASE/ENTRY

Use the RELEASE/ENTRY command when you are ready to begin processing a job that you have delayed with HOLD/ENTRY. The following example releases the job that was held up in the previous example.

```
$ RELEASE/ENTRY:141 [RET]
$
```

STOP/ABORT

Finally, if you want to cancel the currently active print job, you can issue the following command:

```
$ STOP/ABORT LP0: RET
```

or

```
$ STOP/ABORT TT1: RET
```

You have to know the name of the physical device serving as your system's printer (for example, LP0: or TT1:) to use the STOP/ABORT command.

There are many more qualifiers to all the Queue Manager commands. See Chapter 8 in the *Micro/RSX User's Guide* for more information.

Chapter 4
Automatic Command Entry

Often while working on the system, you need to use the same command or sequence of commands repeatedly. It is tiresome to retype the same commands each time you need them. With complicated commands, typing mistakes are particularly annoying. Micro/RSX includes a command processor (called *Indirect*) and a *batch processor* to pass commands automatically to the system.

With both the command processor and the batch processor, you place the commands or series of commands you want executed in a file and pass the file to the system for processing. Otherwise, the two processors are very different.

In particular, the Indirect Command Processor works from a logged-in terminal, while a batch job logs itself in, which allows you to use your computer when you're not even there. The batch job can be scheduled to run at a particular time, while Indirect runs immediately. Batch jobs can be scheduled by priority, while Indirect always runs at the same priority. Batch jobs provide you with a *batch log* indicating how the job ran. Indirect includes a complete programming language, while batch jobs provide only slight programming capability.

These two methods of automatic command entry can easily work together. A batch job can include a command to run Indirect, and Indirect can issue a SUBMIT command to run a batch job.

The following sections describe more about each processor. The material in this chapter is more difficult. Study it carefully and then create your own examples.

Indirect Command Processing

Indirect (the Indirect Command Processor) lets you execute several DCL commands by typing one Indirect command line.

You create a file and put the DCL commands you want to execute into it in the order you want them processed. To execute this command file, type an at sign (@) and then the name of the file. Indirect and DCL then do all the work.

For example, the file SHOW.CMD in the user directory contains the following DCL command lines:

```
SHOW TIME
SHOW USERS
SHOW DEVICES
```

To execute this command file, type the following command line:

```
$ @SHOW [RET]
```

Indirect, which is invoked by the at sign (@), reads the commands in the file one at a time, waiting until each command has been executed before going on to the next one.

There are only two command files in the USER directory. For the other examples in this chapter, use EDT to create new command files. The examples in this chapter use all uppercase letters, but Indirect accepts both uppercase and lowercase. Since Indirect looks for .CMD file types by default, you should create your files with this type. If you use another file type, you must specify that file type whenever you want to execute the file.

Indirect Command Files

Indirect command files are used for many different things. One example is a login command file. When you log in, the system automatically runs the command file LOGIN.CMD, which can set various characteristics for your terminal or automatically run other programs or files. Take a look at the LOGIN.CMD file in the [USER] directory.

To illustrate, you can use LOGIN.CMD to change the characteristics of your terminal if the ones you want are different from the terminal's default characteristics. Put the necessary DCL command lines in LOGIN.CMD. The characteristics will be changed automatically when you log in. Here is an example of a login command file:

```
SET TERMINAL/SPEED:(9600,9600)/WIDTH:80
@COOKIE
SHOW USERS
SHOW TIME
```

This command file sets two different characteristics for the terminal: speed and width. (See the *Micro/RSX User's Guide* for more information about these commands.) The command file also runs another command file, COOKIE.CMD. When COOKIE.CMD finishes executing, Indirect returns to the first file (the login command file) to continue executing it. The SHOW commands display the users currently logged in on the system, and then the current time and date.

When you put commands into a login command file, you do not have to type those commands every time you log in; Indirect does all the work for you. Putting repetitive sequences of commands that you are going to use often into a file is what Indirect is especially good for. Using indirect command files saves you time and prevents mistakes. "Repetitive sequences of commands" can be just about anything. A few examples are listing files in your directory, mounting volumes, backing up files, or doing quick tests at your terminal.

The following sample command files will give you a better idea of what Indirect can do for you:

- To prepare a disk volume for use and then mount it, run a file containing the following command lines:

  ```
  MOUNT/FOREIGN DU1:
  ANALYZE/MEDIA DU1:
  DISMOUNT DU1:
  INITIALIZE DU1:MYDISK
  MOUNT/NOSHAREABLE DU1:MYDISK
  ```

  This file checks the volume (DU1:) for bad blocks (so that data will not be written to them), initializes the volume (which deletes any data currently on the volume and gives it the format of a Micro/RSX volume), and then mounts the volume. (Note that the device on which you mount a volume may not be DU1:, nor is the name of the volume likely to be "MYDISK." See the file named MYDISK.CMD in the USER directory. This command file can be used to prepare any diskette for use on the system.)

- To check for files in your directory, use a file containing command lines similar to the following:

```
DIR *.RNO;*
DIR *.MEM;*
DIR *.TXT;*
DIR *.LST;*
DIR *.CMD;*
DIR *.HLP;*
```

  These command lines display on your terminal lists of various files based on their file type. After looking at these lists, you can decide what you want to do with the files.

Substitution Mode

You may need to change indirect command files often to make them do exactly what you want to do each time. For example, you might use a command file to do a backup procedure but find that you have to edit the file to change the name of the device drive or its unit number. For such cases, Indirect has substitution mode.

Substitution mode allows you to place a special word—called a symbol—in the command line. When you run the command file, it asks you (through a special Indirect command line) for the information that is to be substituted for the symbol. An Indirect directive (or command), .ENABLE SUBSTITUTION, allows you to use substitution mode. Symbols are put in single quotes (' '). The single quotes tell Indirect to substitue a value for the symbol name before executing the command.

The following command file shows substitution mode being used:

```
.ENABLE SUBSTITUTION
.ASKS DEVICE Device to mount?
MOUNT 'DEVICE'
```

These command lines (which can be part of a larger command file) perform the following actions: they enable substitution mode, ask you which device is going to be mounted (DEVICE), assign your answer to the symbol DEVICE, and then mount that device. When you run the file, this is what you see on your terminal:

```
$ * Device to mount? [S]: DU1: RET
$ MOUNT DU1:
```

When you see "* Device to mount? [S]:" on your terminal, type the name of the device to be mounted and press the RETURN key. After you have answered the question, Indirect displays the MOUNT command line on your terminal, with the specific device name substituted for the symbol 'DEVICE', and the system mounts the device.

The asterisk (*) at the beginning of the line indicates that the question is being asked by Indirect. .ASKS means "ask for a string," so the "[S]:" at the end of the question indicates that Indirect expects a string answer, that is, an answer containing a string of alphabetic and/or numeric characters. Indirect also accepts other types of answers, depending on the question being asked.

When the command file is run, Indirect substitutes your answer to the question (DU1:) for the symbol 'DEVICE' in the MOUNT command line following the question. That is why you see "MOUNT DU1:" displayed on your terminal instead of "MOUNT 'DEVICE'." Using substitution mode lets you name any device with your command file.

The next command file, which displays information from a user's local (that is, private) help file, shows a similar use for substitution mode. (These commands could also be part of a larger file.)

```
.ENABLE SUBSTITUTION
.ASKS CMND Enter command name
HELP/FILE:DU2:[BONNIE]COMMANDS.HLP 'CMND'
```

The terminal session would be this:

```
$ @HELP [RET]
$ * Enter command name [S]: Telegram [RET]
$ HELP/FILE:DU2:[BONNIE]COMMANDS.HLP Telegram
   The TELEGRAM command sends a specified message . . .
```

As a way to display help files, this command file asks for the topic for which help is wanted. When the terminal displays "* Enter command name [S]:," you type in the topic you want help on. Indirect takes your answer, substitutes it for the symbol 'CMND', in the HELP command line, and then displays the requested help information immediately afterwards. (See Chapter 3 in the *Micro/RSX User's Guide* for more information on the HELP command.)

Writing Programs with Indirect

As you can see, Indirect can be used to write programs—in fact, the command file just shown is really a simple program. Many common programming techniques are available in Indirect. These techniques include looping, counters, variables, arithmetic and logical operations, and testing system conditions. The techniques are performed through the use of Indirect directives, symbols, and labels.

Directives

.ENABLE SUBSTITUTION and .ASKS are two of the many Indirect directives. This chapter does not describe all the directives, but acquaints you with a few that you are most likely to use and to use frequently. The .ASKS directive has two companion directives, .ASK (for true/false—or logical—questions) and .ASKN (for numeric questions). You can use .ENABLE and its companion directive, .DISABLE, to set and change several other modes in Indirect.

All Indirect directives begin with a period, except for the logical end-of-file directive, which is a slash (/).

For a complete list of the Indirect directives, see Chapter 9 in the *Micro/RSX User's Guide*.

Special Symbols

Indirect has special symbols that it defines automatically. The definitions of the symbols depend on specific system characteristics and the replies to queries given during command file execution. Special symbols can be compared, tested, or substituted and are of three types: logical, numeric, or string. All special symbols have a common format: angle brackets (< >) enclose the special symbol name.

For a list of the special symbols for Indirect, see Chapter 9 in the *Micro/RSX User's Guide*.

Labels

You can also use labels in command files. Labels allow you to organize your file more coherently and to jump to other lines in the file, depending on the results of conditional statements.

For example, the following command file asks for the values of two variables and then compares them.

```
        .ENABLE SUBSTITUTION
        .ASKN A Enter value for A
        .ASKN B Enter value for B
        .IF A > B .GOTO TEST2
        .EXIT
TEST2:  .SET A B
          .
          .
          .
```

Depending on the result of the comparison (done with the .IF directive), the command file either exits (.EXIT) or goes onto the section of the file labeled .TEST2:.

Notice that the label begins in the first column of the command file while the directives begin in the ninth column (one tab stop over). Formatting your command files in this way makes them consistent and easy to read.

Labels are one through six characters in length, begin with a period (.), and end with a colon (:). (The period and colon are not included in the six characters.) When you use labels in command lines within the command file, however, you only need to use the name; you do not need to include the period and colon. The .GOTO directive allows you to go to the different sections of the file marked by different labels.

The .IF and .SET directives, like .ASKS, have companion directives. The other .IF directives allow you to make tests for certain specific conditions. The other .SET directives allow you to set values as true, false, logical, numeric, string, octal, or decimal.

The following command file uses one of the other .SET directives, .SETS, and also the .ENABLE and .GOTO directives. The file also uses the special string symbol <TIME> . A more detailed explanation follows the text of the file.

```
        .; The following file prints a message on the terminal,
        .; depending on the time of day.
        .ENABLE SUBSTITUTION
        .SETS TIME "'<TIME>'"
        .; <TIME> has the format hh:mm:ss.
        .SETS SAYING TIME[8.:8.]
        .; Sets SAYING equal to last digit of <TIME> (1's column
        .; for seconds).
        .GOTO 'SAYING'00
        .; Makes a label based on the second <TIME> is checked.
.000:
        ;  What else can go wrong?
        .GOTO END
.100:
        ;  Have you seen your shrink today?
        .GOTO END
.200:
        ;  Ours is not to reason why.
        .GOTO END
.300:
        ;  Where were YOU when the lights went out?
        .GOTO END
.400:
        ;  Why are you here?
        .GOTO END
.500:
        ;  Everything is relative.
        .GOTO END
.600:
        ;  It will be a good experience for you!
        .GOTO END
.700:
        ;  Don't panic.
        .GOTO END
.800:
        ;  One lousy driver can ruin your whole day.
        .GOTO END
.900:
        ;  Curiosity killed the cat.
        .GOTO END
.END:
        .EXIT
```

Explanation of the Command File

In addition to the directives and special symbol, this command file also illustrates other features of Indirect. The first feature is the use of comments. Comments can be used to describe what the file is supposed to do and to explain what the command lines do or to give additional information about them. Comments that begin with a period and semicolon (.;) are not displayed on the terminal when the file is executed. Comments that begin with only a semicolon (;) or an exclamation point (!) are displayed.

This file, as the introductory comment explains, displays a message on the terminal when the file is run. The message displayed depends on the time at which the file is executed.

When the file begins to execute, substitution mode is enabled and then the symbol TIME is set with the .SETS directive to be equal to the contents of the special symbol <TIME>. <TIME> contains the current time in the format hh:mm:ss. The second .SETS command line sets the symbol SAYING to be equal to the last digit contained in <TIME>. The range [8.:8.] means that Indirect should look for the last character in the string of eight characters; in other words, the second digit for seconds. For example, if <TIME> contains 11:37:56, the symbol TIME is set to 6. That means that Indirect will display this message:

```
; It will be a good experience for you!
```

The .GOTO command line creates a label, using the second from <TIME>, so that Indirect will know which label to go to and which message to display. (In the above example, Indirect branched to label .600:.) The remainder of the file lists the labels and the messages to be displayed, and then branches to the .END: label after the message has been displayed. In that way, Indirect goes directly to the end of the file and exits without first displaying any messages following the one that was displayed.

The following examples will give you an idea of the usefulness and versatility of Indirect. A brief commentary follows each example. For more information on Indirect (directives, symbols, error messages, and so on), please see Chapter 9 of the *Micro/RSX User's Guide*.

Examples

Each example is followed by an explanation.

- The following command file creates a file named after the current date:

```
.ENABLE SUBSTITUTION
.SETS DATE <DATE>
.SETS DAY DATE[1:2]
.SETS MONTH DATE[4:6]
.SETS YEAR DATE[8.:9.]
.ASKS TYPE What file type?
.SETS NAME DAY+MONTH+YEAR+"."+TYPE
EDIT 'NAME'
```

  In this file, substitution mode is enabled, then the symbol DATE is set to the contents of the Indirect special symbol <DATE> (for example, 15-JUL-85). The symbol DAY is then set to the two characters for the date (15), which are the first and second characters contained in <DATE> . The symbol MONTH is then set to the three characters for the month (JUL), which are the fourth through sixth characters contained in <DATE> . The symbol YEAR is set to the two characters for the year (85), which are the eighth and ninth characters contained in <DATE> . (By default, Indirect considers numbers to be octal. Unless .ENABLE DECIMAL is in effect, you must use a decimal point (.) after a number for Indirect to accept it as decimal. Notice the decimal points after 8 and 9 in the example. See Chapter 9 in the *Micro/RSX User's Guide* for more information on .ENABLE DECIMAL.)

  The .ASKS command line asks you for the file type of the file being created, and the symbol NAME becomes the concatenation of the previous three symbols and TYPE. NAME, therefore, becomes the name of the file being created, for example, 15JUL85.TXT. The last command line in the file invokes EDT to edit the new file.

- The following command file concatenates several DCL help files into one file named HELPFILES.TXT and then prints the file after a certain specified time:

```
.ENABLE SUBSTITUTION
.ASKS DEVICE Enter device and directory spec
.ASKS TIME Time to print (hh:mm)?
.IF TIME EQ " .SETS TIME "0"
COPY 'DEVICE'DCL.HLP HELPFILES.TXT
APPEND 'DEVICE'ALLOCATE.HLP,BROADCAST,COPY HELPFILES.TXT
APPEND 'DEVICE'DIRECTORY.HLP,DISMOUNT,HELP HELPFILES.TXT
APPEND 'DEVICE'INITIALIZE.HLP,LINK,MOUNT HELPFILES.TXT
APPEND 'DEVICE'PURGE.HLP,RENAME,RUN,SET HELPFILES.TXT
PRINT/AFTER:('TIME') HELPFILES.TXT
```

In this file, substitution mode is enabled. Indirect asks which device and directory the files are to be copied from ('DEVICE') and the time after which the files are to be printed ('TIME'). If no specific time is given, the files are queued to be printed immediately.

The COPY command creates the new file HELPFILES.TXT and copies into it the help file DCL.HLP. The APPEND commands add more help files to the end of HELPFILES.TXT. The PRINT command prints HELPFILES.TXT after the time given in response to the "*Time to print?" question.

- The following command file can help you delete unnecessary files from your directory:

```
        .ENABLE SUBSTITUTION
.BEGIN:
        .ASKS FILE Which file?
        TYPE 'FILE'
        .ASK DEL Delete this file
        .IFT DEL DELETE 'FILE';*
        .GOTO BEGIN
```

With this file, substitution mode is enabled and Indirect asks for the name of a file to be deleted. However, before the file is deleted, DCL displays the file on the terminal and then Indirect asks if the file should be deleted. This ensures that you do not delete a file that you really want to keep.

If you answer "Yes" (Y) to the question, DCL deletes the file. After the file is deleted, Indirect loops back up to the beginning and asks again for the name of a file to be deleted. If you have no more files to be deleted, type CTRL/Z in response to the "* Which file?" question.

- The following command file gets information about the system, your account, and your terminal, and writes the information into another file:

```
.ENABLE SUBSTITUTION
.OPEN INFO.DAT
.ENABLE DATA
'<DATE>'            !This is today''s date.
'<TIME>'            !This is the current time.
'<UIC>'             !This is your current UIC.
'<LOGDEV>'          !This is your login device.
'<NETNOD>'          !This is the DECnet node name for your
                    !system.
'<TISPED>'          !This is the baud-rate code for your
                    !terminal.
'<TITYPE>'          !This is the type code for the
                    !terminal you are using.
.DISABLE DATA
.CLOSE INFO.DAT
TYPE INFO.DAT
```

With this file, substitution mode is enabled, a new file called INFO.DAT is opened so that the information can be written into it (if the file already exists, Indirect creates a new version), and then data mode is enabled. Data mode allows several lines of text to be written into a file.

Then, Indirect gets the contents of the various special symbols and writes the information into INFO.DAT. After the last symbol is read, data mode is disabled, and INFO.DAT is closed and then displayed on the terminal. For example:

```
$ @INFORM [RET]
$ TYPE INFO.DAT
13-JUL-85               !This is today's date.
10:14:37                !This is the current time.
[303,23]                !This is your current UIC.
DUO                     !This is your login device.
AMITY                   !This is the DECnet node name of your
                        !system.
22                      !This the baud-rate code for your
                        !terminal.
15                      !This is the type code for the
                        !terminal you are using.
$ @ <EOF>
$
```

As you can see, the appropriate information has been written into the new file.

Notice in the command file that there are two apostrophes in "today"s", but only one apostrophe shows in the display. When substitution mode is enabled, you must use two apostrophes in any comments so that the text shows up correctly. When you use only one apostrophe, Indirect assumes the text following the apostrophe to be a string symbol. See Chapter 9 in the *Micro/RSX User's Guide* for more information.

Also, a code of 22(octal) for <TISPED> means that the baud rate of the terminal is 9600. A code of 15(octal) for <TITYPE> means that the terminal is a VT100. See Chapter 9 in the *Micro/RSX User's Guide* for complete lists of the codes for the baud rates and terminal types.

You can also use Indirect directly from the terminal without running a command file. The following command line lets you work with Indirect interactively:

```
$ @TI:  RET
AT.>
```

When Indirect responds with AT.> (the task-name prompt), you can enter Indirect command lines, invoke command files, or display the values of special symbols. To display a symbol, use the .ENABLE SUBSTITUTION directive, and then request the symbol in the following format:

```
AT.>;'<symbol>'  RET
```

For example, if you do

```
AT.>.enable substitution  RET
AT.>;'<time>'  RET
```

Indirect responds with

```
$ ;15:57:56
AT.>
```

The semicolon before the symbol indicates that Indirect should display the time on the terminal, but DCL should not try to execute it as one of its commands.

To exit from Indirect, type CTRL/Z:

```
AT.> CTRL/Z
$ @ <EOF>
$
```

Batch Processing

Batch processing is an alternative method of passing commands to the operating system automatically. The text that follows illustrates how batch processing works on Micro/RSX.

Batch jobs differ from indirect command files in that a batch job is a complete terminal session, whereas an indirect command file is only part of a terminal session. You must be logged in on a terminal to run an indirect command file, but you can run a batch job long after you have logged out and gone home. A batch job runs on a special kind of terminal called a *virtual terminal,* which is really software.

Another difference between batch processing and Indirect processing is that batch jobs can produce a log of the job as it runs.

A final difference is that batch processing does not have the complete programming capability of the Indirect directives. You can, however, invoke indirect command files from within a batch job.

An Example of a Batch Job

The following file, called BATCH.BAT, is an example of a batch file:

```
$JOB HIYA [200/1]
$COPY OLDFILE.TXT HIYA.TXT
$APPEND JOHN.TXT,COVERT.DAT HIYA.TXT
$PRINT HIYA.TXT
$@INFORM
$PRINT SYSTEM.DAT
$EOJ
```

This is a complete user batch job. In batch jobs, a dollar sign ($) precedes batch-specific and DCL commands. The dollar sign notifies the batch processor that a command follows.

The JOB command logs the batch job onto the virtual terminal. This command also gives the name HIYA to the user batch job and, by including the slash in the UIC, keeps all but important login messages out of the batch log.

The COPY, APPEND, and PRINT commands work as usual, as does the indirect command file INFORM.CMD

Comments can be included in the batch job, and thus in the batch log, by using an exclamation point (!) after the dollar sign and before the comment.

A line without a dollar sign in the first position notifies the batch processor that the line contains data. The DATA and EOD commands can be used to include data in batch jobs but are not necessary.

Submitting Batch Jobs

You pass the batch job to the batch processor with the DCL command SUBMIT. For example:

```
$ SUBMIT/AFTER:(17:30) BATCH.BAT [RET]
```

The SUBMIT command places the batch job in the batch queue, from which it will run after 17:30 (5:30 P.M.) on the day it was submitted. When the job runs, it produces a log similar to the following one, which is printed on the line printer when the job completes:

```
QMG Batch Job - BATCH          BPR  V04.00     31-Aug-85 10:37   Page 1
Processor BAPO
10:37:05        $JOB HIYA [200/1]
         ====================================
         User Job - HIYA         Terminal VT2:
                   UIC = [200,1]
         ====================================
         TERM
           .     Micro/RSX  V3.0 BL24   [4,54] System     AMITY
           .     $@LB:[1,2]SYSLOGIN.CMD
10:37:06         $COPY OLDFILE.TXT HIYA.TXT
10:37:08         $APPEND JOHN.TXT,COVERT.DAT HIYA.TXT
10:37:13         $PRINT HIYA.TXT
         TERM    PRI - Job 43, name "BATCH    ", submitted to queue "PRINT "
10:37:14         $@INFORM
         TERM    $@ <EOF>
10:37:16         $PRINT SYSTEM.DAT
         TERM    PRI - Job 43, name "BATCH    ", submitted to queue "PRINT "
10:37:17         $EOJ
         TERM    Connect time:  0 hrs  0 mins  25 secs
           .     CPU time used: 0 hrs  0 mins   5 secs
           .     Task total:    20
                     .
                     .
                     .
```

The batch log includes a record of all commands and data that the batch job passed to the virtual terminal as well as any output sent to the virtual terminal. You should try to relate the lines in the batch job to the lines in the log. The file HIYA.TXT is printed on the second page of the batch log, and the file SYSTEM.DAT is printed on the third page.

For more information on batch processing, see Chapter 8 in the *Micro/RSX User's Guide*.

Glossary

abort

Stopping a program from running before it is finished is called aborting the program. On most Micro/RSX systems, you can do this with the CTRL/C command, but there is also a separate ABORT command in DCL that does the same thing. When nonprivileged users log out, LOGOUT aborts any programs they have runnning at the time.

Aborting a program does the program no harm, nor does it harm the system. If the program keeps records of any kind, as a business system would, then aborting it may result in incomplete records, but these can usually be brought up to date. Aborting programs of this sort may also result in locked files.

account

Each system user has an account. This is a record of the user's User Identification Code (UIC), name, password, default disk, default directory, and privilege status. System managers create accounts using the Account File Maintenance Program, ACNT.

applications task

An applications task is any task that uses the operating system to run, but is not part of that system. Examples include games, office automation programs, graphics programs, control programs, and so forth.

argument

Arguments add specific information to DCL command qualifiers.

A DCL command consists of a command and optional qualifiers. The qualifier alters the operation of the command. For instance, the PRINT command has a /COPIES qualifier. This qualifier accepts an argument specifying the number of copies you want. In DCL, arguments are preceded by a colon (:) or an equal sign (=). The following example includes two qualifiers with arguments.

```
$ PRINT/COPIES:3/NAME:MELISSA IVAN.LAB [RET]
```

This command means print 3 copies of the file IVAN.LAB and give the name MELISSA to the print job.

See the *Micro/RSX User's Guide* for more information on DCL commands.

ASCII

ASCII stands for American Standard Code for Information Interchange. ASCII is the standard format for readable text on computers. It is a code used to translate letters, numbers, and symbols from a keyboard into machine code, and vice versa.

Thus, an ASCII file is a file that can be read both by people and by computers.

bad block

Bad blocks are blocks on a mass storage device that are not usable because there is some physical damage or flaw. The ANALYZE/MEDIA command is used to find bad blocks. You can use disks or diskettes that have bad blocks on them, but if there are many bad blocks, you should consider replacing the disk or diskette. Different devices have different levels of tolerance for bad blocks, but with diskettes you will usually find that there are either no bad blocks or many. You should discard diskettes with more than two bad blocks. You should back up any diskette with any bad blocks, no matter how few.

batch log

A batch log is a file or printed listing documenting everything that happened to a particular batch job.

batch processing

Batch processing is a mode in which all commands to be executed by the operating system, or data to be used as input to the commands, are placed in a file and submitted to the system for execution.

Batch jobs can be scheduled to run at a particular time, such as at night when no one is using the system.

See also Indirect Command Processing.

block

A block is a unit of measurement for files. In almost all cases, a block is 512 bytes. Since each character in text takes one byte, this means that one block in an English language text file contains about 80 words of text.

The term "block" is also used to refer to various parts of the system that contain processing information, such as a Task Control Block (TCB) used by Micro/RSX to control tasks.

boot

See bootstrap.

bootstrap

In computer terminology, a bootstrap is an operation that brings itself into a desired state by its own action, as in the expression "She lifted herself by her bootstraps." In Micro/RSX systems, the bootstrap is a routine included in the MicroPDP-11 computer that includes enough instructions to bring the rest of the operating system into the computer's main memory. A bootstrap is often called a boot, and bootstrapping is often called booting.

Booting means bringing a fresh copy of the operating system into action. This is accomplished by turning the power off and then on again on the MicroPDP-11, or by pressing the RESTART button. When you do this, the computer first checks its hardware and then brings the operating system back from the fixed system disk into memory. See the *Micro/RSX System Manager's Guide* for more information.

buffer

Buffer refers to a temporary storage area in a program. In this book, the term refers to the buffers created by EDT for your use in creating and modifying files. EDT always starts out in a buffer named MAIN, but it has other buffers available, and you can create your own buffers. See the *Micro/RSX User's Guide* for more information on EDT and its buffers.

central processing unit

See CPU.

character mode

EDT's character mode uses the video screen to operate on text one character at a time, in contrast to line mode, which operates on one line at a time. Character mode editing commands are entered using the keypad.

See line mode.

circumflex

The circumflex character, also called an up-arrow, looks like a hat or a roof:

It is used on Micro/RSX systems to indicate that you have typed a control character.

command

A command, when executed, is an instruction to the software to perform a particular action. For instance, the following command directs Micro/RSX to perform a series of operations:

```
$ TYPE IZZY.TXT [RET]
```

This command directs the system to find a file named IZZY.TXT and display the contents of that file on your terminal.

In this book, you learn commands to DCL, the DIGITAL Command Language, and commands to the EDT editor. See the *Micro/RSX User's Guide* for more information on DCL commands.

compiler

Each high-level language is implemented through a compiler. A compiler is a program that takes a source program written in the high-level language and translates it into binary object modules that can then be translated into tasks by the Task Builder. See *Programming on Micro/RSX* for more information.

contiguous

A contiguous file consists of physically adjacent portions on a mass storage device. Contiguous files can be loaded into main memory in a single operation. The most common contiguous files are task image files, but other files can also be contiguous. Contiguous files are indicated by a letter C in the directory listing, as shown below:

```
QIX.TSK;2        40.        C        01-APR-85 00:01
```

control character

A control character is a special form of command to the system entered by pressing the CTRL key and a letter key together. The most important control character is CTRL/C, which aborts any task running on your terminal. Other useful control characters include CTRL/Z, which means "end-of-input," and CTRL/O, which skips over unwanted output on your terminal. Control characters are sometimes indicated by a circumflex (^) followed by the character, as shown below:

```
^Z
```

CPU

CPU stands for Central Processing Unit. It is the hardware that handles all the calculation and routing of input and output (I/O), as well as the execution of tasks. The CPU is the part of the computer that actually computes.

crash

A crash is the system's response to an unstable condition. Rather than continuing to operate and allowing the system to do itself damage, it ceases operation. In general, all you'll need to do is boot the system again; however, persistent crashes are a sign of trouble.

cursor

The cursor is a flashing indicator used on video terminals to point to the screen position where the next character will appear. It is called a cursor because it shows the "course" the printed or typed line will follow. The VT100- and VT200-series terminals allow you to choose a solid block (▒) or an underscore line (_) as a cursor. See your terminal manual for information.

DCL

DCL stands for DIGITAL Command Language. DCL provides a means of communication between the user and Micro/RSX. DCL is designed to be easy to use. Commands are generally English words. If necessary elements are not typed in, DCL prompts for them. DCL also provides help for the user.

DCL is used on most DIGITAL operating systems. There are differences from system to system, but for everyday use, DCL is quite similar on all systems.

default

A default is a value or operation that is automatically included in a command unless you specify otherwise.

In most cases, default settings will be what is normal or expected. Many times, you will not even notice that defaults are being used, but the default settings can always be overridden. You can always find the defaults for any command in the *Micro/RSX User's Guide* and in the help files.

In Micro/RSX and the RSX family in general, a wide range of defaults is used. The idea is that the less the user has to specify in any given situation, the easier the system is to use and the smaller the chance of error.

delimiter

A delimiter is a character that separates, terminates, or organizes the elements of a command or file specification, such as the semicolon (;) before the version number, or the slash (/) that sets off a qualifier from a DCL command.

The RETURN key is a delimiter that marks the end of a command field or command. Other delimiters are punctuation marks, such as the colon (:) and comma (,). Spaces or tabs are also common delimiters. These small elements play an important part in keeping matters organized on the system.

device

A device is any peripheral hardware connected to the processor and capable of receiving, storing, or transmitting data. Devices commonly provided on Micro/RSX systems include terminals, line printers, a fixed disk, diskette drives, and sometimes tape drives.

All devices have names in the same form: two letters, a number, and a colon (:). Terminals are called TT1:, TT2:, and so forth. The line printer is usually LP0:. The first device of any type is always number 0.

DIGITAL Command Language

See DCL.

directory

A directory is a file that briefly catalogs a set of files stored on disk or tape. The directory includes the name, type, and version number of each file in the set. Every user has a default directory.

Directories can have names of up to nine characters, such as [SCHMENDRK] or [JESSEJOE], or they can have names consisting of two numbers, such as [303,26] or [7,11]. Directories with two numbers can generally be referenced either as [7,11] or [007011]. There is no distinction between the two kinds of directories.

The DCL command DIRECTORY displays information about files in directories.

disk

The disk is the major type of mass storage device on Micro/RSX systems. Disks are high-speed, random-access devices. There are several kinds. Most Micro/RSX systems include a fixed disk and two diskette drives with removable diskettes.

disk-based system

On a disk-based system, such as Micro/RSX, the tasks and other functions that make up the operating system are stored on a disk and loaded into memory as they are needed by users, then removed when they are no longer needed. The Micro/RSX system is kept on the fixed disk because of the disk's speed and capacity. The disk with the system on it is called the system disk.

echo

When characters that are typed on a terminal keyboard are also displayed on the terminal, the process is called echoing. Terminals are dual devices, sending input and receiving output. Echoing is one form of receiving output from the system.

editor

An editor is a system task for creating and altering text files. Micro/RSX systems include EDT, the standard DIGITAL editor.

error message

Error messages are sent by the system when some action you have requested fails. Each error message identifies the command or system function that detected the error. For instance, error messages from the TYPE command are labeled TYP.

The great majority of error messages result from mistakes in typing or mistakes in syntax. Often, you can correct the error by retyping the command.

Most system error messages are explained in the *Micro/RSX User's Guide*. See the individual command descriptions and Chapter 16 of the *Micro/RSX User's Guide*.

Executive

The Executive controls the operating system. The Executive coordinates all activities in the system, including task execution, user communication, supervision of input and output (I/O), and resource allocation. The name RSX stands for Resource Sharing Executive.

FCS

FCS stands for File Control Services, a set of routines that can be used in tasks to open and close files, read from them, write to them, extend, or delete them. FCS provides a set of macros to simplify the user's interface to the system I/O structures.

High-level language statements that operate on files on Micro/RSX systems are implemented through these routines, or a similar set of RMS–11 routines. See *Programming in Micro/RSX* and the Advanced Programmer's Kit for more information.

field

The term field usually refers to a portion of a command or command element. For example, the file name and file type are two fields of the file specification.

file

A file is a set of data arranged in a structure significant to the user; it is one of the basic units of information on Micro/RSX.

A file is any named, stored program or data, or both, to which the system has access. Access can be of two types: (1) read-only, meaning the file cannot be altered, and (2) read-write, meaning the contents of the file can be altered. See read and write.

See also volume.

File Control Services

See FCS.

file specification

The file specification, sometimes called a filespec, is the unique identification of a file that gives its physical location and, generally, an indication of its contents.

All file specifications are in the following form:

```
DU0:[USER]FLY.TXT;1
```

The device name is two letters and a number followed by a colon (:), such as DU0:.

Next is the directory, enclosed in square brackets, such as [USER].

File names can include 1 to 9 of the letters A through Z and the numbers 0 through 9, but no other characters. The name, such as FLY, should give some indication of the contents.

The file type starts with a period (.) and includes from 0 to 3 characters. It usually gives some indication of the type of file, such as .TXT.

The version number is set off by a semicolon (;).

See the *Micro/RSX User's Guide* for more information about file specifications and their component parts.

fixed media

Some mass storage devices, notably the main disk on the MicroPDP-11, are fixed in place and cannot be removed. In general, the fixed medium is used for storage that must be on line and accessible at all times, such as the operating system itself. Compare with removable media.

floppy diskette

Floppy diskette is a common term for a flexible diskette.

form feed

A form feed is a nonprinting character that causes a line printer or hardcopy terminal to move the paper up to the next full page. You can include a form feed in text by inserting a CTRL/L while editing. You will see <FF> in your text. When the file is printed, the line printer moves to a new page.

functionality

Functionality is a computer industry term that means nothing more than what the hardware or software can do. A synonym for functionality is feature.

global

Global means affecting the entire file, or the entire system, or the entire task, depending on the context. In this book, you have learned about global substitutions, that is, changing all instances of one string in a file.

hang

When a terminal or task appears to be going nowhere or doing nothing, it is said to be hanging. Hung terminals are sometimes described as static, or dormant, or locked.

Sometimes, you can correct a hung terminal by pressing CTRL/Z or CTRL/C, or by turning the terminal off and on. You should also check to be sure the NO SCROLL or HOLD SCREEN key hasn't been pressed.

hardcopy terminal

Terminals that print output on paper are called hardcopy terminals; they are also called printing terminals. Hardcopy terminals preserve a permanent record of everything that is printed or typed on them.

Micro/RSX systems often use a hardcopy terminal as a line printer.

hardware

Hardware is all the parts of the computer system you can touch. The terminals, the computer, the disk drives, the line printer, are all hardware. Your system may have special hardware.

See software.

Hardware and software must be in harmony for the system to work at full efficiency.

help file

A help file is a text file in a form suitable for use with the HELP command. Many help files are included as part of the Micro/RSX system, but you can also write your own help files. See the *Micro/RSX User's Guide*, Volume 2, Chapter 12, for more information.

high-level language

High-level languages, for example, BASIC-PLUS-2, FORTRAN-77, and COBOL-81, are transportable programming languages. Programs in these languages are not tied to a particular kind of computer. They are called high-level because programs written in these languages usually provide a higher level of information about what the program will do than assembly language provides.

Each programming statement in a high-level language is translated into several machine-language instructions.

Indirect Command Processor

The Indirect Command Processor passes commands to the operating system automatically. In addition, the Indirect Command Processor permits you to use programming techniques, such as loops, counters, labels, and symbol substitution, to set up more elaborate procedures. Any series of commands you have to enter over and over with few or no changes is a candidate for Indirect processing.

See batch processing.

input

Input is a computer term meaning whatever you supply to the system. Most input is typed in, but both batch processing and Indirect processing supply input to the system without typing once they are started.

input file

Many system utilities and commands take existing files and produce new files. For example, the COPY command takes a file from one place and copies it to another. EDT can edit a file and make a new one from it. In these cases, the file being copied is called the input file and the file being created is called the output file.

install

When you copy the operating system or an application from its distribution media to the system disk, you are installing it.

Installing a task has a different meaning. An installed task is named in the System Task Directory (STD), a list of Task Control Blocks (TCBs) that contain information about each task. Taking a task out of the STD is called removing it.

A task cannot run unless it is installed.

Users automatically install and remove their tasks through the RUN command. Privileged users can also install and remove tasks explicitly.

installation

The installation is the full computer system at your location. The installation includes the operating system, the programming languages, and applications tasks, as well as the computer and its hardware devices.

Each installation has a different collection of hardware and software which has been selected and customized for the needs of that particular installation. For this reason, not every capability or function mentioned in the system documentation is available at every installation.

interactive system

Micro/RSX is an interactive system. This means that you and the operating system communicate directly using the terminal. The Micro/RSX operating system immediately acknowledges and acts upon commands you enter at a terminal.

journaling

Journaling is an EDT feature that allows you to recover work that is lost from a system interruption.

While you are using EDT, it records each keystroke you make. If the system crashes while you are editing, this record is preserved. You can then restore your file to where it was before the crash using the /RECOVER qualifier to the EDIT command. See the *Micro/RSX User's Guide* for more information.

language library

Most high-level languages have a unique set of routines for program support that are collected in a separate library. Some of those routines are similar to routines commonly found in the system library. Some routines found in a language library are required by the compiler to properly implement some high-level instructions, such as WRITE or PRINT. Others are required by the Task Builder to correctly link the program.

library

A file containing one or more relocatable routines that can be incorporated into a task is called a library. A system library is supplied, but you may also create user libraries for your installation or application. See *Programming on Micro/RSX* and the Advanced Programmer's Kit for more information.

See also language library, object library, resident library, system library, and user library.

line mode

EDT has two main modes of operation: line mode and character mode. Line mode operates on a line or group of lines and is well suited to manipulating large blocks of text. Line mode editing commands are English words.

See also character mode.

line number

EDT automatically assigns numbers to the lines of the file you are editing. The numbers are useful in finding and manipulating the contents of the file. In line mode, you can see the numbers on your terminal; in character mode, you do not. In any case, the numbers disappear when you leave the editor.

line pointer

A line pointer marks where you are in a file. For example, EDT uses an invisible line pointer to mark your place in the file you are editing.

line printer

The line printer is an output device that prints files one line at a time. It is used to print large amounts of output in a hardcopy form. In some cases, the line printer will actually be a high-speed hardcopy terminal. Not all Micro/RSX systems have line printers.

In general, the line printer is under the control of a system task called the Queue Manager. You send files to the line printer with the PRINT command.

link

See Task Builder.

listing

This is a common computer term for output printed on the line printer. It also refers to the text file of a program as produced by a compiler.

load

When a task is loaded, it is located in main memory and therefore available for use.

Most tasks on Micro/RSX stay on the system disk until needed, at which time the system loads them into memory.

locked files

Occasionally, when a program terminates abnormally (for example, when you issue an ABORT command), files that the program was using are locked. You may not discover the locked files until you try to run the program again and find that you cannot.

Locked files are indicated by the presence of a letter L in the directory listing, such as the following:

```
LCPJUL85.MAI;1        267.        L        01-AUG-85 09:27
```

You can use the DCL command UNLOCK on locked files. You should be aware, however, that Micro/RSX locks files for your protection. You should check to make sure the data in locked files is sound after you unlock them. How you check such data naturally depends on what the files are for, but if a text file has been locked, you can read it over after it has been unlocked.

If you continually have trouble with locked files, there may be some problem with the program that uses the files.

login

Logging in identifies you to the operating system and informs the system that you have certain privileges and are using a particular terminal. You can log in on Micro/RSX with either the HELLO or LOGIN command. They are identical. You'll also need an account to log in to, and a password.

logout

Logging out informs the operating system that you have finished using a particular terminal. You can log out with the LOGOUT or BYE command. Logging out aborts any task you have running from your terminal and eliminates your access to a disk or tape.

macro

A macro, in MACRO–11 Assembly Language, is a single assembly language instruction that generates a predefined set of machine language instructions.

MACRO–11 derived its name from its capacity to define macros. A MACRO–11 user can, in effect, create high-level instructions by writing macros. Many macros are available in the system macro library.

MACRO–11 assembly language

Most of the system tasks and utilities on Micro/RSX are written in MACRO–11 assembly language. The language is called MACRO–11 because it allows programmers to define macros. A macro is a series of instructions that collectively perform some operation, and that can be called by a single name.

MACRO–11 includes a number of functions designed to make programming easier. These functions include directives to divide programs into sections, conditional assembly directives, a comprehensive system macro library, and user-defined macro libraries.

MACRO–11 is available separately in the Advanced Programmer's Kit. See *Programming on Micro/RSX* and the Advanced Programmer's Kit documentation for more information.

main memory

Main memory is a series of storage locations from which the CPU fetches its data. The contents of main memory can be easily altered. It can also be randomly accessed. When a task is run, it is loaded in main memory. The task has no access to the CPU if it is not in main memory.

Compare with mass storage device.

mass storage device

A mass storage device is a device, such as a disk, where data files and other types of files are stored when they are not being used. The Micro/RSX system and its components reside on a mass storage device most of the time.

media

See medium.

medium

The medium is the physical device, such as a disk or magnetic tape, that contains the data. The plural of medium is media.

See also volume.

memory management

Memory management is a process that supports the running of large programs on PDP-11 computers. All current DIGITAL computers include memory management hardware. Micro/RSX includes software that works with this hardware.

The instruction set of the PDP-11 computer forms 16-bit virtual memory addresses, so a program can directly address only 64K bytes of memory. The actual physical address space on PDP-11s is 4096K bytes, or 4 megabytes. Memory management is a combination of PDP-11 hardware and RSX-11 software that permits programs to translate 16-bit virtual addresses into 22-bit physical addresses. With 22-bit addresses, programs can address all of memory.

See *Programming on Micro/RSX* and the Advanced Programmer's Kit for more information. From the software side, memory management is handled by the Task Builder.

mode

Mode refers to a possible condition or state of operation. For example, EDT can operate in line mode or character mode.

mount

Mounting a volume makes the volume recognizable to the system and gives the system access to the files on that volume. For example, before you can use the files on a diskette, you use the DCL command MOUNT to tell the system which drive the diskette is in and to make the files on the diskette available for the system to use.

monitor

DCL checks the activity on your terminal when nothing else is happening. Therefore, DCL is sometimes referred to as monitor level. When you are in EDT, you are not at monitor level.

multiuser

A multiuser system, such as Micro/RSX, permits a number of users to work on their terminals simultaneously with little or no interference between users. Users on a multiuser system have their own files and their own share of time on the system. Commands such as LOGIN and LOGOUT are part of the protection offered on a multiuser system, as is the ability to make one of the diskettes your private device through the command MOUNT/NOSHAREABLE.

nonprivileged

Most Micro/RSX users are nonprivileged. Nonprivileged users run programs, or tasks, on the system, but they have no means of directly affecting the system and its operations. In most cases, users do not need to be privileged. See also privileged.

object library

An object library is a file containing a collection of compiled or assembled routines that can be included in a user program's task image. Object libraries commonly reside on disk devices and are only present in memory when routines within a library are called by a program being compiled or assembled.

object module

An object module is a program, or part of a program, that has been converted from the programming language in which it was written to a format the computer can use. This conversion is performed by a language processor, called an assembler or compiler. Object modules are files with the file type OBJ. An object module must be processed by the Task Builder to make a task file, which is the executable program.

The Micro/RSX Base Kit includes the Task Builder, but does not include any language processors. The MACRO-11 Assembler is available separately as part of the Advanced Programmer's Kit. Many other programming languages are also available separately.

See Chapter 14 of the *Micro/RSX User's Guide* for more information. *Programming on Micro/RSX* also has information on object modules. For full information, see the *RSX-11M/M-PLUS and Micro/RSX Task Builder Manual*, which is available separately or as part of the Advanced Programmer's Kit, or see other documentation provided with your language processor.

ODT

ODT (the On-line Debugging Tool) provides special code that you link into your task image to help debug a program. ODT commands and operators allow you to execute your program gradually by setting breakpoints at selected locations or by stepping through the program one instruction at a time. See *Programming on Micro/RSX* for more information.

off line

Equipment and devices that are unavailable for use are considered to be off line. For example, turning off a line printer puts the printer off line.

on line

On line is a computer term meaning ready for use. Peripheral devices can be on line or off line.

On-line Debugging Tool

See ODT.

operating system

An operating system is a set of computer programs that work together to manage computer resources for efficient operation. An operating system is used for communicating with the computer, for developing programs, and for scheduling the efficient use of the computer hardware, including memory, CPU, terminals, line printers, and communications devices.

The Micro/RSX operating system is part of the RSX–11 family of DIGITAL operating systems.

output

Output is a computer term meaning whatever the system or a program returns to you. For example, all the prompts from DCL are system output. If you use the TYPE command to display a file, your command is input, and the contents of the file displayed at your terminal is output.

output file

Many system utilities and commands take existing files and produce new files. For example, the COPY command takes a file and creates a copy of it. EDT can edit a file and make a new one from the original. In these cases, the file being copied is called the input file, and the file being created is called the output file.

password

A password is a protective mechanism to identify a particular user. Only you should know your password. Anyone who knows your password can log in to your account and do what they like.

peripheral device

Any auxiliary device that can provide the system with input, or accept output from the system, is called a peripheral device or a peripheral. Terminals, line printers, and disks are all peripheral devices.

privilege

Privilege determines the level of system access allowed to a user or a task.

privileged

On Micro/RSX systems, most users are nonprivileged. This simply means that they are not allowed to perform operations or issue commands that will affect the system as a whole. Nonprivileged users can find out what time it is with the SHOW TIME command, but only privileged users can change the time with the SET TIME command.

You become privileged by logging in to a privileged account. The same applies to becoming nonprivileged. The system manager is usually privileged.

For more information on privilege, see the *Micro/RSX User's Guide*.

prompt

A prompt is a sign that the system is ready to accept input from you.

For example, when you enter the TYPE command without specifying a file name, the TYPE command prompts you as follows:

```
File(s)?
```

By default, DCL prompts with a dollar sign ($).

In line mode, EDT prompts with an asterisk (*).

protection

One of the major features of a multiuser system is its ability to tell one user's files from another's. On Micro/RSX systems, each file has a protection code that specifies what kind of access different users can have to the file and what they may do to the file when they access it. File protection is based on the UIC. You can display your UIC with the SHOW UIC command. The UIC is a two-number code, such as [303,5].

There are four kinds of users:

1. SYSTEM—The operating system itself and privileged users, those with group numbers of 10 or less.
2. OWNER—The user with the same UIC as the file owner. You can display the file owner, and the file's protection, with the /FULL qualifier to the DIRECTORY command.
3. GROUP—Users with the same group number, which is the first number of the pair in the UIC.
4. WORLD—Everybody else.

There are also four kinds of access to files:

1. READ ACCESS—A user can read, copy, print, or type the file. If the file is a task image file, READ access means you can run the program.
2. WRITE ACCESS—A user can add new data to the file by writing to it.
3. EXTEND ACCESS—A technical provision so that tasks can change the amount of disk space allocated to the file.
4. DELETE ACCESS—A user can delete the file.

You can display the protection and ownership of any file with the DIRECTORY/FULL command. You can change the protection of files you own with the SET PROTECTION command.

See Chapter 5 of the *Micro/RSX User's Guide* for more information on protection.

pseudo device

A pseudo device is an entity treated as an input/output device by the user or system, although it is not any particular physical device. The pseudo device name is a stand-in name through which the actual physical device is reached. (This is similar to addressing a letter to the Governor of North Carolina without knowing the name of the person holding the office.)

The pseudo device convention makes it possible to refer to a device on any RSX-11 system without knowing its physical name and number. Thus, pseudo device LB: is always the disk containing the operating system itself and TI: is always the terminal you are using, no matter what its number is or whether it is local or remote, hardcopy or video.

See the *Micro/RSX User's Guide* for more information on pseudo devices.

qualifier

A qualifier for a DCL command is always preceded by the slash character (/). The qualifier alters the action of a command. Most often, qualifiers override defaults. For instance, this command uses the default, which is to print one copy:

```
$ PRINT IZZY.TXT [RET]
```

Adding the /COPIES qualifier overrides the default and prints the number of copies you specify, such as the following:

```
$ PRINT/COPIES:2 IZZY.TXT [RET]
```

In this case, /COPIES is a command qualifier, altering the operation of the command itself. The number 2 is an argument to the /COPIES qualifier.

DCL also uses file qualifiers, which alter the effect of a command for one file associated with the command, but not others. For example, the command

```
$ PRINT IZZY.TXT, OZY.TXT/COPIES:2, FIZZY.TXT [RET]
```

prints one copy of IZZY.TXT and FIZZY.TXT because that is the default, but two copies of OZY.TXT. See the *Micro/RSX User's Guide* for more information on DCL qualifiers.

queue

A queue is a waiting line. In the computer industry, a queue is a list of items to be processed according to system or user priorities. On Micro/RSX systems, queues are under the control of the system task called the Queue Manager.

Queue Manager

The Queue Manager, also called QMG, is a system task that controls queues of jobs directed to batch processors, line printers, or other output devices. In general, the Queue Manager keeps the jobs separate and in order.

random access

Random access refers to a type of access to memory or mass storage devices in which any location can be accessed directly, without regard for which location was accessed previously. This term is in contrast to sequential access, such as on a tape, where you have to start at the beginning and move towards the end until you reach the location you want.

range

Range is the expression of the exact number of lines of text that EDT will operate on. The simplest form of range is WHOLE, meaning the whole buffer, but you can use expressions such as 20 THRU 30, meaning from line 20 through line 30. Type HELP RANGE while in EDT for more information, or see the *Micro/RSX User's Guide*.

read

When a task is accepting data, it is said to be reading. This is a standard computer term. When you enter a TYPE command, the system must read the designated file from the disk before displaying it at the terminal.

real-time

Micro/RSX is a real-time system. This means it can respond rapidly, almost without any delay, to any outside event. Real-time systems are often used to control industrial processes, but the real-time nature of Micro/RSX means it can respond rapidly under most circumstances to time-sharing users as well as industrial processes.

Record Management Services

See RMS-11.

reentrant

A program or routine that can be entered at the same time by more than one task is called reentrant.

removable media

Some mass storage devices, such as the diskette drives, have removable media. The diskettes can be removed and replaced. Removable media are convenient for files that need not be on line or accessible at all times, such as text files you've finished editing.

See also fixed media.

resident library

A resident library is a block of executable instructions that normally resides in memory. The routines in a resident library are already linked (task built) and can be shared by several tasks at the same time. Resident library routines are not included as part of your program's task image, but they are directly accessible by your program.

RMS-11

RMS stands for Record Management Services. RMS is the more sophisticated and flexible of the two sets of routines supplied on Micro/RSX systems for file operations; the other is FCS. RMS routines open and close files, read from files, write to files, and extend and delete files.

Most programming languages have their own methods of dealing with files that use these routines. In general, RMS routines are not used directly.

See the Advanced Programmer's Kit for more information on RMS-11 and FCS.

routine

A routine is an ordered set of instructions that performs an operation. A routine can be an entire program or a part of a program.

RSX-11 system

The name RSX stands for Resource Sharing Executive. The RSX-11 family of DIGITAL operating systems has been in use and under subsequent development for more than 10 years. RSX-11 systems are real-time systems with features that also allow many users to share the system. Current members of the RSX-11 family include:

- *RSX-11M*, the oldest active member of the family. RSX-11M is a real-time system with many timesharing features. It runs on any PDP-11. RSX-11M is intended primarily for the smaller, older PDP-11s.
- *RSX-11S*, a specialized real-time system used for process control on systems with no mass storage peripherals.
- *RSX-11M-PLUS*, also a real-time system, but with many additional timesharing features, designed for the current line of PDP-11s. RSX-11M-PLUS systems make the best use of the features available on modern PDP-11s.
- *Micro/RSX*, an RSX-11M-PLUS system designed specifically for the MicroPDP-11 with many features to make it easier to use and install, but still a real-time system for multiple users.
- *VAX-11 RSX*, a system that runs under VAX/VMS and emulates an RSX-style system.

- *P/OS*, the *Professional Operating System*, an RSX-11M-PLUS system designed specifically for the Professional 300 series of desktop computers. It has many new features designed for a single user with little interest in or knowledge of computers.

scroll

When more than a screenful of output is sent to a video terminal, the output usually scrolls up. New output appears at the bottom of the screen and eventually disappears off the top, just as if it were on a scroll that is being unrolled at the bottom and rolled at the top.

Use the NO SCROLL or HOLD SCREEN key on your terminal if the output scrolls too fast.

sequential access

This term refers to a method of access to memory or mass storage devices where the records or files are read one after another in the order they appear in the file or volume. A magnetic tape is an example of a sequential access device. If you are half way through a tape and want to read a record that is one third of the way through the tape, you must go back to the beginning and read through until you get to the record that you want.

See also random access.

software

All computer programs are software. Software is all the parts of the computer you cannot touch. Software is the collection of tasks, procedures, and rules associated with the operation of a particular computer system. The operating system is software. EDT is editing software. Office automation is software. The purpose of software is to make the computer easier to use.

Compare with hardware.

source file

A source file is a text file; it is a program in some programming language to be translated into an object module by the MACRO-11 Assembler or a compiler. A source file cannot be run or task built. The Task Builder uses the object module to produce a task file; the task file is a runnable program. See *Programming on Micro/RSX* for more information.

string

A string is a sequence of characters. When you search for a word in EDT, you are searching for a string. The sequence of characters that forms a command is sometimes called a command string. Strings are not always what they seem. The string " Jena" is different from the string "Jena", because the first string includes a space. Similarly, the string "7" is different from "007", even though they are equal numbers.

subroutine

A subroutine is a set of instructions, or a routine, that can be called by other routines. A subroutine performs a secondary function in a larger program.

syntax

Syntax refers to the structure or format that a command must follow. Misspelled words are the most common syntax errors. You can always find the complete syntax for any command in the *Micro/RSX User's Guide*.

system disk

The fixed disk that contains the operating system is called the system disk. You can find the system disk on an RSX system by referencing pseudo device LB:.

system library

All the relocatable routines used by the operating system are defined in the system library. These routines perform various common functions, such as formatting input and output, managing memory, and converting binary numbers to decimal. See *Programming on Micro/RSX* and the Advanced Programmer's Kit for more information.

system task

A task that performs a system-level function is called a system task. Most parts of the operating system, such as EDT or DCL, are system tasks.

See also applications task.

task

The task is the fundamental executable programming unit on Micro/RSX. Almost everything that runs on a Micro/RSX system—EDT, DCL, applications—is a task.

task build

Task build is another term for "link." See Task Builder.

Task Builder

The Task Builder is a translator that uses an object module to produce a runnable task. It allocates the space the task needs to run and makes sure that the symbols used by the program are properly related to one another. This is also called linking. The LINK command invokes the Task Builder.

See Chapter 14 of the *Micro/RSX User's Guide* for more information. *Programming on Micro/RSX* also has information on the Task Builder. For full information, see the *RSX-11M/M-PLUS and Micro/RSX Task Builder Manual,* which is available separately, or as part of the Advanced Programmer's Kit.

task image file

A task image file is a file that contains a runnable program, or task. Most task image files have the file type .TSK and also include a letter C in their directory listing, indicating that they are contiguous and not spread out over the disk. Here is a directory listing for a task image file:

```
QIX.TSK;2        40.        C        01-APR-85 00:01
```

terminal

A terminal is a hardware device with two functions: sending input to the system and receiving output from the system. Most Micro/RSX systems have video terminals. Terminal input usually comes from a typewriter-like keyboard. Output appears on the video screen.

On hardcopy terminals, the screen is replaced by a piece of paper.

text file

Text files are those files that are readable by people, for example, files created using EDT. Text files are often called ASCII files.

UIC

UIC means User Identification Code. This code is a pair of numbers that identifies each user. It is used for file protection and privilege.

User Identification Code

See UIC.

user library

The DCL command LIBRARY allows you to incorporate your own unique set of special routines into a library that you create yourself. In this way, you can store your own commonly used routines and recall them when you need them. See the *Micro/RSX User's Guide* for a description of the LIBRARY command.

utility

A utility is a general purpose task included in the operating system to perform common functions, such as editing or queue management.

video terminal

A video terminal is a terminal with a video screen for accepting output.

Most Micro/RSX systems have video terminals, in particular DIGITAL's VT100- and VT200-series terminals.

virtual terminal

A virtual terminal is a software terminal created by the Executive to pass commands and data to the operating system, as from batch jobs. As far as the system is concerned, a virtual terminal has the same behavior as a physical terminal. See also terminal.

volume

The volume is the largest unit of the file structure. A volume contains files. A volume can be on any medium. The fixed disk and the diskettes are physical media containing files arranged in volumes. In other words, the medium is the physical disk and the volume is the arrangement of the information on the disk. In most cases, you will probably think of the medium and the volume as being the same, but you should be aware of this distinction.

wildcard

A wildcard character is an asterisk (*) or percent sign (%) used to replace parts of a file specification that are not entered in a command. See the *Micro/RSX User's Guide* for complete information on wildcards.

write

When a task is sending output, it is said to be writing. This is a standard computer term. When you issue a TYPE command, the system must read the file from the disk and then write that file to the terminal.

Index

E

F

G

H

I

K

L

M

N

W

USER'S COMMENTS

Your comments and suggestions are welcome and will help us in our continuous effort to improve the quality and usefulness of our documentation and software.

Remember, the system includes information that you read on your terminal: help files, error messages, prompts, and so on. Please let us know if you have comments about this information, too.

Did you find this manual understandable, usable, and well organized? Please make suggestions for improvement.

_____

_____

_____

_____

_____

_____

Did you find errors in this manual? If so, specify the error and the page number.

_____

_____

_____

_____

_____

_____

What kind of user are you? ____ Programmer ____ Nonprogrammer

What do you use the system for? _____

Years of experience as a computer programmer/user: _____

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code _____
or Country

Do Not Tear - Fold Here and Tape

digital

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SSG PUBLICATIONS ZK1-3/J35
DIGITAL EQUIPMENT CORPORATION
110 SPIT BROOK ROAD
NASHUA, NEW HAMPSHIRE 03062-2698

Do Not Tear - Fold Here

digital

AA-Y538B-TC