

Introduction to **RSX-11M-PLUS**

Order No. AA-FD03A-TC

RSX-11M-PLUS Version 3.0

digital equipment corporation • maynard, massachusetts

First Printing, September 1979
Revised, November 1981
Revised, July 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.

Copyright ©1979, 1981, 1985 by Digital Equipment Corporation

All Rights Reserved.
Printed in U.S.A.

The postpaid READER'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-2638

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.

HOW TO ORDER ADDITIONAL DOCUMENTATION

In Continental USA and Puerto Rico call 800-258-1710

In New Hampshire, Alaska, and Hawaii call 603-884-6660

In Canada call 613-234-7726 (Ottawa-Hull)
800-267-6215 (all other Canadian)

DIRECT MAIL ORDERS (USA & PUERTO RICO)*

Digital Equipment Corporation
P.O. Box CS2008
Nashua, New Hampshire 03061

*Any prepaid order from Puerto Rico must be placed with the local Digital subsidiary (809-754-7575)

DIRECT MAIL ORDERS (CANADA)

Digital Equipment of Canada Ltd.
100 Herzberg Road
Kanata, Ontario K2K 2A6
Attn: Direct Order Desk

DIRECT MAIL ORDERS (INTERNATIONAL)

Digital Equipment Corporation
PSG Business Manager
c/o Digital's local subsidiary or
approved distributor

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Northboro, Massachusetts 01532

Contents

Preface

ix

Chapter 1 Getting Started

Hardcopy Terminals and Video Terminals	1-1
Before You Start	1-3
Before You Log In to the System	1-3
Command Line Interpreter (CLI)	1-4
Logging In	1-4
Login Messages from the System	1-5
Correcting Typing Mistakes	1-6
Correcting Mistakes on a Video Terminal	1-7
Correcting Mistakes on a Hardcopy Terminal	1-7
Verifying Corrections	1-8
Deleting Whole Lines	1-8
Ending Input	1-8
Displaying Information on Your Terminal	1-9
Shortening Commands	1-9
Help from RSX-11M-PLUS	1-10
More Help from RSX-11M-PLUS	1-10
A Directory of Your Files	1-11
Devices on Your System	1-11
File Specifications	1-12
Displaying Files on Your Terminal	1-12
Defaults in File Specifications	1-13
Controlling Output to Your Terminal	1-14
Stopping the Action Entirely	1-14
Setting and Showing	1-15
Displaying System Information	1-16
Tasks on the System	1-16
Logging Out	1-16
Summary	1-17

Chapter 2 Creating Files

Creating a File	2-1
EDT, the DIGITAL Standard Editor	2-2
Startup Command Files	2-2
Character Mode Editing	2-3
The Keypad	2-3
Beginning an Editing Session	2-4
Entering Character Mode	2-5
Getting Help on EDT	2-5
Moving the Cursor	2-5
Using the Arrow Keys	2-5
Changing the Direction of Cursor Movement	2-6
Moving the Cursor by Character, Word, and Line	2-6
Inserting Text	2-6
Deleting Text	2-7
Undeleting Text	2-7
Locating Text	2-7
Moving Text	2-8
Leaving the Editor	2-8
Line Mode Editing	2-9
Range Specifications	2-10
Getting Help in Line Mode	2-10
Displaying Lines of Text	2-11
Inserting Text	2-12
Renumbering Text Lines	2-13
Deleting Lines from Text	2-13
Searching Through Text	2-13
Moving and Copying Text within the File	2-15
Replacing Words	2-15
Line Mode Commands Also Used in Character Mode	2-16
Creating a Second File	2-16
Including a Second File in Your Text	2-16
Summary	2-17

Chapter 3 Using DCL Commands

Wildcards and Other Wild Things	3-2
DCL Commands for File Management	3-3
COPY	3-4
CREATE	3-4
DELETE	3-4
DIRECTORY	3-6
EDIT	3-6
PURGE	3-7

RENAME	3-7
TYPE	3-8
DCL Commands for General System Use	3-8
BROADCAST	3-8
HELP	3-9
RUN	3-9
SET DEFAULT	3-9
SET PASSWORD	3-10
SHOW DEFAULT	3-10
SHOW DEVICES	3-10
SHOW TIME	3-11
SHOW USERS	3-11
DCL Commands for the Queue Manager	3-11
PRINT	3-11
SHOW QUEUE	3-13
SET QUEUE	3-13
DELETE/ENTRY	3-13
HOLD/ENTRY	3-14
RELEASE/ENTRY	3-14
STOP/ABORT	3-14

Chapter 4 Automatic Command Entry

Indirect Command Processing	4-1
Indirect Command Files	4-2
Substitution Mode	4-2
Writing Programs with Indirect	4-3
Directives	4-3
Special Symbols	4-3
Labels	4-3
Comments	4-3
Examples	4-4
Batch Processing	4-5
An Example of a Batch Job	4-5
Submitting Batch Jobs	4-6

Chapter 5 Working on the System

Running Tasks Directly	5-2
Creating a Task Image	5-3
The Source Language	5-3
Translating the Source File into an Object File	5-4
Transforming the Object File into a Task Image File	5-6
Running the Task	5-7
Using Subroutines	5-8
High-Level Languages	5-10
Gaining Access to High-level Languages	5-10
Naming Tasks	5-11
Aborting Tasks	5-13
Other References	5-13

Chapter 6 The System in Operation

Hardware and Software	6-1
Applications and Operating Systems	6-2
The Real-Time Control Environment	6-2
The Applications Environment	6-2
The General Purpose Timesharing Environment	6-3
The Purpose of the Operating System	6-3
Control through Privilege	6-4
Control through Priority	6-4
Control through File Protection	6-4
Operating System Resources	6-5
Memory	6-5
The CPU	6-6
Devices	6-7
Stored Information	6-8
The SHOW MEMORY Command	6-9

Glossary

Index

Figures

1-1	LA36 Printing Terminal	1-2
1-2	VT100 Video Terminal	1-2
1-3	VT220 Video Terminal	1-3
2-1	VT100 Keypad	2-3
2-2	VT220 Keypad	2-4
6-1	Structure of Files on a Volume	6-9
6-2	SHOW MEMORY display	6-10



Preface

Purpose of this Manual

This manual is intended to help new users of the RSX-11M-PLUS operating system get started using their system. It provides examples of commonly used commands, as well as instructions for creating and editing files. RSX-11M-PLUS 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 RSX-11M-PLUS on the PDP-11 computer.

Intended Audience

This book is designed for any new user of RSX-11M-PLUS, 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 RSX-11M-PLUS* consists of seven parts:

- The RSX-11M-PLUS warm-up session. This uses a terminal on an RSX-11M-PLUS 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 RSX-11M-PLUS 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 description of how to prepare and run a program.
- A description of how RSX-11M-PLUS systems are used, including a description of system resources and controls.
- 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

The more you know about RSX-11M-PLUS, the more use you will get out of it. Good sources of information about your system are the system manager, in-house documentation, and other, experienced users. You should also read the system documentation.

After you finish the *Introduction*, you should read the *RSX-11M-PLUS Command Language Manual*, which describes the DIGITAL Command Language (DCL). The *RSX-11M/M-PLUS Utilities Manual* also includes information useful to general users of the system.

If you are a programmer, you should see the *RSX-11M/M-PLUS Guide to Program Development* for an introduction to program development facilities on RSX-11M-PLUS.

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.
-

Chapter 1

Getting Started

RSX-11M-PLUS is an *operating system*. The RSX-11M-PLUS operating system is a collection of *software* designed to make it easy to use the PDP-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, RSX-11M-PLUS is called an *interactive system*.

Generally, any mistakes you make in using an RSX-11M-PLUS 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 an RSX-11M-PLUS system is also applicable on other DIGITAL computers, large and small.

Hardcopy Terminals and Video Terminals

You can use either a *hardcopy terminal*, also called a printing terminal, or a *video terminal*. A hardcopy terminal has a *print head* and paper in it. It looks like a typewriter. A video terminal has a screen in place of the print head and paper.

Video terminals are faster and more versatile than hardcopy terminals, but hardcopy terminals leave a permanent record of activity at the terminal.

Both terminals have a keyboard similar to that on a typewriter. Most terminals also have a numerical keypad like that on a calculator.

RSX-11M-PLUS systems support many kinds of terminals; any of these terminals may have special features.

In particular, three terminals manufactured by DIGITAL are likely to be at your *installation*. DIGITAL's standard printing terminal is the LA36 DECwriter II (Figure 1-1). DIGITAL's standard video terminals are the VT100 and the VT220 (Figure 1-2 and Figure 1-3).

Figure 1-1: LA36 Printing Terminal



Figure 1-2: VT100 Video Terminal



Figure 1-3: VT220 Video Terminal



Before You Start

Make sure the terminal you are using is turned on. Look for an ON/OFF (or a 1/0) switch and turn it to ON (or 1). Once your terminal is turned on, you can begin the exercise in the first example. The text explains what is happening.

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 either dollar sign (\$) or right-angle bracket (>) 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
$
```

If you are using a video terminal, 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.

If you are using a hardcopy terminal, the print head usually points to the right of where the next character appears. On most hardcopy terminals, the print head moves to the right after you stop typing, so that you can see what you have typed. When you press another key, the head moves back into the print position before it prints the character.

On the left-hand side of the keyboard is a key marked CTRL, which stands for "control." Press the CTRL key a few times. Nothing should happen. Now press the CTRL key and the C key at the same time. (In examples, the symbol `CTRL/C` indicates that you press the C while pressing the CTRL key.) You should get a prompt composed of three letters and a right-angle bracket (>).

\$ **CTRL/C**
MCR>

The CTRL key and the C key together are called "Control/C". As you will see, you can use the CTRL key with several letters besides C. The symbol **CTRL/Z** means that you press the CTRL key and Z keys together; the symbol **CTRL/O** means that you press the CTRL and O keys together. Some of these combinations appear on the screen (or paper) as a *circumflex* (^), also called an up-arrow, standing for the CTRL key, and the letter you typed. Thus, CTRL/Z sometimes appears as follows:

^Z

The RETURN and CTRL/C keys are both commands to the operating system. Together, they are the best test of whether your terminal is ready to be used. If these two commands have the effects described here, all major components of the terminal, the operating system, and the computer are ready for use.

Command Line Interpreter (CLI)

A command line interpreter, or CLI, is your means of communicating with the operating system.

There are two CLIs available on RSX-11M-PLUS, MCR, the Monitor Console Routine, and DCL, the DIGITAL Command Language. MCR is present on all RSX-11M-PLUS systems; DCL is optional, but is included on most systems with many terminal users.

When you issued the CTRL/C command, the system returned a prompt composed of three letters and a right-angle bracket, like this:

MCR>

This is called an *explicit prompt*, because it identifies the CLI.

DCL is easier, to use and learn than MCR, because DCL commands are English-like words. This manual teaches you how to use DCL.

Note

DCL is used in this manual, because it is more suited to inexperienced users. In fact, all DCL commands are translated into MCR commands for execution by the system. If you intend to use MCR as your CLI, you can make use of this manual by issuing the DCL command SET DEBUG/EXECUTE. This command causes the MCR translation to appear on your terminal before the command is executed. This option is not recommended for inexperienced users.

Now you can log in.

Logging In

Logging in gains you access to the system.

LOGIN is the DCL command that logs you in. HELLO is the MCR command that logs you in.

RSX-11M-PLUS 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 either the LOGIN or HELLO 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]
```

or

```
$ HELLO [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 RSX-11M-PLUS 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, RSX-11M-PLUS 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:

Lowercase letters are not used in the examples in this book, but RSX-11M-PLUS 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.

Correcting Mistakes on a Video Terminal

The DELETE key on video terminals erases the deleted characters.

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.

For example, type the following series of characters on your terminal:

```
$ THIMK DEL DEL NK
```

Here is what you will see on the screen, in succession:

```
$ THIMK
```

After you press the DELETE key twice:

```
$ THI
```

After you enter the corrections:

```
$ THINK
```

The danger of the BACK SPACE key is greatest on video terminals, because what you see on the screen is almost the same as with the DELETE key, but the effect on the operating system is entirely different and incorrect.

Correcting Mistakes on a Hardcopy Terminal

The DELETE key on hardcopy terminals prints the deleted characters.

The DELETE function on hardcopy terminals takes some getting used to.

When you press the DELETE key, you eliminate the character immediately to the left of the next print position. Since the character cannot literally disappear from the paper, the system indicates its disappearance by printing a backslash and the deleted character on your terminal. If you press the DELETE key again, the next character to the left will also be eliminated and reprinted. When you have eliminated what you wish, simply continue typing; the correct characters will be echoed on your terminal.

This can make for a confusing line, as you will see.

For example, type the following series of characters on your terminal:

```
$ THIMK DEL DEL NK
```

Here is what you see, in succession:

```
$ THIMK
```

After you press the DELETE key twice:

```
$ THIMK\KM\
```

After you enter the corrections:

```
$ THIMK\KM\NK
```

The BACK SPACE key is less dangerous on printing terminals, because it simply types over what you typed before, clearly not doing what you wanted it to do.

Verifying Corrections

CTRL/R retypes a line with corrections.

If you are confused by the reprinting of deleted characters on a line, or if you're not sure your corrections were made, use the CTRL/R function. Press the keys marked CTRL and R at the same time. This causes the system to retype your line, without the deletions and corrections. You can then continue with whatever you were typing.

```
$ THINK \KM\NK CTRL/R
$ THINK
```

Note that CTRL/R echoes as follows:

```
^R
```

CTRL/R works only on the current line, before you press RETURN. After you use CTRL/R you are still on the same line. You can continue typing, with additional deletions if you wish. A command will not be entered until you press the RETURN key.

CTRL/R is most useful on hardcopy terminals, but it works on video terminals as well. You can use it to verify that you have typed what you think you typed.

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.

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. RSX-11M-PLUS 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
```

RSX-11M-PLUS 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 RSX-11M-PLUS

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
```

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

More Help from RSX-11M-PLUS

You can also get help from RSX-11M-PLUS 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-86
```

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 DB0: [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 RSX-11M-PLUS systems. Everything on a RSX-11M-PLUS 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

All RSX-11M-PLUS devices have names such as DB1:, which is a disk drive. Notice that the name has two letters, a number, and ends with a colon (:). 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 RSX-11 M-PLUS 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 (DB0:) and directory name ([USER]) where the files are stored. Within the listing, each file is identified by a file name (WHATSOEVER), 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:

```
DB0:[USER]WHATSOEVER.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
File(s)? DB0:[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
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 RSX-11M-PLUS, 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]
      DB0:[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 RSX-11M-PLUS, with the SHOW DEFAULT command. Every file in DB0:[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 DB0:[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, such as DR2:[200,20]FLY.TXT;3.

One important default does not show. If you do not supply a version number, RSX-11M-PLUS 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 DB0: and [USER] are overridden.

Type and enter the following command:

```
$ TYPE DB0:[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   
File(s)? LONG.TXT 
```

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 
```

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

The ABORT command halts execution of a command or a task resulting from a command.

Type DIRECTORY and enter it. Type a CTRL/C before the directory listing is complete.

Now, in response to the explicit prompt, type ABORT DIRECTORY and enter it. The directory will run a few lines more and then a message verifying the abort will appear. The message mentions MCR, because MCR actually does the abort, but your terminal remains set to DCL.

Later you will learn to abort tasks by name as well.

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
```

RSX-11M-PLUS 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 *RSX-11M-PLUS Command Language Manual* 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. RSX-11 M-PLUS 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.

Tasks on the System

The command SHOW TASKS/ACTIVE displays a list of tasks that are active at your terminal. The command SHOW TASKS/ACTIVE/ALL displays a list of all tasks active on the system.

Type SHOW TASKS/ACTIVE and enter it. The system displays the tasks currently active at your terminal. User tasks are identified by the terminal from which they are being run. The display from SHOW TASKS/ACTIVE includes, at least, DCL, which you are running, and something called SHOT with a number. This second task is the SHOW command itself; the T and the number identify the terminal that issued the SHOW command.

```

$ SHOW [RET]
Function?  TASKS/ACTIVE [RET]

```

Logging Out

LOGOUT is the DCL command that logs you off the system. BYE is the MCR command that logs you off the system.

When you are through using RSX-11M-PLUS, you must log yourself out using either the LOGOUT or the BYE 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

```

or:

```
$ BYE RET  
Connect time: 0 hrs 30 minutes 8 secs  
CPU time used; 4 hrs 0 minutes 23 secs  
Task total: 84  
Have a Good Afternoon  
14-MAR-85 14:48 TT2: logged off
```

RSX-11M-PLUS 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 RSX-11M-PLUS 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, it will probably be brought back up in a few minutes.

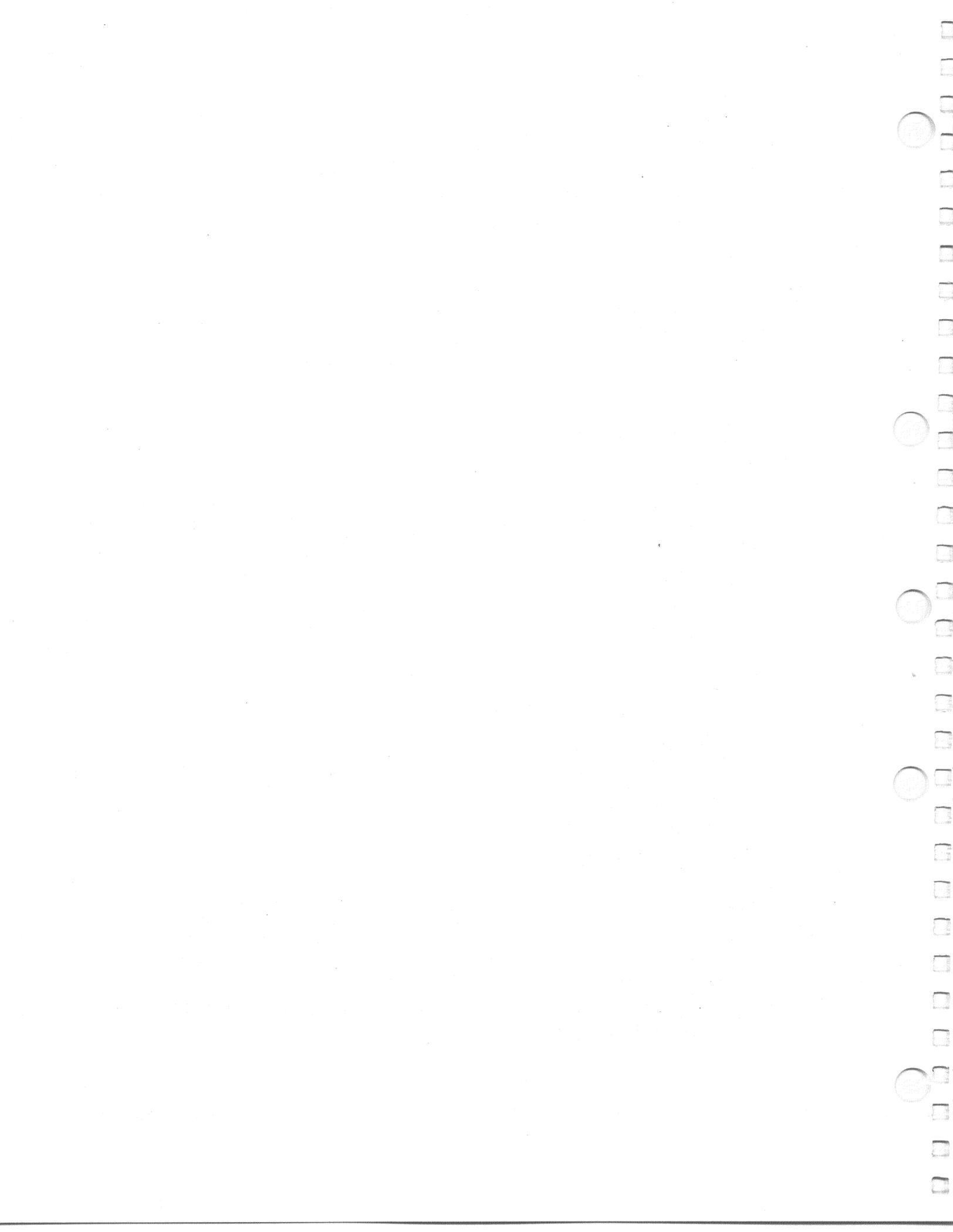
No one is logged in if the system has crashed and then been restarted.

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 get the explicit prompt
- ABORT command to stop a program
- 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 RSX-11 M-PLUS system. You have had to learn some computer jargon and some RSX-11M-PLUS jargon, and you will be learning some more as you continue working with this book. RSX-11 M-PLUS 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 RSX-11M-PLUS 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 4 of the *RSX-11M-PLUS Command Language Manual*.

This chapter tells you how to create and edit files on an RSX-11 M-PLUS 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. RSX-11M-PLUS permits you to give whatever name and file type you like to your files. On the other hand, RSX-11M-PLUS systems also provide default file types for various purposes; these are discussed in Chapter 4 of the *RSX-11M-PLUS Command Language Manual*. (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*, RSX-11M-PLUS 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 RSX-11M-PLUS 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 *EDT Editor Manual*.

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 the *EDT Editor Manual* for information on using character mode.

The Keypad

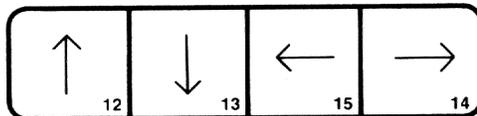
Character (or keypad) editing works on the VT100 and VT200-series video terminals and other terminals with a numerical keypad. In character editing, you request editor functions by pressing keys on the keypad. No RETURN is required to enter the command. Anything you type on the regular keyboard, including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

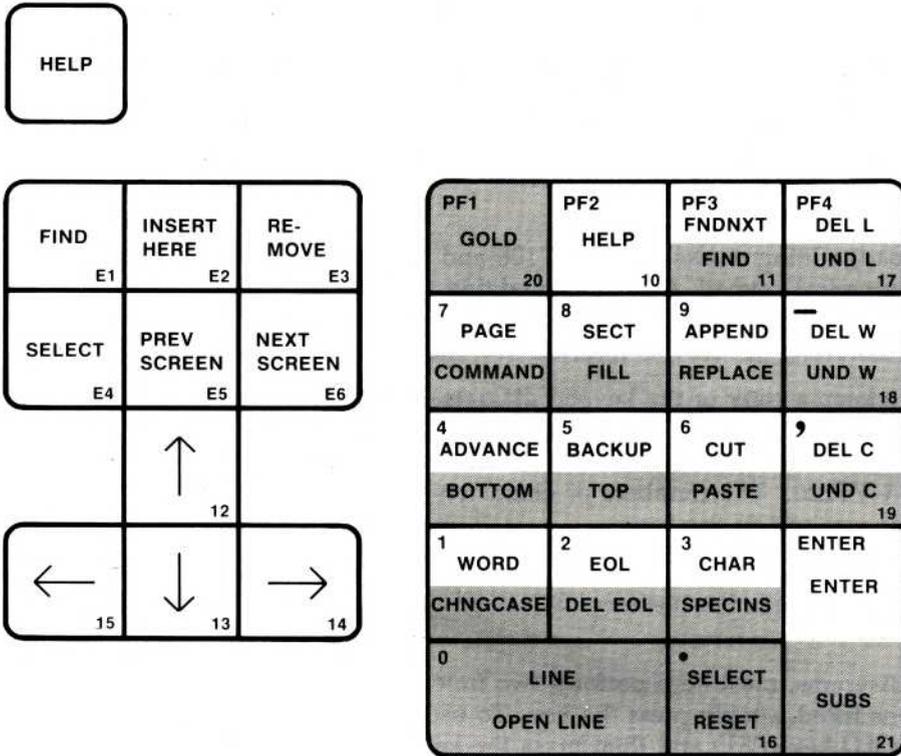
Figure 2-1: VT100 Keypad



PF1 GOLD 20	PF2 HELP 10	PF3 FNDNXT FIND 11	PF4 DEL L UND L 17
7 PAGE COMMAND	8 SECT FILL	9 APPEND REPLACE	DEL W UND W 18
4 ADVANCE BOTTOM	5 BACKUP TOP	6 CUT PASTE	DEL C UND C 19
1 WORD CHNGCASE	2 EOL DEL EOL	3 CHAR SPECINS	ENTER ENTER
0 LINE OPEN LINE	• SELECT RESET 16	SUBS 21	

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]
[EOBI
```

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 the *EDT Editor Manual* 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 VT 100 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 **BACKUP** immediately with **ADVANCE**.

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 **ADVANCE** or **BACKUP**.

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 the *EDT Editor Manual*.

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 DOCTORXEL. 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 the *EDT Editor Manual*.

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 informatdon 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]
A
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.
*
```

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,

*
```

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.

```
[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 + 1) 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 DOCTORYEL. The two lines are not deleted from DOCTORYEL.

```
* WRITE LIKE.DOC 1 THRU 2 [RET]
DBO : [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 (or at the cursor in character mode). 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,
[EOB]  6      The reason why I cannot tell,
*
```

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 the *EDT Editor Manual* 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 the *EDT Editor Manual* for ideas and suggestions on how you can use the advanced features of EDT.



Chapter 3

Using DCL Commands

RSX-11M-PLUS 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 DCL commands most likely to be of everyday use to an average RSX-11M-PLUS system user. The RSX-11M-PLUS help files and the *RSX-11M-PLUS Command Language Manual* 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 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 JXT 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 JXT 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 4 of the *RSX-11M-PLUS Command Language Manual*.

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 RSX-11M-PLUS file management commands in alphabetical order: COPY, CREATE, 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.

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]
```

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 4 of the *RSX-11M-PLUS Command Language Manual*. 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
$
```

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, aborting the DELETE command 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;* 
```

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, RSX-11M-PLUS 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 DB3:[BERRY]:

```
$ DELETE/QUERY *.TMP   
Delete file  DB3:[BERRY] NADINE .TMP ; 1      [Y/N/G/Q] ? Y   
Delete file  DB3:[BERRY] 16 .TMP; 1          [Y/N/G/Q] ? Y   
Delete file  DB3:[BERRY] CHUCK .TMP; 1      [Y/N/G/Q] ? N   
Delete file  DB3:[BERRY] DUCK .TMP;1        [Y/N/G/Q] ? G   
  
The following files have been deleted:  
DB3:[BERRY]HAVANA.TMP;1  
DB3:[BERRY]MOON .TMP ;1  
DB3:[BERRY]ROLLOVER.TMP;1  
DB3:[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 mass storage device you specify. For example:

```
$ DIRECTORY/FREE DB3: [RET]
DB3: has 169709. blocks free, 170961. blocks used out of 340670.
Largest contiguous space = 152069. blocks
15243. file headers are free, 8757. headers used out of 24000.
$
```

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.

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.LIS.

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 *RSX-11M-PLUS Command Language Manual*; 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 RSX-11M-PLUS systems have help files available for users.

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.)

RUN

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

For example, this command runs a program called HIYA:

```
$ RUN HIYA [RET]
```

Chapter 7 in the *RSX-11M-PLUS Command Language Manual* provides more information about running tasks.

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, for example DBO:, and in a particular directory, for example [KILROY]. Therefore, DBO:[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)? DB0: [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)? DB3:[JANE]TOP.CAT [RET]
```

If you want to do a number of things with files from DB3:[JANE], use the SET DEFAULT command, as in this example:

```
$ SET DEFAULT DB3:[JANE] [RET]
$ PRINT TOP.CAT [RET]
```

When you're through with DB3:[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]
DB0:[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 some of the DB: devices on a system:

```
$ SHOW DEVICES DB: [RET]
DB0:    Loaded Type=RP06
DB1:    Loaded Type=RP06
DB2:    Offline Loaded Type=unknown
DB3:    Public Mounted Loaded Type=RP06
```

DB0: and DB1: are both *loaded*, meaning the driver for each is present in memory. They have not been mounted, which indicates they are not presently being used. The device is an RP06, which is a type of disk.

DB2: is reported to be off line and loaded, type unknown. Loaded means is 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.

DB3: is mounted public, which makes it accessible to all users. The device is an RP06; its driver is loaded.

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
```

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 the *RSX-11M/M-PLUS Batch and Queue Operations Manual* 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. You can use 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 *RSX-IIM-PLUS Command Language Manual* 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.

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 substitute a value for the symbol name before executing the command.

Writing Programs with Indirect

As you can see, Indirect can be used to write programs. 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 is one 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. 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 the *RSX-11M/M-PLUS Indirect Command Processor Manual*.

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 the *RSX-11M/M-PLUS Indirect Command Processor Manual*.

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.

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.

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.

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 the *RSX-11M/M-PLUS Indirect Command Processor Manual*.

Examples

Each example is followed by an explanation. See the *RSX-IIM/M-PLUS Indirect Command Processor Manual* for more examples.

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 the *RSX-IIM/M-PLUS Indirect Command Processor Manual* 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.

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 RSX-11M-PLUS.

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 [200/11
$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
                RSX-11M-PLUS  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             $0 <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 SYSTEMDAT is printed on the third page.

For more information on batch processing, see the *RSX-11M/M-PLUS Batch and Queue Operations Manual*.

Chapter 5

Working on the System

Everything that is done on or by an RSX-11M-PLUS system is done by a task or group of tasks. The system itself is a group of tasks and an Executive.

When you issue a command like SHOW TIME, you set a series of tasks in motion:

1. The terminal driver, a part of the Executive named TTDRV, reads what you have typed on your terminal and looks for a task that can do what you have asked. In this case, a task named MCR..., the command dispatcher, answers the call. In these examples the dots are part of the task names.
2. MCR... checks the command over and passes it to the DCL *parser*, a task named ...DCL.
3. A copy of DCL is started and given a name that includes your terminal. For example, if you entered the command from terminal TT10:, the copy of DCL is named DCLT10.
4. DCL10 translates your SHOW TIME command into the equivalent MCR command, which is TIM, and passes that command to a task named ...MCR. Your copy of ...MCR is named SHOT10, for the command and the terminal from which it was entered.
5. SHOT10 goes to the Executive to find out what time it is, translates the time into readable form, and sends it to SHOT10 to be written to your terminal.

This is a simplified description, of course. Each of these tasks may also employ other tasks. All this activity must be coordinated.

The operating system is a collection of tasks cooperating under the direction of the Executive to make your use of the PDP-11 computer easier. DCL is a task that provides you with the means of putting the rest of the system tasks to work.

DCL commands are not executed directly by DCL. Most DCL commands are translated and passed to MCR. Others are passed to system utilities, such as EDT or PIP. These utilities are themselves tasks.

The file management commands you learned in the last chapter rely for the most part on a utility called PIP, the Peripheral Interchange Program. This utility is used to move files around the system, from one peripheral device to another.

DCL makes PIP *transparent* to the user. This means you can use it without seeing it. DCL commands give you access to a number of utilities. Each of these utilities has commands of its own, just like EDT. DCL saves you the trouble of learning the commands for commonly used system utilities. (If you want to see the commands in the format the utility uses, use the SET DEBUG command. See the HELP file for more information.)

See the *RSX-11M/M-PLUS Utilities Manual* for more information on what the system utilities will do. If there is some utility function that you want to use that does not seem to be available under DCL, you can run the utility at your terminal.

Running Tasks Directly

You can run utilities directly from your terminal.

Type `RUN $PIP` and enter it. The dollar sign (\$) tells the system where to look for PIP.

PIP will come back with its own prompt (PIP >). You can then issue commands directly to PIP, instead of through DCL. These commands must be PIP commands and not DCL commands.

Type `/LI` and enter it. You get the same list of files produced by the DCL command `DIRECTORY`. This occurs because the `DIRECTORY` command causes the DCL task to issue a `PIP/LI` command to the system.

When you are through using PIP, type `CTRL/Z`, to signify end-of-input and a return to DCL.

```
$ RUN $PIP RET
PIP> /LI
Directory DBO: [USER]
10-JUN-85 10:00

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
```

```
PIP> CTRL/Z
$
```

By contrast, the `EDIT/EDT` command starts EDT running, but you can also start EDT with the command `RUN $EDT`.

The `RUN` command is itself a task. Loosely speaking, a task is a "computer program." Strictly speaking, a task is the fundamental executable unit in *RSX-11M-PLUS* systems.

As you know, a program is nothing more than a set of procedures. As such, it can be written on paper. In France, two centuries ago, tables of complex mathematical functions were prepared by hundreds of clerks, each following simple written-out procedures.

Thus, there were programs before there were computers. There was software before there was hardware.

Creating a Task Image

This section demonstrates how a written procedure becomes an executable task image that will run on the PDP-11 hardware.

You do not need to be a programmer to do this demonstration. It is not a programming demonstration. It is a demonstration of how the system does things.

A written procedure becomes an executable task image in four steps:

1. You must design the procedure using a source language to define it.
2. You must type the source program as a text file using an editor or the CREATE command.
3. You must translate the source file into a machine-readable *object module* using an assembler or compiler.
4. You must transform the object module into an executable task image using the LINK command.

For this demonstration, the first two steps have already been done for you. A source program called HIYA.MAC has been designed and entered as a text file. In the demonstration, you will translate this source file into an object module and then transform the object module into a task image.

The Source Language

In the directory for the USER account, you will find a file named HIYA.MAC. The MAC file type identifies the file as a source program written in the MACRO-11 Assembly Language. Display the file on your terminal using the TYPE command. '

```
$ TYPE HIYA.MAC [RET]
        .TITLE  HIYA
        .LIST   TTM
        .NLIST  BEX
        .ENABL  LC

; MACRO LIBRARY CALLS
        .MCALL  EXIT$$,QIOW$,DIR$,GTSK$

INDPB:  QIOW$  IO.RLB,5,1 ,,IOST
OUTDPB:  QIOW$  IO.WLB,5,1 ,,IOST,< " 40>
SYSDPB:  GTSK$  SYSBUF

        LOCAL EQUATES
BSIZE=80.                                ACCEPTS NAMES UP TO 80 CHARACTERS

        LOCAL DATA BUFFERS
MSG1:    ASCII  /Could I have your name please?/
MSG1L=.-MSG1          ; THE LENGTH OF MSG1

MSGMP:    ASCII  /RSX-11M-PLUS calling /
MSGMPL=.-MSGMP        ; THE LENGTH OF MSGMP

MSGM:    ASCII  /RSX-11M calling /
MSGML=.-MSGM          ; THE LENGTH OF MSGM

BUFF:    BLKB   BSIZE   ; SET UP BUFFER LENGTH = BSIZE
```

As you see, the first two steps of the process have been accomplished. The program has been designed and entered. Do not worry if you cannot understand it. You will see what it does in a few minutes. Take particular note of the lines labeled MSG 1: and MSGMP: and MSGM:.

A source language is your means of defining the procedure you want followed. MACRO-11 is one of several source languages that can be used on RSX-11M-PLUS systems. It was chosen for this demonstration because all RSX-11M-PLUS systems include MACRO-11. In fact, most of the system tasks were originally written in MACRO-11.

All source files, regardless of language, are encoded in ASCII (American Standard Code for Information Interchange). ASCII is a univesal code used to convert characters we can read into *binary machine code* that the computer can work with.

Before HIYA.MAC can run on the system, it must first be translated from text into binary machine code, and then be transformed into a task image.

Translating the Source File into an Object File

The MACRO command invokes the MACRO-11 Assembler. The MACRO command uses the default file type MAC. The MACRO-11 Assembler assembles, or translates, a MACRO-11 source file into an object file.

Using the DIRECTORY command and wildcards, check the directory of the USER account for files named HIYA. You should find only a single file with the file type MAC.

```
$ DIRECTORY HIYA.*;* [RET]
Directory DB0:[USER]
10-JUN-85 10:07
HIYA.MAC;1          6.          27-JAN-85 08:58
Total of 6./6. blocks in 1. file
```

Type the command MACRO and enter it. Respond to the **File(s)?** prompt with the file name HIYA. Notice that you have given only the file name in response to the MACRO command prompt. You do not have to give a file type or version number.

```
$ MACRO [RET]
File(s)? HIYA [RET]
```

The MACRO command invokes the MACRO-11 Assembler. The version number defaults to the most recent vesion, as usual. The file type defaults to MAC because the MACRO-11 Assembler can only assemble MACRO-11 input files. Therefore, it is simplest to give your MACRO-11 source files the MAC file type.

After a short delay, the prompt returns, signifying completion of the assembly. If you like, while you are waiting, you can issue a SHOW TASKS/ACTIVE command to see that the assembly is taking place.

Now check the directory again. You should find two files named HIYA. The new file has the file type OBJ, identifying it as an object file. This file type is the default for output files produced by the MACRO-11 Assembler.

```
$ DIRECTORY HIYA.*;* [RET]
Directory DB0:[USER]
10-JUN-85 10:08
HIYA.MAC;1          6.          27-JAN-85 08:58
HIYA.OBJ;1          2.          10-JUN-85 10:08
Total of 8./8. blocks in 2. files
```

Notice that the files are not the same size. The object file is smaller, because it has been translated from the less efficient, readable text, including comments, into the efficient binary machine code, without the comments.

This completes the third step of the process, creating an object module. Object files are machine-readable only. People cannot read object files.

Now print the object file on your terminal using the TYPE command. This will work on either a video or hardcopy terminal, but it will use a lot of paper on the hardcopy terminal. You may prefer to wait until you have a video terminal to try it out.

The terminal will buzz, beep, or ring the bell and issue dozens of line feeds while printing the file. The file itself will seem to consist mostly of confused jabber. This is because the assembler has translated the instructions that were in text form in the source file into binary machine code.

```
$ TYPE HIYA.OBJ RET
3@/L4:br@4:"@)s)@@(4H}h0((Could I have your name please?RSX-11M-PLUS
calling@Aw @3APw @A'w h}3Iq(h)*f3
```

```
DSW = IOS*T =
$
```

The jabber is the terminal's attempt to interpret binary machine code as if it were ASCII.

Caution

Displaying binary files on your terminal may cause your terminal to hang. If this happens, try pressing CTRL/Z or CTRL/C, or try turning your terminal off and on.

Now display the file again, but this time be ready with the NO SCROLL or the HOLD SCREEN key, so you can look at the first part of the file. In the midst of the jabber, you will see the two messages that the HIYA task will eventually issue. These are not translated from ASCII, because the machine does not need to read them. Since they are only read by people, they are left as they were typed.

Although the object file is machine-readable, it still is not a task.

Here is what happened so far. The assembler has checked the source code in the source file for errors. It has translated the source code from text to binary machine code, and it has assigned *relocatable* (provisional) addresses to all parts of the program. These addresses are assigned as if HIYA were going to have the computer all to itself.

Finally, the assembler has constructed a *symbol table* containing all symbols referenced by the program. Some of these symbols, called *local symbols*, are defined in the program. Other symbols are *global*, meaning they are not defined in the program. The OBJ file contains references to all local and global symbols, but the resolution of the global symbols still remains to be done.

In HIYA.MAC, examples of local symbols include BSIZE, which is the number of characters a program will accept, and MSG1L, which is the length of the first message. An example of a global symbol is \$CBDSG, a routine from the *system library*, which converts a binary number to a signed decimal ASCII representation. The global symbol \$CBDSG is not defined in the program; it is defined in the system library. (LB:[1,1]SYSLIB.OLB)

Again, the description is greatly simplified.

Transforming the Object File into a Task Image File

The LINK command invokes the Task Builder. The LINK command uses the default file type .OBJ. The Task Builder transforms an object file into a task image file.

Type the command LINK HIYA, after the prompt appears.

```
$ LINK RET  
File(s)? HIYA RET
```

Even though there are now files in the directory named HIYA, you still do not have to give a file type with the LINK command. The LINK command invokes the Task Builder, and the Task Builder can only process object files. Therefore, the file type defaults to OBJ.

Thus, you can see that if you give your source file a file type that properly identifies the language used in the file, you do not have to include the file type with subsequent commands used to turn the source file into a runnable task image.

Again, there will be a short delay while the Task Builder transforms the object file into a task image file.

Look at your directory for files named HIYA. There should now be three of these files:

1. Your original source file with the MAC file type
2. The object file made by the assembler with the OBJ file type
3. The task built by the LINK command with the .TSK file type

```
$ DIRECTORY HIYA.*;* RET  
Directory DB0:[USER]  
10-JUN-85 14:06  
  
HIYA.TSK;1          5.      C  10-JUN-85 14:04  
HIYA.OBJ;1          2.      10-JUN-85 10:08  
HIYA.MAC;1          6.      27-JAN-85 08:58  
  
Total of 13./13. blocks in 3. files
```

Notice that the .TSK file is larger than the OBJ file, perhaps as large as the MAC file. It is larger than the OBJ file, because it includes the symbol definitions that were left unresolved by the assembler. (The relative sizes of source, object, and task image files vary considerably from task to task. Only in a simple case, such as this, can you expect the relative sizes to be clear.)

The C in the directory listing identifies the .TSK file as a contiguous file. None of the other files you have created has been contiguous. A noncontiguous file may be scattered all over the disk; its *file header* contains a map of the blocks used in the file. Task image files must be contiguous, meaning they are not scattered, but are together in one location. Not all contiguous files are task images, but all task image files are contiguous.

Since the task image file is contiguous, it can be located quickly on the disk and brought into the system. (Many data files that must be used by tasks are contiguous for the same reason, to save time on I/O.)

Task image files are not readable, but they are different from object modules. Display the task image file on your terminal using the TYPE command. If you have a hardcopy terminal, you may prefer to wait for a chance to try this on a video terminal.

Once again, the terminal tries to interpret the file as if it were ASCII. The jabber produced is different from the jabber produced by the attempt to print the object file. It takes much longer to finish.

The two ASCII messages are still there and are still readable, if you can catch them.

```
$ TYPE HIYA.TSK [RET]
```

```
@0~SYSSYSYTICLzI@Tx~zVxTI@Could I have your name please?RSX-11M-PLUS call-  
ing @zAw @APw
```

```
/,f f
```

```
v W%
```

Here is what happened to the object module as a result of the LINK command. The Task Builder resolved the reference to the undefined global symbol \$CBDSG by finding its definition in the system library. This made it possible for the Task Builder to complete the symbol table constructed by the assembler.

The Task Builder also changed the relocatable addresses into addresses that the system can use. As you recall, the assembler assigned addresses as if the resulting task would have the computer to itself. However, no task that is run under a multiuser system like RSX-11M-PLUS has the computer to itself. Therefore, the Task Builder applied a special addressing scheme that makes it possible for the task to run in competition with other tasks.

Resolving symbol references and assigning addresses are major functions of the Task Builder. The Task Builder may also build tasks out of more than one object module, as will be demonstrated later in this chapter.

Running the Task

The RUN command installs, runs, and removes tasks. It uses the default file type TSK.

Now type and enter a RUN command. When you receive the prompt Task?, type the name HIYA. You do not need to specify the file type, because the RUN command defaults to TSK.

```
$ RUN [RET]  
Task? HIYA [RET]  
Could I have your name please?  
Bo Diddley [RET]  
RSX-11M-PLUS calling Bo Diddley  
$
```

When the HIYA task requests your name, type it. In this case, your name is *data* to be processed by the task. The processing consists of returning your name in a different form.

The DCL commands themselves are tasks that process data. Proper data for a RUN command is the name of a task image file. The RUN command processes this data by:

1. Finding the file
2. Naming the task
3. *Installing* the task
4. Running the task
5. *Removing* it when its run is completed

As it is generally used, the RUN command is actually a combination INSTALL-RUN-REMOVE command.

The INSTALL command is the task that makes HIYA known to the system by placing a *Task Control Block (TCB)* in the *System Task Directory (STD)*. The INSTALL command also *requests* that the task be run. As soon as the Executive grants this request (when space is available in memory), the task is run. Once the task has finished running, the REMOVE command takes over. The REMOVE command is a task that makes HIYA unknown to the system by taking the TCB out of the STD.

Privileged users can install tasks with a separate INSTALL command. That is how system tasks are made available to everyone. Nonprivileged users can only install tasks using the RUN command. These tasks only stay installed while they are in use.

Since the RUN command includes the INSTALL command, you will sometimes get error messages referring to the INSTALL command instead of the RUN command.

Using Subroutines

Now run EDT using the command line shown in the example. This command line directs EDT to use HIYA.MAC as an input file and, after editing it, to create the output file NEWHI.MAC.

```
$ EDIT/EDT/OUTPUT:NEWHI.MAC HIYA.MAC
* 'EXIT$$' RET
8          .MCALL EXIT$$,QIOW$$,DIR$,GTSK$
* 'EXIT$$' RET
50         EXITS          ; LEAVE
*I RET
TAB CALL TAB STARS TAB; CALLS STARS SUBROUTINE RET
CTRL/Z
* EXIT RET
DB0:[USERINEWHI.MAC;1 150 LINES
$
```

Previously, when you invoked EDT, you did not specify an output file. You did, however, specify by default an output file of the same name and type as the input file but with a version number one higher than that of the input file.

This time, you are giving the output file a new name, one that does not appear in the directory, and thus will be version 1. The file NEWHI.MAC;1 will be created when you exit from EDT.

As before, both the new and old form of the file being edited will still exist, but where before the new form had only a new version number (by default), now the new form will have an entirely new file specification (by explicit action).

When EDT returns with its prompt, search for the second instance of the string EXITS. This is the line in the MACRO-11 program that causes the task to exit when it has finished running.

You are going to insert a new line ahead of the line so that the program will call a *subroutine* named STARS. This reference to STARS puts another undefined global symbol in the program.

Insert the following line:

```
TAB CALL TAB STARS TAB; CALLS STARS SUBROUTINE
```

The symbol `TAB` means that you press the TAB key. Tab positions are set every eight spaces. The TAB key moves you to the next tab position, which may or may not be eight spaces from where you are.

The semicolon (;) marks the beginning of a comment. The text preceding the semicolon is code.

Now exit from EDT.

Show a directory of only MAC files. There may be others, but you will at least have files named HIYA.MAC, NEWHI.MAC, and STARS.MAC. STARS.MAC is the source file for the subroutine for which you just inserted a call.

An object module can contain a subroutine, which can greatly alter the performance of a task. You can link more than one object module to form a task image.

Now use the MACRO command to turn the source program NEWHI.MAC and the source subroutine STARS.MAC into object modules. STARS.OBJ includes the global definition of the global symbol STARS.

After issuing the first MACRO command, wait for the return of the prompt before issuing the second MACRO command.

```
$ MACRO NEWHI [RET]
$ MACRO STARS [RET]
```

Type and enter the LINK command, naming both object modules: NEWHI, STARS. Separate the module names with a comma. When the Task Builder is through, run NEWHI;

```
$ LINK NEWHI, STARS [RET]
[RET]
$ RUN NEWHI [RET]

Could I have your name please?
Rachmaninoff [RET]
RSX-11M-PLUS calling Rachmaninoff
```

As you see, the STARS subroutine has altered the performance of the task. When you built HIYA.TSK the first time, you used only one object module. References to local symbols were resolved by the assembler. The Task Builder had to resolve the reference to the global symbol \$CBDSG, which is defined in the system library. NEWHI.MAC, however, contains a reference to an additional global symbol, STARS, which is defined in STARS.OBJ. The Task Builder will always look to the other object modules specified in a LINK command for global symbol definitions before it goes on to the system libraries to try to resolve them.

Try linking NEWHI.OBJ without the STARS.OBJ; you will get an error message stating that the symbol STARS is undefined.

In addition to linking subroutines, the Task Builder can also go to the system library of object modules supplied with your system or to libraries created at your installation for special purposes.

High-Level Languages

So far, this demonstration has concentrated on the MACRO-11 Assembly Language, because the MACRO-11 Assembler is bundled with every RSX-11M-PLUS system. Most systems will include one or more high-level languages in addition to MACRO-11.

Each computer must have an assembly language. This language is designed at the same time as the hardware. In general, one line of assembly language translates into one line of machine language. The ASCII text of one line of MACRO-11 code becomes one line of binary machine code.

This is a major distinction between an assembly language and the high-level languages. The high-level languages—such as COBOL or FORTRAN—are compiled. Each statement in a high-level language typically translates into more than one line of machine language.

Machine language is different on every machine, but FORTRAN is much the same from machine to machine and company to company. Naturally, there are differences related to how the hardware works, but, generally speaking, a FORTRAN program is transportable from one computer to another and from one operating system to another. All it takes is a FORTRAN compiler.

The compiler is designed to translate source files into binary machine code. Thus, whereas input files—source text files—are transportable, output files from the computer—object files—are different on different computers.

Each high-level language is designed for a particular kind of use. Programmers say that any program can be written in any language, but, in fact, different languages were designed with different applications in mind. FORTRAN was designed for scientific applications. FORTRAN stands for FORMula TRANslator. COBOL stands for COMMON Business Oriented Language and was designed for commercial applications. BASIC stands for Beginners All-purpose Symbolic Instruction Code and is widely used as a first programming language.

All these languages are supported on RSX-11M-PLUS. This means that language compilers, libraries, and other necessary software have been prepared to run on a PDP-11 computer with the RSX-11M-PLUS operating system.

Although the MACRO-11 Assembler and the various high-level language compilers translate source files written in different languages, the assembler and compilers both produce object files. The Task Builder processes object files without regard to the source language. Thus, the object file is a common goal for any assembler or compiler operating on RSX-11M-PLUS systems, a target at which they all aim.

The high-level languages are supported, but they are not bundled. This means that they must be purchased separately. Some of them may not be available at your installation, and there may be other languages at your installation that are not mentioned here.

Gaining Access to High-level Languages

DCL includes commands analogous to the MACRO command for the most popular languages. There is a FORTRAN command, and two different FORTRAN compilers are supported. There is also a COBOL command. If your installation includes compilers for which specific DCL commands are not supplied, you can use the RUN command to run the compiler at your terminal.

BASIC-PLUS-2 programs can be compiled, but a special procedure is used. BASIC-PLUS-2, the form of BASIC supported on RSX-11M-PLUS, is an enhanced version of BASIC used in commercial applications.

In most cases, high-level languages include more conveniences for the programmer than assembly languages. Programs in assembly language must be explicit in everything they do. You may have noticed that HIYA.MAC goes into great detail at the labels MSGM: and MSGMP: to determine the length of each message. That kind of detailed programming is rarely needed in a high-level language. By the same token, programs written in a high-level language should be more readable. They include a higher level of information than assembly language programs.

On the other hand, MACRO-11 was named after its capacity for using macros. Macros allow programmers to gather a number of lines of assembly language code under one name. Thus, a macro (like the EXITS macro referred to when you edited HIYA.MAC into NEWHI.MAC) actually assembles into more than one line of binary machine code. With this capacity for including macros, MACRO-11 can, in effect, sidestep some of the detail of assembly language programming while remaining close to the actual workings of the computer.

Regardless of your programming language, familiarity with MACRO-11 can benefit you as a programmer. All higher-level languages supported on RSX-11M-PLUS systems include the capability of calling assembly language subroutines. These subroutines can often increase the speed and efficiency of your program.

A further demonstration of running tasks follows.

Naming Tasks

Tasks initiated by the RUN command are named after the terminal from which the RUN command was issued. Tasks initiated by other commands are named after the command itself and the terminal from which it was issued.

Run PIP, but do not issue any commands. Instead, enter CTRL/C. The explicit DCL prompt shows that you have interrupted the execution of the PIP task and can issue a DCL command.

```
> RUN $PIP [RET]
PIP> [CTRL/C]
DCL>
```

Type SHOW TASKS/ACTIVE and enter it. (If you get an error message, issue the command again. If you still get an error message, type CTRL/Z to terminate PIP and try again.)

```
DCL> SHOW TASKS/ACTIVE [RET]
SHOT10 (TT10:)
TT10 (TT10:)
[CTRL/C]
$
```

DCL returns a list of the tasks active at your terminal. The SHOW task is identified by a name in the form SHOTnn. The task name gives you some information about the task itself

The SHO indicates that the task was initiated with a SHOW command.

The T indicates that the command was issued from a terminal.

The nn is the number of the terminal.

The RUN command differs from other commands in how tasks resulting from it are named. Tasks initiated by a RUN command are named directly after the terminal from which the command was issued. Thus, in the example, PIP is named TT10 after the terminal from which the RUN command was issued.

More than one task at a time can be run from your terminal. The /TASK_NAME qualifier to the RUN command overrides standard task naming.

Type `RUN $PIP` and enter it. You should get the `PIP >` prompt.

Now enter `CTRL/C`. Type and enter `RUN HIYA` in response to the `DCL` prompt. You should get a `RUN` error message saying that the task name is already in use.

```
$ RUN $PIP [RET]
PIP> [CTRL/C]
DCL> RUN HIYA [RET]
RUN -- Task name already in use
```

This may seem confusing, because you know you are not running `HIYA`.

`HIYA` is the name of the file that contains the task image, but it is not the task name referred to by the error message. The message refers to the name in the form `TTnn`, where `nn` is the number of the terminal from which the `RUN` command was issued. Since `PIP` is already using that name, you cannot simply issue another `RUN` command to run a task. Instead, you must give the task some other name for your second `RUN` command.

Type `RUN/TASK_NAME:LURG HIYA` and enter it. As you see, both `PIP` and `HIYA` are now active. In the example, `PIP` has the name `TT10` and `HIYA` has the name `LURG`.

After `HIYA` has finished, issue `CTRL/Z` to leave `PIP`.

```
PIP> [CTRL/C]
DCL> RUN/TASK_NAME:LURG HIYA [RET]
PIP> [RET]
Could I have your name please?
Lash LaRue [RET]
PIP> [RET]
RSX-11M-PLUS calling Lash LaRue
$
PIP> [CTRL/C]
```

Run `PIP` again, but use the `/TASK_NAME` qualifier to keep `PIP` from being named after your terminal. Name it `LOLA` instead. Since `PIP` is now installed under a different name, the `PIP` prompt is changed to `LOL`. Make sure it is really `PIP` by issuing a `/LI` command to the `LOL` prompt. You should get a directory listing.

```
$ RUN/TASK_NAME:LOLA $PIP [RET]
LOL> /LI [RET]

Directory DBO:[USER]
10-JUN-85 14:33

A.A;i          1.          12-FEB-85 13:16
AZ.CMD;1       1.          13-MAR-85 15:38
COPY.CMD;2     1.          16-FEB-85 12:27
EDT.CMD;5      1.          28-MAR-85 13:30
LOL>
```

An explanation of how to abort a renamed task, such as `LOLA`, follows.

Aborting Tasks

You can abort tasks by name as well as by command.

In an earlier section, you learned to abort tasks by aborting the command that initiated them. You can abort tasks by name as well, using the /TASK qualifier to the ABORT command. This qualifier works very much like the /TASK_NAME qualifier to the RUN command.

If you tried to abort PIP through the ABORT RUN command, you would receive the error message:

```
ABO -- Task not in system
```

You cannot use the ABORT RUN command here, because that command looks for a task whose name includes your terminal number. Since you overrode that task name, as confirmed by SHOW TASKS/ACTIVE in the following example, you must abort the task by specifying the name you gave it.

Type ABORT/TASK LOLA and enter it as shown.

```
LOL> CTRL/C
DCL> SHOW TASKS/ACTIVE RET
DCL... (TT10:)
SHOT10 (TT10:)
LOLA (TT10:)
```

```
LOL> CTRL/C
DCL> ABORT/TASK LOLA RET
```

```
15:55:18 Task "LOLA" terminated
          Aborted via directive or CLI
          and with pending IO requests
```

Other References

This completes the terminal warm-up session.

Now you should look up the commands you have learned in the *RSX-11M-PLUS Command Language Manual*. You will find there are many additional ways of using the commands you have already learned.

After you have looked at the *RSX-11M-PLUS Command Language Manual*, you may want to look in the *RSX-11M/M-PLUS Utilities Manual* for information on the full functions of the system utilities.

If you are a programmer, you should also read the documentation for your programming language. There are many features of the high-level languages that are found only on DIGITAL versions of these languages, and DIGITAL versions may differ from one operating system to another.

The last chapter of this manual will teach you a little about how what you do at your terminal fits with the operations of the rest of the system. It includes a demonstration of the SHOW MEMORY command, which produces a live-action picture of what is happening on the system.



Chapter 6

The System in Operation

Thus far, this manual has taken the point of view of the terminal user. For most of the time that you are using the operating system, you will feel as if you are the only user.

In fact, some systems may have dozens of terminals as well as other *peripheral devices* for passing input to the system and receiving output from it, but the operating system makes it possible for each user to act independently.

This chapter presents some generalizations about RSX-11M-PLUS. You can follow up on these generalizations to learn the particulars of the system you are using. As an individual user, you will generally not need to concern yourself about most of the topics presented here, but some points may become important to you as you gain more experience.

Hardware and Software

RSX-11M-PLUS offers a wide range of services and utilities; the system supports many kinds of input and output devices. RSX-11M-PLUS is designed as a general purpose system that can be customized for each installation. License holders can choose from unbundled software options, or users can write their own system software.

Once the hardware and software elements have been chosen, the process of *system generation* ties these choices together into a customized system. The system you are using has been generated to include your installation's mix of hardware and software.

Because each installation is different, not all installations have all the capabilities discussed in this and other system manuals. For example, some systems include support for DECnet, which means they can be tied into networks of computers, and some do not.

In general, systems with heavy terminal use—general purpose timesharing systems—will include most of the system generation options that affect the terminal user and the programmer. Other options are removed during system generation to save the memory these options would otherwise require.

The RSX-11M-PLUS operating system runs on processors with 22-bit *addressing*; it provides software features that take advantage of the hardware features of these processors. These software features are outside the scope of this manual, but include the virtual terminal, which is used by the batch processing subsystem; multiuser tasks; and Resource Accounting.

The variety of possibilities can be confusing to a new user, but the purpose is to make possible an operating system closely suited to the needs of your installation.

You are not likely to encounter any absent system generation options. It is much more likely that some task or utility you need will simply not be installed when you need it. In such a case, you can usually have your system manager install the task for as long as you need it and then remove it when you no longer need it.

Applications and Operating Systems

RSX-11M-PLUS systems are *real-time* systems. This means that the system is designed to respond rapidly, either to input from users or to input from application tasks.

RSX-11M-PLUS systems are also *multiuser* systems. This means that more than one user can have access to the system at any time.

The combination, a real-time, multiuser system, allows real-time activity—such as data acquisition or control of an industrial process—to occur at the same time as program development from interactive terminals.

Typically, commercial and industrial data-processing applications fall into one of three categories: real-time control, applications processing, and general purpose *timesharing*.

In the real-time control environment, the operating system is used as a tool. This is also true of the applications environment. In these environments, rapid response is the more important capability of an operating system. In the general purpose environment, parts of the system are used as tools and *throughput*, the total volume of work performed in a period of time, is the more important capability.

The Real-Time Control Environment

The real-time control environment is one in which the principal function of the operating system is to handle rapid data movement with little human interaction. Typical examples of such environments are steel rolling mills, oil refineries, and communication switching centers. When certain conditions are met—a thickness, a temperature, a delay, the system must respond rapidly—closing a valve, slowing a motor, throwing a switch.

The operating system, of course, does not know about steel rails, or long-distance calls, or milling machines. The principal function of the system in a real-time control environment is to receive, verify, reply to, and move data messages rapidly and without error.

The Applications Environment

The applications environment is one in which the greatest part of the system's resources are given over to continuous, high-volume data handling. Again, rapid, error-free handling of data messages is the principal function of the system, but instead of controlling a process, the messages update a data base under the control of the applications task.

In the applications environment, most terminal users have no direct contact with the operating system. Typically, the terminals they use are *slaved* to the applications task, and the terminal users communicate directly with the task rather than with the operating system. The terminal users enter data for processing by the applications task. The task opens and closes files, updating and altering information as it is entered. In the applications environment, there are few users of the operating system itself.

The General Purpose Timesharing Environment

The general purpose timesharing environment is one in which program development and testing is a major activity. Terminal use is prominent in this environment, meaning that the system spends a great deal of time waiting for terminal input. Assembling, compiling, and task building make heavy use of the CPU. In this environment, there may be many users at one time, but most of the time, they are thinking, looking up commands, or the like between keystrokes.

In the general purpose timesharing environment, the system's interactive facilities are heavily used. These include DCL, the editors, the utilities, and the program development tasks, such as the assembler or compiler and the Task Builder. Because human input is so slow compared to input from machines, the system's real-time capabilities are not as important as in the applications or real-time control environment.

Often, one system may be used to develop programs to be run on another system. These programs might be intended for real-time control or as an applications task. On the other hand, the programs may perform special computations, such as modeling, statistical analysis, or forecasting, which are to be run on the same system they were developed on.

RSX-11M-PLUS systems can be used in any of these three environments. Or, an installation can combine any or all of these kinds of functions.

Every installation is a custom installation, a combination of hardware and software designed to fulfill the needs of the installation. Therefore, the best sources of information about the operating system are the system manager, in-house documentation, and other people who use it.

The Purpose of the Operating System

The purpose of any operating system is to make the computer hardware easier to use.

The operating system is under the control of the Executive, a set of routines that coordinate all activities in the system, including supervision of input and output, allocation of resources, task execution, and operator communication.

The Executive is the *kernel* of the operating system. The operating system consists of the Executive plus the utilities, the programming languages, *device drivers*, and other system components. The installation consists of the operating system plus the applications tasks, as well as the computer and all its hardware devices.

The operating system manages the software and hardware resources of the system. This management requires that the operating system do four kinds of things:

- Keep track of all resources
- Enforce policy on who gets which resources, when, and how much
- Allocate the resources according to system policies
- Reclaim the resources when they are no longer needed

The operating system uses three control mechanisms that it applies to all system users: privilege, priority, and file protection.

Control through Privilege

System users are divided into privileged and nonprivileged groups. Installations usually have only a few privileged users. The system manager is always privileged. Privileged users have access to every part of the operating system. Nonprivileged users can use most of the operating system, but they cannot change it. For example, nonprivileged users can issue a `SHOW TIME` command; privileged users can also issue a `SET TIME` command to change the system time.

Usually your lack of privilege is of no concern. Most privileged functions have to do with system control and maintenance, not with common use of the system facilities. If you should need access to a privileged function, you can usually arrange it through a privileged user.

In addition to privileged users, there are privileged terminals and privileged tasks. A privileged terminal is any terminal with a privileged terminal logged in on it. Privileged commands can be issued only from a privileged terminal.

Privileged tasks are tasks that perform operations normally considered to be the domain of the Executive or that can affect the operations of the system as a whole. Nonprivileged users can use privileged tasks. Many system tasks are privileged, but it is the task, not the user, that has the privilege.

Only privileged users can permanently install tasks in the system. Nonprivileged users install nonprivileged tasks with the `RUN` command, but these tasks are removed as soon as they have finished running.

Control through Priority

Privileged users can build, install, and run tasks at priorities of from 1 through 250. Nonprivileged users can only install and run tasks at the default priority of 50, but they can build tasks with any priority.

A task's priority determines the preference given its requests for services from the Executive. In particular, a task's access to memory and to the CPU are determined by priority. The highest priority task that has access to all the resources it needs is granted control of the CPU.

In systems that combine real-time applications with less urgent work, the real-time applications are given higher priority, because they must be processed immediately to give the response demanded in a real-time environment.

In timesharing systems, interactive tasks, such as editors, are generally installed at a higher priority than tasks that run unattended, such as the Task Builder. This means that users at their terminals are less likely to have to wait for a response.

How the system uses priority to control access to system resources is explained later in this chapter, in the discussion of memory.

Control through File Protection

The system also controls access to information through file protection, which determines which users and tasks can use or alter the contents of files. You can set the protection status of your own files with the `SET PROTECTION` command.

Operating System Resources

There are four basic resources under the control of the operating system. These are:

- Memory, the system's workspace, where active tasks, their data, and the Executive itself are located
- The Central Processing Unit, or CPU, the part of the computer that executes instructions or computes
- Peripheral devices, the input and output devices, including mass storage disks, line printers, terminals, and the like
- Stored information, the file system, the organization of files into directories and directories into volumes

Each task has different resource requirements. Involved scientific and statistical calculations, "number-crunchers," use a great deal of CPU time and memory, but make few demands on the system's devices or the file system. Conversely, printing a long listing can tie up an output device like a line printer for hours, while using little memory and only a few seconds of CPU time.

Memory

The size of memory is measured in *words*. The unit of measure is a K, which stands for kilo and is equal to 1024, or 2^{10} .

Memory should not be confused with mass storage. Mass storage, such as disks or tapes, is where files are kept when no immediate use is being made of them. Memory is the random-access workspace in which all instructions and data in current use by the system are kept. These instructions and data can be accessed immediately.

Instructions are executed in memory after having been read from a file on a mass storage device. Instructions act on data in memory. The data has either been created in memory or read from a file on a mass storage device.

Part of memory—the amount depends on the choice of system generation options—is occupied by the Executive and the operating system.

Included in the Executive's *partition* in memory is the Dynamic Storage Region, commonly called *pool*. The pool contains dynamic information on the current state of the system. The pool space is generally available to the Executive and to privileged tasks, to use as it is needed. The information in the pool enables the Executive to perform its functions. RSX-11M-PLUS systems relegate some this information to secondary pool.

All memory is divided into partitions, subdivisions *dedicated* to a particular task or to system functions. All partitions have a name and a size. Some partitions are used by the system, such as SYSPAR, the partition used by MCR. If you do not specify a partition when you install and run a task, it will be installed in the default partition, named GEN.

An installed task has an entry in the System Task Directory (STD), but it is not resident in memory or competing for other system resources. For example, EDT is usually installed even if no one is using it. It is dormant until some terminal user issues the EDIT/EDT command. A *dormant task* uses no memory, but it is quickly available when needed.

An *active task* is a task that has been requested to run. It is usually *resident* in memory, either as a ready-to-run task or as a blocked task, meaning a task that is waiting for some needed resource. Only tasks that are resident in memory can have access to the CPU.

When the Executive receives a request to activate a dormant task that is not in memory, it allocates the required memory resources, brings the task into memory (if there is space available in its partition), and puts the task into competition for system resources with other tasks resident in memory. If there is no memory space available in the task's partition, the task is still considered active and is placed in a queue by priority with other active, waiting tasks.

Checkpointing is the process of temporarily removing a partly executed task from memory to make room for a higher-priority task. If the partition in which the task is to run is fully occupied, checkpointing can clear the space.

The Executive accomplishes checkpointing in the following fashion. The prerequisites are that the waiting task must have a higher priority than the task resident in memory, and the task resident in memory must have been built, or installed, as a checkpointable task. If these prerequisites are met, the Executive saves the resident task in its incompletely executed state and writes it to a reserved checkpoint space on the disk. Then, the higher-priority task is brought into the memory space that has thus been freed. When the higher-priority task has finished running or when some other space in the partition becomes available, the checkpointed task is returned to memory to continue its processing.

Checkpointing depends on differences in priority, but many tasks in the system run at the default priority of 50. This means that tasks of the same priority can block each other. The Executive gets around this problem through a variation on checkpointing called *swapping*. With swapping, the Executive regularly lowers the priority of tasks resident in memory so that other, waiting tasks will have a higher priority and can thus effect checkpointing. The checkpointed tasks return to their regular priority. Once the new tasks are resident, their priorities will also be lowered, enabling the first tasks to checkpoint them in turn.

The CPU

The CPU can only execute one instruction at a time, but it does that fast. Almost everything that is done on the system must pass through the CPU to get done, but you will probably never address the CPU directly. Access to the CPU is under the control of the Executive. Tasks must be resident in memory to gain access to the CPU. Only one task at a time can have control of the CPU. Multiprogramming is possible because the task operation almost always involves more than just the CPU.

The Executive's control of the CPU is accomplished through *significant events*. A significant event causes the Executive to reevaluate the eligibility of active tasks to run. When a significant event occurs, the Executive scans the list of active tasks and runs the highest-priority task that is ready to run. Here are the most important significant events:

- The completion of input or output. If a task is waiting for I/O or cannot continue its I/O because the I/O device is unavailable, then it has no further need for control of the CPU.
- The execution of a task.
- The execution of an Executive directive that causes a significant event. System directives are services provided to the programmer by the Executive that make it possible for tasks to synchronize their own execution, get device and system information, communicate with other tasks, and generally communicate with and work through the system.
- The execution of the *round-robin scheduler*. The round-robin scheduler is a form of timesharing that overcomes the Executive's tendency to give the most CPU time to tasks that appear first in the Active Task List (ATL). The round-robin scheduler rotates entries in the ATL and then causes a significant event to occur after a given period of time. This significant event causes the Executive to look for a higher-priority task to take over the CPU. The time interval is usually one-tenth of a second.

Once again, the explanation is greatly simplified. Fortunately, the system can display a moving picture of these processes as they occur.

Devices

Device control is another important element of the system. The SHOW DEVICES command lists the devices on the system. Devices are often called peripherals, because they are located outside the computer. All input to the computer and output from it is handled by the peripheral devices.

In the manual, we have given attention to three devices found on all RSX-11M-PLUS systems. These are the terminal, the printer, and the mass storage disk. The terminal is two devices combined: the keyboard is an input device, and the screen or print head is an output device. The printer is strictly an output device.

The terminal and printer are both *record-oriented devices*. This means they handle information one record at a time. A record is one line of information. In other words, record-oriented devices have a limited capacity for storing information. As soon as they have received one record, they must process it before going on to the next record.

Mass storage disks and DECTapes are *file-structured devices*. This means that these devices, which allow random access, are capable of working with the system's file services.

Many kinds of devices are supported on RSX-11M-PLUS systems. Several DIGITAL terminals and printers among record-oriented devices and a dozen or more file-structured disk and tape devices are supported. This support consists primarily of device drivers that enable the system to handle I/O to the devices.

These devices have different physical characteristics. On the simplest level, disks for one type of disk drive will rarely fit on any other type of disk drive. Programmers do not need to concern themselves with these physical differences. Programs accept input from devices and send output to devices, but the coding is independent of the physical characteristics of the devices for the most part. Tasks perform I/O on LUNs (Logical Unit Numbers), which the programmer or operator can assign to specific devices before the program uses the devices.

In addition to the devices discussed here, the operating system can control industrial and commercial devices, such as lathes or communications switching apparatus. The disk and tape drives are complex machines that are controlled by the operating system through device drivers. By the same token, lathes and switchboards can also be controlled by the operating system, but users must write their own device drivers to suit these machines.

Furthermore, in some environments, you may find software handling I/O as if the software were a physical device. For instance, RSX-11M-PLUS supports *virtual terminals*, software that appears to the system to be a physical terminal. In these cases, the devices look the same to the operating system. All are treated as part of the system's resources under the control of the Executive. Virtual terminals are used in batch processing, which is described in Chapter 4.

DCL provides a number of ways to associate LUNs with physical devices. See the *RSX-11M-PLUS Command Language Manual* for more information. At this point, you need only understand that almost all device use is transparent on RSX-11M-PLUS systems.

Stored Information

Control of stored information, or data processing, is the purpose and function of the computer. Every key you strike, every task that runs, is information being processed. On a less abstract level, the file system is the system's way of organizing stored information so that you can use it.

Most of the information to be processed by the operating system is located on disks and tapes. These disks and tapes are magnetic *media*, which means that the information on them is stored in the form of magnetic impulses.

Because RSX-11M-PLUS is a disk-based system, most of the discussion that follows refers to disks. A disk is a random-access medium, meaning that all the information on it is equally accessible. Most magnetic tape, however, is a sequential-access medium, meaning that to get to any particular record on the tape, you have to read the tape from the beginning until you get to the record you want. Tape is economical but slow; disks are more expensive but faster.

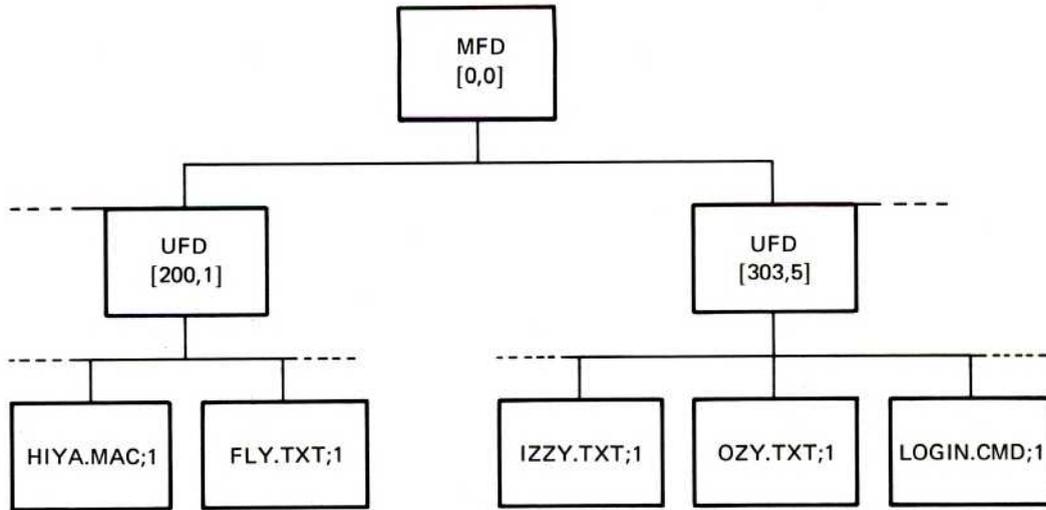
The magnetic impulses are read and written by electromagnetic heads, much like the recording and playback heads on a tape recorder. These heads and the movement of the disk are controlled by a *device controller*. This is all hardware, however, and tells us nothing about the organization of the information on the disk.

Users access information in files. The files are organized in directories, and these directories are organized in volumes located on a mass storage disk. The volume is the software equivalent of the magnetic medium in hardware. However, although magnetic media differ, the operating system considers all file-structured disk volumes to be identical. This is because all file-structured disk volumes are organized in the same format, called Files-11. Files-11 volumes are created through the DCL command INITIALIZE.

An explanation of how the hardware and software handle I/O is beyond the scope of this manual. In the simplest terms, the file system makes it unnecessary for you to worry about the physical location of your files on the disk. Each Files-11 volume has an MFD, or Master File Directory, which is a file of user file directories on that volume. (See Figure 6-1.) Each directory is a file containing the names of a user's files and pointers to each file header. The file header contains information about the physical location of the file's contents on the disk. The system finds files by using the information in the directories and file headers.

Tasks running on RSX-11M-PLUS access data within files through one of two sets of routines: FCS, or File Control Services, and RMS-11, or Record Management Services. Both FCS and RMS-11 organize information within files. RMS-11 was developed after FCS and allows more complex file organization than is possible with FCS.

Figure 6-1: Structure of Files on a Volume



ZK-282-81

The SHOW MEMORY, Command

The DCL command `SHOW MEMORY` displays information about the status of the system. In particular, it displays information about the contents of memory and the task that currently controls the CPU.

If you issue the `SHOW MEMORY` command from a hardcopy terminal, you get a snapshot of memory at the moment you issued the command. Issue the command twice, so that you will have two snapshots to compare.

If you issue the `SHOW MEMORY` command from a video terminal, you can see a moving picture of tasks coming into memory and leaving it, and you can watch the changing control of the CPU. If you have access to the computer room, you may find a video terminal permanently running the memory display task, which is called RMD.

Log in to the USER account and issue the following command:

```
$ SHOW MEMORY RET
```

The `SHOW MEMORY` command requires that RMD be installed on your system. If the command does not work, you should find out if the display is available elsewhere. Most systems have a video terminal running the display somewhere near the computer. If not, ask your system manager for assistance.

Figure 6-2 is a sample `SHOW MEMORY` display, followed by an explanation of the numbered elements.

You should understand that the display changes only once a second; it is possible that tasks are in and out of memory or in and out of the CPU in less than that time. Nonetheless, RMD displays graphically how the Executive controls the contents of memory and use of the CPU.

- ⑬ The size and location of the partitions and tasks. The lines of asterisks are proportional representations of the amount of memory each partition or task occupies.
- ⑭ The system error count sequence recorded by the Executive.

There is a more detailed explanation of the meaning of this display in the *RSX-11M-PLUS Command Language Manual*. For now, you should simply watch it changing. Notice which tasks seem to occupy the most memory or use the most CPU time. There may be one large task using most of memory and much of the CPU time. Or, a number of smaller tasks may be sharing memory.

As explained earlier, tasks are usually named for the terminal from which they originate. In any case, write down the names of some of the more prominent tasks from the SHOW MEMORY display, and see if some knowledgeable system user can identify them for you. This will give you a better idea of what is going on at your installation and of how the system manages its resources.

If you are at a video terminal, press CTRL/Z to cancel the SHOW MEMORY display.

The SHOW MEMORY display helps the system manager observe the system running and find bottlenecks, such as a task running at a higher priority than it should be and, therefore, taking too much CPU time. As you know more about the system, this display will become more meaningful to you.



Glossary

abort

Stopping a program from running before it is finished is called aborting the program. When nonprivileged users log out, LOGOUT aborts any programs they have running 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.

active task

All tasks that have been requested to run are included in the list of active tasks. This is a priority-ordered list of all tasks resident in memory or checkpointed. Active tasks are in active competition for system resources. Checkpointed tasks are out of memory and waiting for system resources.

addressing

All computer operations depend on the ability to address specific memory locations. Put simply, the longer the possible address, the more locations you can reference. In its original design, the PDP-11 used 16-bit addresses, which meant that programs could not use any more memory than could be addressed in 16 bits. Later, optional memory management hardware extended that addressing ability to 18 bits. Finally, the more recent PDP-11 processors include memory management hardware that extends the addressing ability to 22 bits. The greater the addressing ability, the larger the possible program.

RSX-11M-PLUS systems run on 22-bit processors and can address from 256K bytes of memory up to 3.8 megabytes of memory. Since RSX-11M-PLUS was specifically designed for the 22-bit processors, it can take advantage of features available only on 22-bit processors.

See your processor handbook and the *RSX-11M/M-PLUS Task Builder Manual* for more information on addressing capabilities.

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 *RSX-11M-PLUS Command Language Manual* 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.

assembler

The MACRO-11 Assembler takes ASCII files written in the MACRO-11 Assembly Language and assembles them into a relocatable object module suitable for processing by the Task Builder.

assembly language

MACRO-11 is the assembly language on RSX-11M-PLUS.

Assembly language is used to generate binary machine code. See *MACRO-11 Assembly Language*.

binary machine code

Binary machine code is the internal instruction format actually used by the computer. It is called binary because only two characters-0 and 1-are used in this code.

Here is an instruction in MACRO-11 Assembly Language:

```
MOV R0,R1
```

This is an instruction to move the number in register 0 to register 1. The assembler translates this instruction into the following binary machine code:

```
0001000000000001
```

You will rarely see binary machine code. While this is the only form that the computer can actually use, it is difficult for humans to use. Thus, humans are provided with languages, compilers, and the assembler, and these are used to generate binary machine codes.

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 that have bad blocks on them, but if there are many bad blocks, you should consider replacing the disk.

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 Processor.

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 RSX-11M-PLUS 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 RSX-11M-PLUS systems, the bootstrap is a routine included in the PDP-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.

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 *EDT Editor Manual* 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.

checkpointing

Checkpointing is the process by which the Executive makes memory space and processor time available to tasks according to their priority. Also called "rolling out."

If a higher-priority task is ready to run and no memory is available, then lower-priority tasks will be temporarily removed, or checkpointed, to make room for the higher-priority task. The lower-priority tasks are saved on the disk exactly as they were when interrupted. When memory is available, the tasks are returned to memory and take up exactly where they left off.

Your task can be checkpointed without your knowing it. If your task seems slow or refuses to accept input, it may be checkpointed. Checkpointing is an automatic process. Your task will probably return to active status shortly after it is check-pointed.

circumflex

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

It is used on RSX-11M-PLUS systems to indicate that you have typed a control character.

CLI

CLI stands for command line interpreter. The CLI is a system feature that makes it possible for you to communicate with the operating system from your terminal. RSX-11 M-PLUS provides two CLIs. For further information, see DCL and MCR.

command

A command, when executed, is an instruction to the software to perform a particular action. For instance, the following command directs RSX-11 M-PLUS 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 *RSX-11M-PLUS Command Language Manual* for more information on DCL commands.

command line interpreter

See CLI.

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.

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:

```
HIYA.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.

data

Data is a general term used for any representation of facts, concepts, or instructions in a form suitable for communication, interpretation, or processing.

In the demonstration in this manual, when the HIYA task asks for your name, it is asking for data. It then processes this data by inserting the name you give into a greeting.

Many commands are tasks. When they prompt you for command elements, they are asking you for data to process.

DCL

DCL stands for DIGITAL Command Language. DCL provides a means of communication between the user and RSX-11M-PLUS. 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.

Compare with MCR.

dedicated

In the computer industry, a system resource—an I/O device, task, or the entire system—is said to be dedicated when it is assigned to a single application or purpose.

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 *RSX-11M-PLUS Command Language Manual* and in the help files.

In RSX-11M-PLUS 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.

delete

Removing a file header from a directory and deallocating its reserved space is called deleting the file. The file cannot be accessed after a delete operation because it cannot be found. The disk space occupied by the file is available to any user.

You can use the SET PROTECTION command to protect your files against deletion.

The terminal key marked DELETE or $\langle X \rangle$ deletes previously typed characters.

EDT, the DEC Editor, provides commands for deleting text from buffers.

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 found on RSX-11M-PLUS systems include terminals, line printers, disk drives, and 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.

device controller

Each physical device included in the system is associated with a hardware device controller that consists of electronic circuits. The device controller serves as the interface between the processor and the device hardware.

device driver

Each device included in the system has a device driver, which is the software interface between the Executive and the device controller.

DIGITAL Command Language

See DCL.

directive

Some requests for system functions are called directives. This manual refers to Executive directives, which are requests to the Executive for system services, and indirect command directives, which are instructions to the indirect command file processor.

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 RSX-11M-PLUS systems. Disks are high-speed, random-access devices. There are several kinds.

disk-based system

On a disk-based system, such as RSX-11M-PLUS, 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 RSX-11M-PLUS system is kept on a disk because of the disk's speed and capacity. The disk with the system on it is called the system disk.

dormant task

A dormant task is installed but not yet requested to run.

See also task state.

DSR

Dynamic Storage Region. See pool.

Dynamic Storage Region

See pool.

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. RSX-11M-PLUS 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.

Error messages are explained in the documentation.

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.

explicit prompt

The three-letter prompt that identifies the command line interpreter or other system task is called an explicit prompt. For example:

```
DCL> DIGITAL Command Language  
MCR> Monitor Console Routine  
PIP> Peripheral Interchange Program
```

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 RSX-11M-PLUS systems are implemented through these routines, or a similar set of RMS-11 routines.

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 RSX-11M-PLUS.

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 header

Each file has an associated file header block that includes information needed by the file system to find and use the file. Some of the information in the file header block is displayed by the DCL command DIRECTORY.

file-structured device

A file-structured device is a device, such as a disk or tape, that can accept data organized into files. See also volume. Compare with record-oriented device.

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:

DB0: [USER] FLY .TXT ; 1

The device name is two letters and a number followed by a colon (:), such as DB0:

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 *RSX-11M-PLUS Command Language Manual* for more information about file specifications and their component parts.

Here are some file types commonly used on RSX-11M/M-PLUS systems:

File Type	Use
.BAS	BASIC-11 source program. System default.
.BAT	File containing batch processing commands. System default, RSX-11M-PLUS only.

File Type	Use
.BP2	BASIC-PLUS-2 source program. System default.
.CBL	COBOL source program. System default.
.CMD	Indirect command file. System default.
.COR	SLP file used to correct a source file. System convention.
.DAT	File containing data, as opposed to code. System convention.
.FTN	FORTTRAN source program. System default.
.LOG	Log of batch processing session. System default.
.LST	Listing file. System default.
.MAC	MACRO-11 source program. System default.
.MAP	Task builder map file. System default.
.MLB	Macro library. System default.
.OBJ	Object module output from assembler or compiler. System default.
.ODL	File containing Overlay Descriptor Language to be used by the Task Builder. System default.
.OLB	Object module library. System default.
.SYS	Bootable system image. System default.
.TMP	Temporary file. System convention.
.TSK	Task image file. System default.
.TXT	Text file. System convention.
.ULB	Universal library.

Some of these file types are system defaults, automatically supplied and sought by the software. You can override these defaults, but in most cases they are convenient. Other file types are system conventions, also not required, but commonly used and recommended for your use.

Files-11

Files-11 is the name of one of the file structures used on RSX-11M-PLUS. Volumes from other operating systems can be converted to Files-11 structure with FLX, the File Transfer program.

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.

global symbol

A global symbol is a value defined in one object module that can be used in other object modules. Many global symbols are defined in the system library. Global symbols are identified and defined by the Task Builder.

See local symbol.

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.

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.

Hardware and software must be in harmony for the system to work at full efficiency.

See software.

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 RSX-11M-PLUS system, but you can also write your own help files.

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.

implicit prompt

The right angle-bracket prompt (>) is called the implicit prompt. It indicates that a command line interpreter is ready to receive input. When you type a command to the system and enter it, the implicit prompt does not return until the action of the command has completed. If you press the RETURN key, you will get an implicit prompt, but there will be another prompt outstanding, which will appear when the task has completed. The presence of the implicit prompt is not required to enter commands, but if it is not present, the terminal may not be ready to accept command input.

The following example shows what can happen if you do not wait for the implicit prompt to return following a command.

```
① >MACRO HIYA [RET]
② LINK HIYA [RET]
      TKB -- *FATAL*-FILE HIYA.OBJ;1 HAS ILLEGAL FORMAT
③
④ >LINK HIYA [RET]
⑤
```

- ① The MACRO command was issued to assemble the file HIYA.MAC.
- ② Before the implicit prompt returned, indicating completion of the MACRO command, the user attempted to LINK HIYA. This failed, because the file HIYA.OBJ was not yet complete.
- ③ A prompt was issued following the failed LINK command.
- ④ The MACRO command completed, causing another prompt to be issued. The LINK command issued in response to this prompt is completed successfully.
- ⑤ A prompt was issued indicating completion of the LINK command.

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

RSX-11M-PLUS is an interactive system. This means that you and the operating system communicate directly using the terminal. The RSX-11M-PLUS 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 *EDT Editor Manual* for more information.

K

K is a unit for measuring the size of memory or similar resources. K is short for kilo and is used roughly to mean 1000, although formally K is equal to 2^{10} , or 1024.

kernel

The irreducible minimum of the Executive is called the kernel. It is the core of the operating system. The kernel runs in kernel mode, which has no hardware protection at all and no restrictions on machine use. In most cases, however, Executive and kernel are synonyms.

label

A label is one or more characters used to identify a source language statement or a line in a program. In the demonstration program for this manual, HIYA.MAC, the label MSG1: identifies the line that contains the first message sent by the program.

The term label is also used to identify a particular Files-11 volume.

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 also language library, object library, resident library, system library, and user library.

license

Each copy of the operating system is sold to run on a particular PDP-11 processor and no other. This is called a license. There are several varieties of license to suit particular situations.

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.

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 RSX-11M-PLUS stay on the system disk until needed, at which time the system loads them into memory.

local symbol

A symbol that cannot be referenced outside its defining object module is called a local symbol. Local symbols are identified and defined by the assembler or compiler.

See also global symbol.

In MACRO-11 programs, the term local symbol is also used for a label that cannot be referenced outside its local symbol block.

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 RSX-11M-PLUS 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.

log

A log is a record of performance. In this manual, the term refers to a file produced by a batch processor in which is recorded all terminal activity resulting from a QMG batch job. A QMG batch job consists of one or more user batch jobs, each of which has a separate section of the log. Batch processing is included on RSX-11M-PLUS systems only.

logical unit number

See LUN.

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 RSX-11M-PLUS 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.

LUN

LUN is an acronym for logical unit number. A LUN is a number associated with a physical device during a task's I/O operations. Each task can establish its own correspondence between LUNs and physical device units. See the *RSX-11M-PLUS Command Language Manual* for more information.

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 RSX-11M-PLUS 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.

See the *RSX-11M/M-PLUS Guide to Program Development*, the *IAS/RSX System Library Routines Reference Manual*, the *PDP-11 MACRO-11 Language Reference Manual*, and your processor handbook 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 RSX-11M-PLUS system and its components reside on a mass storage device most of the time.

MCR

MCR stands for Monitor Console Routine, the prime interface with the system. MCR commands go directly to the system utilities and installed tasks. Most MCR commands use initials or special characters in strict syntax, rather than English-language words. Compare with DCL.

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. The RSX-11M-PLUS operating system 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.

mnemonic

Mnemonic is an aid to memory. It is pronounced ne-MON-ic. PIP is a mnemonic for Peripheral Interchange Program.

Most mnemonics are acronyms.

mode

Mode refers to a possible condition or state of operation. For example, EDT can operate in line mode or character mode.

monitor

Command line interpreters (CLIs) are sometimes called terminal monitors. They monitor the activity on your terminal when nothing else is happening. Commands at monitor level are directed to the operating system. CTRL/C gives you access to monitor level from within a task. See also DCL and MCR.

Monitor Console Routine

See MCR.

multiprogramming

A multiprogramming system, such as RSX-11M-PLUS, can run more than one task at a time without interference among tasks.

multiuser

A multiuser system, such as RSX-11M-PLUS, 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 disks or tapes your private device through the command MOUNT/NOSHAREABLE.

nonprivileged

Most RSX-11M-PLUS 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.

For more information see the *RSX-11M/M-PLUS and Micro/R SX Task Builder Manual*.

octal number

A number in the base 8 numbering system is called an octal number. Only the numerals 0 through 7 are used in this system. If a number includes an 8 or a 9, it cannot be an octal number. Octal numbering is used in computer systems because it is easy to convert to the binary numbers that are actually used by the computer.

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 *The RSX-11M/M-PLUS and Micro/R SX Debugging Reference Manual* 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 RSX-11M-PLUS 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.

parse

Parsing is breaking a command string into its elements to interpret it.

A PRINT command without a file specification, or with illegal characters in the file specification, will not parse correctly.

partition

A partition is a predetermined, contiguous area in memory in which tasks are loaded and executed. Each partition has the following characteristics:

1. A name
2. A defined size
3. A fixed starting address

The default partition is named GEN. If you do not specify a partition, your tasks will run in GEN. Other partitions are reserved for other system functions, such as DRVPAR, the partition for device drivers.

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.

pool

The Dynamic Storage Region (DSR) is commonly called the pool. The pool is part of the Executive's partition in memory. The pool contains the Executive's data base.

print head

The print head is the moving mechanism on a hardcopy terminal that prints characters on the paper. On many hardcopy terminals, the print head rests just to the right of the next character position when it is not printing. The print head is an output device.

priority

Priority is a rank assigned to a task to determine its precedence in obtaining system resources when the task is run. Priority is usually set when the task is built. Priority can also be set when the task is installed or when it is run.

The default priority is 50. Only privileged users can build, install, or run tasks at any other priority. Once a privileged user has built or installed a task to run at a higher priority, however, nonprivileged users can run it at that higher priority.

Priority numbers go from 1 to 250 (decimal) with the higher number having priority.

privilege

Privilege determines the level of system access allowed to a user or a task.

privileged

On RSX-11M-PLUS 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.

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 RSX-11M-PLUS 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.

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.

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 *RSX-11M-PLUS Command Language Manual* 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 **RSX-11M-PLUS** 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 *EDT Editor Manual*.

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

RSX-11M-PLUS 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 *RSX-11M-PLUS* means it can respond rapidly under most circumstances to time-sharing users as well as industrial processes.

Record Management Services

See *RMS-11*.

record-oriented device

A record-oriented device is a device such as a line printer or terminal that deals with information one record, or one line, at a time. See *file-structured device*.

reentrant

A program or routine that can be entered at the same time by more than one task is called reentrant.

remove

A task is removed when its name and address are taken out of the System Task Directory (*STD*). A task must have been installed before it can be removed.

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.

RMD

RMD is the Resource Monitoring Display invoked by the *SHOW MEMORY* command. RMD displays the current contents of memory, currently active task, and other system information.

RMS-11

RMS stands for Record Management Services. RMS is the more sophisticated and flexible of the two sets of routines supplied on RSX-11M-PLUS 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.

round-robin scheduler

The round-robin scheduler is a form of time sharing that gives tasks of equal priority equal access to the CPU. The Executive tends to give CPU time to the first task in the System Task Directory (STD). The round-robin scheduler rotates the entries in the STD. The round-robin scheduler also causes a significant event after a given time interval. The significant event causes the Executive to search the STD for a task that is eligible to run. The first task in the STD gains access to the CPU. After a time interval, the round-robin scheduler again rotates the entries in the STD and causes another significant event. The new first task in the STD gains access to the CPU and so forth. In this way, tasks of the same priority have an equal share of CPU time.

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.

STD

STD stands for System Task Directory. The STD is a list of all tasks installed on the system. You can display the STD through the DCL command SHOW TASKS/INSTALLED.

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.

swapping

Swapping is a system generation option. It is a variation on checkpointing where tasks of equal priority have their swapping priorities systematically raised and lowered so that they can checkpoint each other and all gain access to the CPU and memory.

SY:

SY: is the pseudo device that stands for the user's default device. Your system can be located on any of a number of different physical devices. Using SY: in commands ensures that the command will go to your current default device even though you have changed devices since you wrote the task.

symbol

A symbol is a representation of something by reason of relationship, association, or convention. In a programming context, a symbol (sometimes called a variable) is an entity that must be defined, or given a meaning, so that it can be used.

symbol table

Each task has a symbol table constructed by the assembler or compiler and completed by the Task Builder, which identifies and defines all symbols used in the task.

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 *RSX-11M-PLUS Command Language Manual*.

system disk

The 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 generation

System generation is the process of tailoring an operating system for a particular hardware configuration with modifications and additions to the software configuration as well.

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.

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 RSX-11M-PLUS. Almost everything that runs on an RSX-11M-PLUS 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. For more information, see the *RSX-11M/M-PLUS and Micro/RSX Task Builder Manual*.

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:

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

task state

An installed task may be in either of two task states:

1. Dormant—installed, but not yet requested to run
2. Active—requested to run. It remains active until it exits, terminates, or is aborted.

An active task may be in either of two substates:

1. Ready-to-run—competing with other tasks for CPU time on the basis of priority
2. Blocked—unable to compete for CPU time, or because a needed resource is not available

terminal

A terminal is a hardware device with two functions: sending input to the system and receiving output from the system. Terminal input usually comes from a typewriter-like keyboard. Output appears on terminals in two ways, depending on the terminal type.

- Hardcopy terminals keep a permanent record of output on paper.
- Video terminals have a video screen for receiving output.

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.

throughput

The total volume of work performed by a computer system over a given period of time is called its throughput, that is, how much has been put through the system.

TI:

TI: is the terminal input pseudo device; TI: is your terminal. You can use TI: in place of your terminal's device name (TTn:) in commands. The terminal you are using will always be TI: regardless of whether it is the same number or type you were originally using.

time sharing

A time-sharing system is a system in which each user gets equal computer time in turn. This is in contrast to the allocation based on need and priority in a real-time system.

Although RSX-11M-PLUS is fundamentally a real-time system, the round-robin scheduler provides a form of time sharing called time slicing.

See also round-robin schedule and real time.

transparent

A function of an operating system is called transparent when the user can use the function without seeing it. For instance, the DIRECTORY command uses the system task PIP, the Peripheral Interchange Program, but the user need not issue any commands directly to PIP to use PIP.

This is a computer industry term.

UFD

See directory.

UIC

Each RSX-11M-PLUS user has a two-number identification code enclosed in brackets that can be used (with password) for logging in. The number is in the form [g,m], with "g" giving the user's group number, and "m" giving the user's member number. Users working together often have the same group number, because the default file protection setup permits group members to use each other's files without hindrance.

user batch job

A user batch job is a complete terminal session consisting of commands to be processed, each preceded by a dollar sign (\$). The user batch job begins with \$JOB, which logs the job in, and ends with #EOJ, which logs the job out. A file can contain only one user batch job.

More than one user batch job can be passed to a batch processor with a single SUBMIT command. The SUBMIT command creates a QMG batch job consisting of one or more user batch jobs. The batch log is a record of the QMG batch job.

See the *RSX-11M/M-PLUS Batch and Queue Operations Manual* for more information.

User File Directory

See directory.

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 *RSX-11M-PLUS Command Language Manual* 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.

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. Disks 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.

word

The word in PDP-11 terminology is a 16-bit unit of data. The word consists of two eight-bit bytes. The CPU and memory are organized around this word length.

Each ASCII character uses a byte. The blocks used in measuring file size are 256 words, or 512 bytes, each.

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

A

Abbreviating commands, 1-9
ABORT command, 1-14, 5-13
Address
 relocatable, 5-5
ADVANCE key (EDT), 2-6
Application, 6-2
Arrow keys (EDT), 2-5

B

BACKSPACE key, 1-6
BACKUP key (EDT), 2-6
Batch log, 4-6
Batch processing, 4-1, 4-5 to 4-6
Binary machine code, 5-4, 5-5
BOTTOM key (EDT), 2-6
BROADCAST command, 3-8

C

Central Processing Unit
 See CPU
CHANGE command (EDT), 2-5
Change mode (EDT)
 See character mode
Character mode (EDT), 2-3 to 2-5
CHAR key (EDT), 2-6
Checkpointing, 6-6
CLI (Command Line Interpreter),
 1-4
COBOL command, 5-10
Command, 1-1
COMMAND key (EDT), 2-8, 2-16
Command Line Interpreter
 See CLI
Compiler, 5-3
Control key
 See CTRL key
COPY command, 3-4
COPY command (EDT), 2-15
CPU (Central Processing Unit),
 6-5, 6-6

Crash, 1-17
CREATE command, 2-1, 3-4
CTRL key
 CTRL/C, 1-3
 CTRL/O, 1-14
 CTRL/R, 1-8
 CTRL/U, 1-8
 CTRL/Z, 1-8, 1-9
Cursor, 1-3
CUT key (EDT), 2-8

D

DCL (DIGITAL Command
 Language), 1-1, 1-10, 3-1,
 5-1
Default, 1-12 to 1-13, 3-9
DEL C key (EDT), 2-7
DELETE/ENTRY command, 3-13
DELETE command, 3-4 to 3-5,
 3-7
DELETE command (EDT), 2-13
DELETE key, 1-6 to 1-8
Deleting text (EDT), 2-7, 2-13
DEL L key (EDT), 2-7
DEL W key (EDT), 2-7
Device, 1-5, 1-11, 3-10, 6-7
 peripheral, 6-1, 6-7
 pseudo, 3-11
DIGITAL Command Language
 See DCL
Directives (ICP), 4-2 to 4-3
Directory, 1-11
DIRECTORY command, 1-11, 3-6

E

Echo, 1-5
EDIT command, 3-6
 /OUTPUT qualifier, 5-8
EDT editor, 2-2
EDTINI.EDT file, 2-2
EOL key (EDT), 2-6
Error message, 1-1

Executive, 5-1, 6-3, 6-5, 6-6
EXIT command (EDT), 2-8

F

File, 2-1, 3-1
contiguous, 5-6
EDTINI.EDT, 2-2
name, 3-1
object, 5-4
protection, 3-4, 6-4
source, 5-5
specification, 1-12 to 1-13
startup command, 2-2
task image, 2-1, 5-6
text, 2-1
type, 2-1

Filespec

See file specification

File specification, 1-12 to 1-13

File type, 2-1

FIND command (EDT), 2-13

FIND key (EDT), 2-7

FIND NEXT key (EDT), 2-8

FORTTRAN command, 5-10

G

GOLD key (EDT), 2-3

H

HELLO command, 1-4

HELP command, 1-10, 1-11, 3-9

HELP command (EDT), 2-5, 2-10

High-level language, 5-10 to 5-11

HOLD/ENTRY command, 3-14

HOLD SCREEN key, 1-14

I

ICP (Indirect Command Processor), 4-1

INCLUDE command (EDT), 2-16

Indirect Command Processor

See ICP

Input, 1-3

INSERT command (EDT), 2-12

Inserting text (EDT), 2-6

INSTALL command, 5-8

K

Keypad (EDT), 2-3

L

LA36 terminal, 1-2

Labels (ICP), 4-3

LINE key (EDT), 2-6

Line mode (EDT), 2-4, 2-9

LINK command, 5-3, 5-6

Logging in, 1-4 to 1-5,

LOGIN command, 1-4 to 1-5

LOGOUT command, 1-16

M

Machine code, binary, 5-4, 5-5

macro, 5-11

MACRO-11 Assembler, 5-4, 5-10

MACRO command, 5-4

MCR (Monitor Console Routine),
1-4, 5-1

Media, 6-8

Memory, 6-5

partition, 6-5

Monitor Console Routine

See MCR

MOVE command (EDT), 2-15

Moving text (EDT), 2-8, 2-15

Multuser system, 6-2

N

NO SCROLL key, 1-14

O

Object module, 5-3, 5-4, 5-9
system library, 5-9

OPEN LINE key (EDT), 2-7

Operating system, 1-1, 5-1

multiuser, 6-2

real-time, 6-2

P

PAGE key (EDT), 2-6

Password, 1-5, 3-10

PASTE key (EDT), 2-8

PIP (Peripheral Interchange Program), 5-1

Priority, 6-4

Privilege

privileged task, 6-4

privileged user, 6-4

Program, 5-2

Prompt, 1-3, 1-9

explicit, 1-4

Pseudo device, 3-11

PURGE command, 3-7

Q

Qualifier (DCL), 3-1

QUIT command (EDT), 2-8, 2-9

R

Range (EDT), 2-10 to 2-12

Real-time system, 6-2

RENAME command, 3-7
RESEQUENCE command (EDT),
2-12, 2-13
RESET key (EDT), 2-8
RETURN key, 1-3
RMDemo, 6-9
RUN command, 3-9, 5-7, 5-11
/TASK_NAME qualifier, 5-11

S

SECT key (EDT), 2-6
SELECT key (EDT), 2-8
SET command, 1-15
SET DEFAULT command, 3-9
SET PASSWORD command,
3-10
SET QUEUE command, 3-13
SET TERMINAL command,
1-15
SHOW command, 1-9
SHOW DEFAULT command,
1-13, 3-10
SHOW DEVICES command,
3-10
SHOW MEMORY command,
6-9
SHOW QUEUE command, 3-13
SHOW TASKS command, 1-16,
5-4
SHOW TERMINAL command,
1-15
SHOW TIME command, 1-9,
3-11
SHOW USERS command, 1-16,
3-11
Significant event, 6-6
Source file, 5-3, 5-4, 5-5
Source language, 5-3
Special symbols (ICP), 4-3
Startup command file, 2-2
STOP/ABORT command, 3-14
SUBMIT command, 4-6
Subroutine, 5-8
SUBSTITUTE command (EDT),
2-15
Substitution mode (ICP), 4-2
Symbol
global, 5-5, 5-9
local, 5-5
Syntax, 1-12

T

TAB key, 5-8
Task, 5-1, 5-2
active, 6-5
dormant, 6-5

Task (cont'd.)
name, 5-11
resident, 6-5
running, 5-2
Task Builder, 5-6 to 5-7
Task image, 5-3
file, 5-6
Terminal, 1-1, 1-5
hardcopy, 1-1
video, 1-1
VT100, 1-1, 1-2
VT220, 1-1, 1-3
Timesharing, 6-3
TOP key (EDT), 2-6
TYPE command, 1-12, 3-8

U

UFD
See directory
UIC, 3-10
UND C key (EDT), 2-7
Undeleting text (EDT), 2-7
UND L key (EDT), 2-7
UND W key (EDT), 2-7
User File Directory
See directory
Utility, 5-1

V

VT100 terminal, 1-1, 1-2
VT220 terminal, 1-1, 1-3

W

Wildcard, 3-1 to 3-3
WORD key (EDT), 2-6
WRITE command (EDT), 2-16

**READER'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 _____

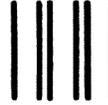
Organization _____

Street _____

City _____ State _____ Zip Code _____
or Country

Do Not Tear - Fold Here and Tape

digital



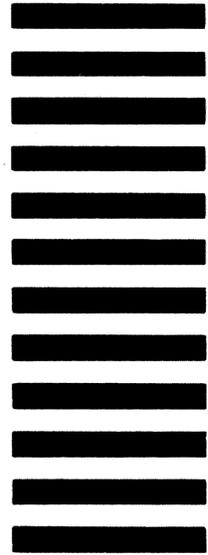
No Postage
Necessary
if Mailed in the
United States

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

Cut Along Dotted Line