Introduction to
RSX–11M and RSX–11M–PLUS

Order No. AA-L763A-TC

RSX-11M Version 4.0
RSX-11M-PLUS Version 2.0

digital equipment corporation · maynard, massachusetts

First Printing, September 1979
Revised, November 1981

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.

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	DECsystem-10	PDT
DECUS	DECSYSTEM-20	RSTS
DIGITAL	DECwriter	RSX
PDP	DIBOL	VMS
UNIBUS	EduSystem	VT
VAX	IAS	digital
DECnet	MASSBUS	

ZK2049

Contents

Chapter 2 How to Create Files

Chapter 3 How to Manage Your Files

Chapter 4 How to Do Work on the System

Figures

Preface

Manual Objectives and Intended Audience

This manual is designed for any new user of RSX-11M or RSX-11M-PLUS, whether familiar with computers or not. RSX-11M and RSX-11M-PLUS are complex systems used for many purposes. This manual will get you started using either system. As you learn more about your system, you will find it provides you with many facilities, both conveniences and necessities.

NOTE

System Managers

See the *Release Notes* for information on setting up a special account for new users who will be using this manual.

Structure of This Document

The manual consists of three parts:

Part I DCL How-To

A warm-up session at a terminal on an RSX-11M or RSX-11M-PLUS system. Part I consists of four chapters:

How to Use Your Terminal — How to log in and log out, how to correct mistakes, and how to issue commands to the system to display system information.

How to Create Files — Including an introduction to EDT, the DEC Editor.

How to Manage Your Files — How to delete them, copy them, rename them, and list them.

How to Do Work on the System — How to prepare and run a program.

Part II Learning the System	An introduction to the system and its operation, including some conveniences for users. Part II consists of two chapters: The System in Operation — How RSX-11M/M-PLUS systems are used. The purpose of the operating system. System resources and controls. Some System Conveniences — Broadcasting messages. The PRINT command. Automatic command entry.
Part III Glossary	A definition of many of the terms used on RSX-11M/M-PLUS systems. All terms introduced in *italics* in the text of this manual are defined in the Glossary.

The DCL How-To consists of examples of terminal use, plus explanations of what the example shows. Users who are experienced with computers will probably find the examples self-explanatory. If not, experienced users can read the text. Inexperienced computer users should read both the examples and the text. In any case, you can reproduce the examples by using files specially prepared for use in DCL How-To.

Your system manager will tell you where these files are located and provide you with information about how to use them. Many systems include a special USER account just for the DCL How-To.

You do not have to be a programmer to use this manual.

Associated Documents

The more you know about RSX-11M or RSX-11M-PLUS, the more use you will get out of it. Your best sources of information on your system are the system manager, in-house documentation, and other, experienced users. You should also be prepared to study the system documentation.

The *RSX-11M/M-PLUS Command Language Manual* describes the DIGITAL Command Language (DCL). This manual should be read after you finish this *Introduction.* If you are a programmer, you should also see the *RSX-11M/M-PLUS Guide to Program Development* for an introduction to the program-development facilities on RSX-11M and RSX-11M-PLUS. The *RSX-11M/M-PLUS Utilities Manual* includes information useful to general users of the system.

Conventions Used in This Document

The following conventions are used in this manual.

Convention	Meaning
n	A number.
CTRL/X	The symbol CTRL/X indicates you must press the key labeled CTRL while you simultaneously press another key, for example, CTRL/C CTRL/O.
RET	A symbol with a 1- to 3-character abbreviation indicates that you press a key on the terminal. RET indicates the RETURN key.
TAB	Indicates the TAB key.
. . .	The vertical ellipsis indicates that not all of the statements in an example or figure are shown.
Red Letters	In examples of commands you enter and system responses, all output lines and prompting characters that the system prints or displays are shown in black letters. All lines that you type are shown in red letters.
Grey Shading	A feature of RSX-11M-PLUS only.
Pink Shading	A feature of RSX-11M only.
	Portions of text that are not shaded describe both operating systems.
UPPERCASE LETTERS	Uppercase letters are used to express command keywords when they appear in text.

Part I
DCL How-To

Chapter 1
How to Use Your Terminal

Here is some help in getting used to operating a terminal on an RSX-11M or RSX-11M-PLUS Operating System. The *operating system* is your primary means of communication with the computer hardware.

The *terminal* is your primary means of communication with the operating system. All system users are terminal users. From your terminal you can issue *commands* that will put the system to work. The system will immediately acknowledge and act upon your commands. Because of this interaction between you and the system, RSX-11M and RSX-11M-PLUS are called *interactive systems.*

Generally, any mistakes you make will result in an *error message.* Often, you will be able to tell what the mistake was from the message. Sometimes you will be able to correct the error. If not, you will find most error messages explained in the system documentation.

At any rate, nothing you are asked to do in this manual can cause the computer system any harm. Some small embarrassment to yourself is the only danger.

You do not need to be a proficient typist. Most commands are short and specific.

Hard Copy Terminals and Video Terminals

You can use either a *hard-copy terminal* or a *video terminal.* A hard-copy terminal has a *print head* and paper in it. It looks like a typewriter. A video terminal has a TV screen in place of the print head and paper.

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

Both terminals have a keyboard similar to that on a typewriter. Some terminals also have a numerical keypad like a calculator keypad on the right-hand side.

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

In particular, three terminals manufactured by DIGITAL are likely to be in use in your *installation.* The LA36 DECwriter II (Figure 1-1) is perhaps the most common hard-copy terminal. The VT52 DECscope (Figure 1-2) is a

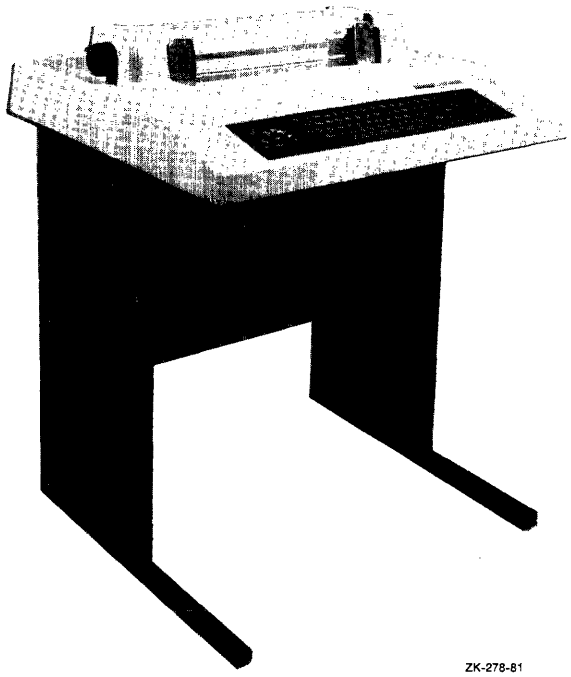

Figure 1-1: LA36 Hard-Copy Terminal

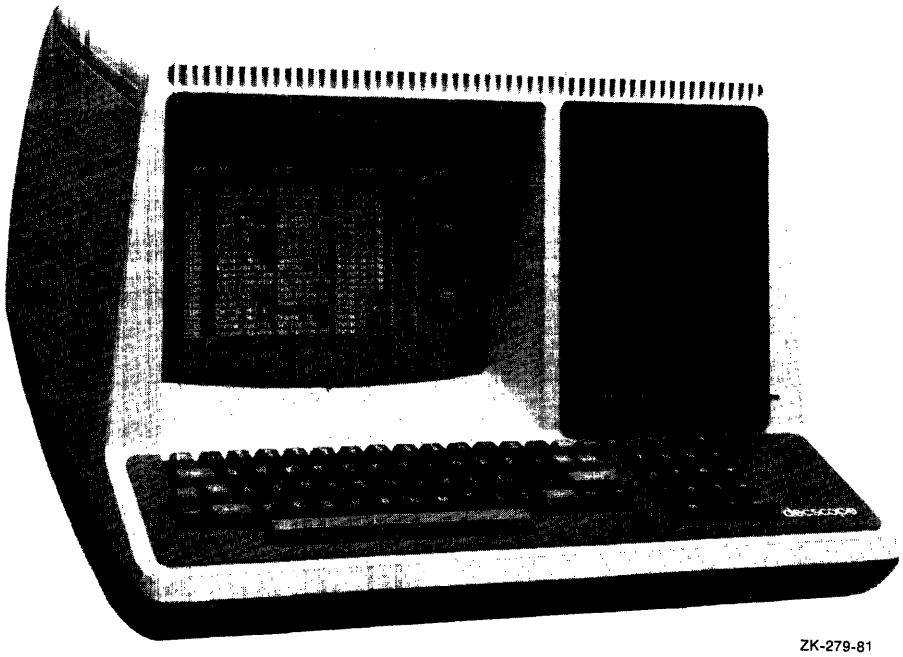

Figure 1-2: VT52 Video Terminal

popular video terminal. The VT100 DECscope (Figure 1–3) has a separate keyboard and is DIGITAL's standard video terminal. Some features of the system are for video terminals only and there are some differences between the VT52 and the VT100 as well. These differences will be noted in this manual as they come up.

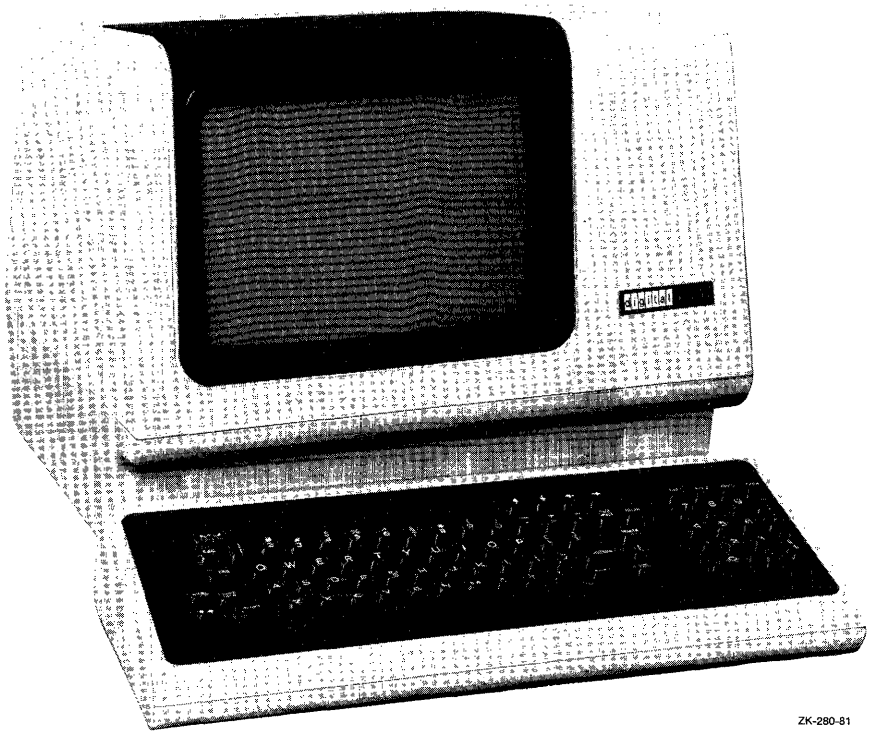

Figure 1–3: VT100 Video Terminal

Before You Start

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

Before You Log In To the System

```
> RET
> RET
> CTRL/C
MCR>
```

- **The RETURN Key Enters Commands and Tests the Terminal.**
- **CTRL/C Captures the Attention of the System.**

As you start, you should see a right-angle bracket (>) or *prompt* on the left margin of the terminal display.

On the right-hand side of the keyboard is a key marked RETURN. Press it a couple of times. You should get a series of right-angle-bracket prompts, each

on a line by itself. These prompts inform you that the system is ready to accept input. (In this manual, the symbol RET means that you press the RETURN key.)

If you are using a video terminal, a blinking indicator, called the *cursor,* should appear next to the right-angle bracket. 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 hard-copy terminal, the print head will usually point to the right of where the next character you type will appear. On most hard-copy 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 will move back into 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 couple of times. Nothing should happen. Now hold the CTRL key down and press the C key at the same time. You should get a three-letter prompt and a right-angle bracket.

The CTRL key and the C key together are called a "control/C." In this manual, the symbol CTRL/C means that you press the C while holding down the CTRL key.

As you will see, you can use the CTRL key with several letters besides C. In this manual, the symbol CTRL/Z means that you press the CTRL key plus Z, CTRL/O means the CTRL key plus O, and so forth. Some of these combinations appear on the screen (or the paper) as a *circumflex* (^) standing for the CTRL key, plus the letter you typed. Thus, CTRL/Z usually appears as:

```
^Z
```

NOTE

> Not all RSX-11M systems support all control commands. See your system manager for more information. RSX-11M-PLUS systems always support these commands.

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.

Now you can log in.

Logging In

```
MCR>LOGIN (RET)
Account or name: USER (RET)
Password: (RET)
```

or

```
MCR>HELLO (RET)
Account or name: USER (RET)
Password: (RET)
```

- **Logging In Gains You Access to the System.**
- **LOGIN is the DCL Command to Log In.**
- **HELLO is the MCR Command to Log In.**

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

```
MCR>
```

This is called the *explicit prompt*.

MCR stands for the Monitor Console Routine. MCR is a command line interpreter, or *CLI*. A CLI is your means of communicating with the operating system.

There are two CLIs available on RSX-11M and RSX-11M-PLUS, MCR and DCL. MCR is present on all RSX-11M/M-PLUS systems; DCL is optional, but is included on most systems with many terminal users.

DCL is the DIGITAL Command Language. It is used on several DIGITAL operating systems, notably VAX/VMS.

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

NOTE

DCL is used in this manual because DCL 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 SET DEBUG/EXECUTE command. This command causes the MCR translation to appear on your terminal before the command is executed. This option is not recommended for inexperienced computer users.

Your terminal is set to either MCR or DCL when you log in to the terminal. (Before logging in, all terminals are set to MCR.)

Continue the exercise in the example. You can use either the MCR HELLO command or the DCL LOGIN command. They are the same. Press the RETURN key to enter the command.

The system will ask you to identify yourself. Type the name USER (or another name, if you have been given special instructions) and press the RETURN key.

The system will ask you for your password. Type the password you were given, but do not press the RETURN key.

Until now, everything you have typed has appeared on the terminal, but the password does not. Passwords are supposed to be secret, to keep unauthorized users off the system. Passwords do not show up on your terminal for this reason.

This illustrates an important point about terminals. Each terminal is really two *devices* in one package. The keyboard is an input device for sending messages to the operating system. The print head or screen is an output device for receiving messages from the operating system.

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

What you seemed to type on the terminal was really an *echo* from the system, confirming that you typed what you thought you typed. For the password, however, the echo is overridden for security purposes.

Occasionally, when the system is busy, you may notice that the echo takes a little longer than usual.

Now enter the password by pressing the RETURN key. You use the RETURN key to enter all commands to the system.

Login Messages from the System

During the *login* procedure, the system checks your identification and password to make sure you should be allowed on the system. If you are identified, then messages from the system (like the messages in the following example) are displayed, and finally your terminal is made available to you, as indicated by the *implicit prompt.*

```
>LOGIN (RET)
Account or name: USER (RET)
Password: (RET)

   RSX-11M V04 BL32 Multi-user [1,54] System
GOOD MORNING
08-SEP-80 08:45 Logged on terminal TT10:

**SYSTEM WILL BE DOWN TONIGHT FROM 21:00 TO 24:00**
                **DISK-TO-DISK BACKUP**

PLEASE PURGE YOUR FILES!!!

June 19, 1980

There are now two VT100s and an LA120 available in
the machine room. Try them out, but do not remove
the manuals from the room.
>@LOGIN.CMD

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
!                                                                      !
!     Hello.                                                           !
!                                                                      !
!      This is the USER account for RSX-11M and                        !
!      RSX-11M-PLUS.                                                   !
!                                                                      !
!      Most of the files in this directory are                         !
!      used in the terminal warm-up session in                         !
!      the Introduction to RSX-11M and                                 !
!      RSX-11M-PLUS.                                                   !
!                                                                      !
!                                                                      !
!      Some files include examples of operating                        !
!      system features and conveniences. See                           !
!      WHATSHERE.TXT for more information. Other                       !
!      files may be deleted from this directory                        !
!      at any time.                                                    !
!                                                                      !
!                                                                      !
!      Take it easy.                                                   !
!                                                                      !
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>


>
```

After everything has happened that is going to happen, the terminal returns the angle-bracket prompt, signifying that it is ready for more input. The text printed on your terminal is probably different.

If you do not get a prompt, type CTRL/C or press the RETURN key.

Setting the Command Line Interpreter

```
> CTRL/C
DCL>
```

or

```
> CTRL/C
MCR>SET /DCL=TI: RET
> CTRL/C
DCL>
```

- **CTRL/C Identifies the Command Line Interpreter.**

Now type another CTRL/C. You should get an explicit prompt for DCL, like this:

```
DCL>
```

The explicit DCL prompt indicates that the terminal is set to DCL.

If you should get the explicit MCR prompt, issue the following command:

```
MCR>SET /DCL=TI: RET
```

Don't forget the space before the slash or the colon at the end. Now type another CTRL/C. You should get the explicit DCL prompt.

The explicit prompt is a guarantee that whatever command you issue will go directly to the operating system. This explicit prompt is almost always available to you with CTRL/C regardless of any other activity at your terminal.

Sometimes, you must type CTRL/C more than once to get the prompt, but once the prompt appears, you know you have the attention of the operating system.

Generally, you do not need CTRL/C to get the attention of the operating system. The implicit prompt (angle-bracket) will usually be enough, but with the explicit prompt you can be sure.

Correcting Typing Mistakes

The BACK SPACE key Is Not Used in DCL.

If you make a typing mistake, use the DELETE or RUBOUT key. In this manual, the symbol DEL means press this key.

Beware the BACK SPACE key. This key is not used in DCL. Any command line including a BACK SPACE character will either be rejected or misinterpreted by the system.

The DELETE key always eliminates the character immediately to the left of the next print position. The key has the same effect on either hard-copy or video terminals, but the output sent back to the terminal is slightly different, as shown in the following example.

The most common errors made by terminal users are confusing the zero and the capital O and confusing the number one and the lowercase L or I. You

should type these characters to see how they differ in appearance on your terminal. After you have typed them, delete them.

Lowercase letters are not used in the examples in this manual, but most systems do accept lowercase letters for most functions. You may want to use the keys labeled SHIFT or CAPS LOCK to make your examples look like those in this manual. The CAPS LOCK key usually has no effect on the number and symbol keys.

Correcting Mistakes with a Video Terminal

```
>THIMK DEL DEL NK
```

- **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 will appear in the vacated location. You can continue deleting until you reach the left margin, but you usually 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:

```
THIM
```

After you press DELETE Key again:

```
THI
```

After you enter the corrections:

```
THIN
```

and

```
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 with a Hard-Copy Terminal

```
>THIMK DEL DEL \KM\NK
```

- **The DELETE Key on Hard-Copy Terminals Prints the Deleted Characters.**

The DELETE function on hard-copy 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 signifies 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 and the correct characters will be echoed on your terminal.

This makes 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 will see:

```
$THIMK\K\M\NK
```

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

Verifying Corrections

```
$THIMK\K\M\NK CTRL/R
$THINK
```

- **CTRL/R Retypes a Line With Corrections.**

If you are confused by the reprinting of deleted characters in a line, or, if you're not sure your corrections were made, enter CTRL/R as shown in the example. This causes the system to retype your line, without deletions and with corrections. You can then continue with whatever you were typing.

Note that CTRL/R echoes as follows:

```
^R
```

CTRL/R does nothing but retype the line. It does not enter a command. CTRL/R works only on the current line, before you press the RETURN key. After the 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 hard-copy 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 Lines

If you wish, you can eliminate a line with CTRL/U. The system gives you a fresh line to start on. See the following example.

```
$SHOE TITE^U
$
```

- **CTRL/U Deletes an Entire Line.**

Ending Input

```
More input?^Z
>
```

- **CTRL/Z Indicates End-of-File or End-of-Input.**

CTRL/Z indicates to the system that you have finished supplying input. It is used with many *system tasks,* but for now all you need to know is that CTRL/Z is another command you can try when your terminal appears to be *hanging,* or not accepting new input. Sometimes CTRL/Z must be entered with the RETURN key or appear on a line by itself in order to clear your terminal.

If at any time you wish to cancel a DCL command, type a CTRL/Z in response to any prompt.

Displaying Information on Your Terminal

```
>SHOW (RET)
Function? TIME (RET)
12:37:18 08-JAN-81
```

or

```
>SHOW TIME (RET)
12:37:33 08-JAN-81
```

- **The SHOW Command Displays System Information.**
- **The SHOW TIME Command Displays the Time.**

Type SHOW and enter it. DCL will prompt you for the next portion of the command. DCL prompts you whenever you have not given a complete command. Some commands prompt you more than once, depending on the options you choose.

TIME is one function you can display. Type TIME in response to the Function? prompt.

Now type SHOW TIME on one line.

Both forms of the command — with or without the Function? prompt — return the same information. The Function? prompt identifies the next command element that DCL is expecting. As you will see, you can use SHOW to display a variety of system information. The Function? prompt can have more than a dozen responses.

These prompts are most useful when you are learning the proper format for a command. As you see in the two forms of the example, DCL prompts you only when you omit a necessary part of a command. Once you have learned a command format, you probably will not need the prompts.

Shortening Commands

```
>SH TIM RET
12:42:19 08-JAN-81
>S T RET
DCL -- Function not unique
>
```

- **DCL Does Not Require the Full Command.**

Now try dropping letters from the command. You will find that SH TIM is sufficient to return the time.

The full command, such as SHOW TIME, identifies the action of the command. For everyday convenience, however, DCL accepts the minimum number of characters needed to distinguish one command from all the others. All DCL commands work this same way.

In the case of SHOW, SH is enough. For some commands, a single character will do. In any case, three letters will usually be enough and four letters will always be enough. You can experiment with each command as you learn it, shortening the command until you get an error message like the one shown in the example.

If, however, your command word contains an error, even in the last character, DCL rejects the command.

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

Help from the System

```
>HELP SHOW TIME RET

SHOW DAYTIME
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.

>HELP TYPE RET

TYPE filespec[s]

The TYPE command displays the contents of text files on your
terminal.
```

- **The HELP Command Displays Information about the System and its Functions.**

Type the command HELP SHOW TIME and enter it. Text explaining the SHOW TIME command appears on your screen.

Now type the command HELP TYPE and enter it. Text explaining the TYPE command appears on your screen. Most RSX-11M/M-PLUS systems include the HELP command. As you go through this manual, you should use the HELP command for each new command that you learn.

A Directory of Your Files

```
>DIRECTORY (RET)


Directory DB0:[200,1]
28-MAR-80 14:33

A.A;1                          1.        12-FEB-80 13:16
AZ.CMD;1                       1.        13-MAR-80 15:38
COPY.CMD;2                     1.        16-FEB-80 12:27
EDT.CMD;5                      1.        28-MAR-80 13:10
LOGIN.CMD;4                    1.        20-MAR-80 13:20
HIYA.MAC;1                     5.        27-MAR-80 10:42
LOGIN.TXT;3                    1.        27-MAR-80 10:19
ERROR.TSK;1                    4.     C  16-FEB-80 12:30
SEVERE.TSK;1                   4.     C  16-FEB-80 12:30
SUCCESS.TSK;1                  4.     C  16-FEB-80 12:29
WARNING.TSK;1                  4.     C  16-FEB-80 12:29
FLY.TXT;3                      1.        15-FEB-80 13:46
FLY.TXT;2                      1.        14-FEB-80 08:09
FLY.TXT;1                      1.        15-FEB-80 13:52
STARS.MAC;1                    2.        26-MAR-80 12:15

Total of 32./125. blocks in 15. files
```

- **The DIRECTORY Command Displays Information about Stored Files.**

Type DIRECTORY and enter it. The example shows the format, but your system probably has a different list of files.

The output from this command is a *directory* of all files stored on a particular *mass-storage device* under a particular *UFD,* or User File Directory. The device and the UFD are given at the head of the directory listing. In the example, the mass-storage device is named DB0:. The *account* has the UFD of [200,1]. The USER account may not have the same number on all RSX-11M/M-PLUS systems.

The User Identification Code — The same number — [200,1] — is also the *UIC,* or User Identification Code, of the account. You can log in to the account using either the name USER or the UIC. Four different forms of the UIC — as well as the account name — are acceptable responses when the HELLO or LOGIN command asks you to identify yourself.

Here they are:

```
[200,1]
[200/1]
200,1
200/1
```

If you use the slash (/) instead of the comma (,), you suppress all or most of the login messages that the system sends you. Some messages cannot be suppressed. Generally, you should display the messages the first time you log in each day. You probably will not want to see them every time you log in.

Each time you log in, use a different form of the UIC or name until you are accustomed to the different effects.

You may find that UIC and UFD seem to be used interchangeably. They often are, but the UIC identifies the user and the UFD identifies the directory where the user's files are stored. As you will see, users can go from one UFD to another, but they always keep the same UIC once they are logged in.

You can use the UIC to log in. The UFD is where the files are. You usually log in where your files are, so the numbers are usually the same.

Device Names — All devices have the same kind of name, with the format ddnn:, where dd is a two-letter identification of the device type and nn is a one- to three-digit number identifying the specific device of a particular type. The colon (:) is a part of the device name.

There are many kinds of devices besides mass-storage devices. *Line printers* and terminals are both devices, for instance. At this point, all you need to know is that files are kept in UFDs on mass-storage devices.

File Specifications

Each file stored on an RSX-11M or RSX-11M-PLUS system has a unique identification, or specification, called a *filespec.* The device name and UFD are important parts of the filespec.

At the head of your directory listing, you will see the device name and the UFD where the files are stored. Within the listing, each file is identified by a file name and file type, separated by a period, and a version number preceded by a semicolon. These five elements fully distinguish one file from all others on the system.

A complete filespec must be entered in just the following manner, without spaces, and with all punctuation marks included.

```
DB0:[200,1]FLY.TXT;3
```

- **A Complete Filespec Consists of a Device Name, a UFD Number, a File Name, a File Type, and a Version Number.**

The glossary includes the *syntax* rules for all *fields* of the filespec. As you go through the exercises in this manual, you will see that each field of the filespec has a part to play in the smooth functioning of the system.

Displaying Files on Your Terminal

```
>TYPE (RET)
File(s)? DB0:[200,1]FLY.TXT;3 (RET)
Time flies like an arrow.
Space flies like a bow.
Fruit flies like a banana.
```

- **The TYPE Command Displays Selected Files On Your Terminal.**

Type TYPE and enter it.

Enter the filespec as shown in response to the File(s)? prompt, using the device number that you were given in place of DB0: if necessary.

A short file should be printed on your terminal.

Now type the TYPE command and the filespec on one line.

Now try dropping various elements of the filespec.

As you will see, you do not always have to include the full filespec to specify a given file. This is because some parts of the filespec are included by *default* if you do not specify them.

Defaults in Filespecs

```
>SHOW (RET)
Function? DEFAULT (RET)
   DB0:[200,1] TT22:
```

- **The SHOW DEFAULT Command Displays the Default Settings of a Terminal.**
- **The Version Number Defaults to the Highest Numbered Version.**

The default device and UFD are automatically included in every filespec you supply if you do not otherwise specify them. This means the system will look no further than the current device and UFD unless you require it. There may be many files on the system called FLY.TXT;3, but there can be no more than one in a given UFD on a given device. If you want to see files with the same name but located in a different UFD or on a different device, include the UFD and device name in the filespec.

One important default does not show. If you do not supply a version number, the system will usually default to the highest numbered version of a file. Some commands require a version number, however.

Type and enter:

```
>TYPE FLY.TXT (RET)
```

The file FLY.TXT;3 is displayed.

Now type and enter:

```
>TYPE FLY.TXT;1 (RET)
```

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

The use of the word "default" may be slightly confusing. In general, the term "by 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/M-PLUS systems, if you don't supply a value, then the system will supply a value of its own, for lack of competition.

If you want to see a copy of a file from another device and UFD or both, simply include the device name and UFD in the filespec and the defaults will be overridden.

Type and enter:

```
>TYPE LB0:[1,2]LOGIN.TXT (RET)
```

You should get most of the same text that is printed on your terminal automatically when you log in.

There are many kinds of defaults used on RSX-11M/M-PLUS systems. So far, you have learned only filespec defaults, but there are many others. The system documentation will tell you what defaults will be applied in various instances.

Defaults are designed for your convenience, but you should keep in mind that you may be specifying several defaults with a single command. Usually, defaults are set to produce the most commonly used form of the command with the least typing.

Controlling Output to Your Terminal

- **CTRL/S and CTRL/Q Delay Output to Your Terminal.**
- **CTRL/O Skips Over Output to Your Terminal.**

Type DIRECTORY and enter it. Depending on the installation, the directory for the USER account may be long or short. Whatever the size, it takes some time to print the directory on your terminal.

Often, particularly on video terminals, the output from a command may *scroll* past too fast for you to read. On hard-copy terminals, the action of the print head may distract you from reading what is being printed.

CTRL/S and CTRL/Q

Type DIRECTORY and enter it, followed immediately by CTRL/S. The output display from the DIRECTORY command should stop.

Type CTRL/Q. The output display from the DIRECTORY command takes up from exactly where the CTRL/S stopped it.

You are not missing any output when CTRL/S is in effect. The output is being saved for you, to be passed along after you cancel the CTRL/S with a CTRL/Q.

The VT52 and VT100 DECscope terminals offer another means of controlling or delaying output to the terminal.

HOLD_SCREEN on a VT52

If your terminal is a VT52, type and enter the following command:

```
>SET TERMINAL/HOLD_SCREEN
```

The screen of the VT52 goes blank and the implicit prompt appears in the upper left-hand corner of the screen. Enter one or more DIRECTORY com-

mands until the screen is full. Notice that the output does not scroll away. Now press the SCROLL key on the VT52. One additional line of output appears. Next, press the SHIFT/SCROLL combination and an entire screenful of new output appears. Another SHIFT/SCROLL combination brings up another screenful of output and so forth. You can cancel HOLD_SCREEN with CTRL/C and the RETURN key or with the following command:

```
>SET TERMINAL/NOHOLD_SCREEN
```

NO SCROLL Key on a VT100

If your terminal is a VT100, you can use the NO SCROLL key. Enter a DIRECTORY command. Press the NO SCROLL key. The output stops. Press the NO SCROLL key again. The output starts again.

CAUTION

If your terminal appears to be hanging (not accepting new input), you may have forgotten that you are controlling output through CTRL/S, the NO SCROLL key, or the SET TERMINAL/HOLD_SCREEN command.

CTRL/O

Now type DIRECTORY and enter it, followed immediately by CTRL/O. The output display from the DIRECTORY command should stop. Type another CTRL/O. The output display should start again, but not at the same place.

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

Say you are checking to see if a particular file is in a directory. You issue the DIRECTORY command and see the file you were looking for close to the top of the listing. You have no interest in the other files on the list. Type CTRL/O and the excess output will be skipped over. If you do not type another CTRL/O, the right-angle bracket prompt will appear when the system has finished with the output from the DIRECTORY command.

The system can run through this output much faster when it does not have to print it on your terminal. If you should type another CTRL/O before the system prompt is returned, the system will start printing output on your terminal at the point it had reached when you typed the second CTRL/O.

NOTE

Remember, not all control commands are supported on all RSX-11M systems. These commands are always supported on RSX-11M-PLUS systems.

Setting and Showing Terminal Characteristics

```
>SET (RET)
Function? TERMINAL (RET)
Terminal Attribute? LOWER
>SHOW TERMINAL
TT10:  [303,5]          [303,5]
       CLI    = DLC     BUF    = 132.    HFILL = 0      SPEED=(9600:9600)
       LINES  = 24.     TERM   = VT100   OWNER = NONE    BRO      NOABAUD
       LOWER    NOPRIV  NOHOLD  NOSLAVE  NOESC   CRT     HFF      NOREMOTE
       ECHO     NOVFILL NOHHT   NOFDX    WRAP    NORPA   NOEBC    TYPEAHEAD
>SET TERMINAL/UPPER
>SHOW TERMINAL
TT10:  [303,5]          [303,5]
       CLI    = DCL     BUF    = 132.    HFILL = 0      SPEED=(9600:9600)
       LINES  = 24.     TERM   = VT100   OWNER = NONE    BRO      NOABAUD
       NOLOWER  NOPRIV   NOHOLD  NOSLAVE  NOESC   CRT     HFF      NOREMOTE
       ECHO     NOVFILL NOHHT   NOFDX    WRAP    NORPA   NOEBC    TYPEAHEAD
>
```

- **The SET Command Establishes Certain System Characteristics.**
- **The SET TERMINAL Command Establishes Certain Terminal Characteristics.**
- **SET TERMINAL/LOWERCASE Causes Terminals to Leave Lowercase Input Unchanged.**
- **The SHOW TERMINAL Command Displays Characteristics of Your Terminal and Other Terminals on the System.**
- **SET TERMINAL/UPPERCASE Causes Terminals to Convert Lowercase Input to Uppercase.**

Type and enter the following command:

```
>SET TERMINAL/UPPERCASE (RET)
```

Remember, you can use either the prompting form shown in the example or the single-line form shown here.

Now type and enter the following command:

```
>SHOW TERMINAL (RET)
```

The system displays all the attributes set for your terminal. Among these attributes you will see LOWER. This means the terminal does not change lowercase characters to uppercase before sending them out to the system. If your terminal is not set LOWER, any lowercase character you type will be echoed in uppercase.

You will also see many other attributes of your terminal listed by the SHOW TERMINAL command. Not all of them have obvious meanings, but they include your terminal number and the width (BUF) and length (LINES) of your terminal's display area. The other terminal attributes are explained in the *RSX-11M/M-PLUS Command Language Manual* and the HELP files.

Now type and enter the following command:

```
>SET TERMINAL/UPPERCASE
```

Your terminal is now set to translate any lowercase characters you type into uppercase before transmitting them to the system. Any lowercase character you type will be echoed in uppercase. You might have to do this to type commands or data to some program on your system that does not recognize lowercase characters. Try typing a command in lowercase letters to see the effect.

Now type and enter the SHOW TERMINAL command again. Where the display listed LOWER, you will now find NOLOWER. You may want to set your terminal back to LOWER before continuing with this terminal session.

Displaying System Information

Now for some variations on the SHOW command.

```
>SHOW (RET)
Function? USERS (RET)
TT1: [305,306] [305,306]     10-JAN-81 11:44:14     0        J. RILEY
 .
 .
 .
TT60: [1,2] [307,3]          10-JAN-81 13:40:30     2        Y. HICKS
```

- **The SHOW USERS Command Displays a List of Logged-In Users.**

Other Users on the System

Type SHOW USERS and enter it. The display shown differs from system to system.

On RSX-11M systems, the display includes only the terminal number and login UIC.

On RSX-11M-PLUS systems without Resource Accounting, the display includes only the terminal number, the login UIC, and the default UFD.

On RSX-11M-PLUS systems with Resource Accounting, the display includes the terminal number, the default UFD, the login UIC, the time each user logged in, the number of tasks active at each terminal, and each user's name.

Regardless of the system, someplace in this display you will find yourself.

Notice that the terminal number is in the same form as other device names, that is, ddnn:. The TT identifies the type of device, in this case, a terminal. The number is the number of the terminal, and the colon marks the end of the device name.

Tasks on the System

```
>SHOW RET
Function? TASKS RET
Active or installed? ACTIVE RET
DCL...  (TT10:)
SHOT10  (TT10:)
> RET
> RET
>SHOW TASKS/ACTIVE/ALL RET
...LDR  (C00:)
RMDEMO  (TT62:)
MCR...  (TT21:)
DCL...  (TT21:)
SHOT10  (TT10:)
F11ACP  (C00:)
.       .
.       .
.       .
EDIT57  (TT57:)
SRDT20  (TT20:)
TT3     (TT3:)
DSCT60  (TT60:)
TT35    (TT35:)
>
```

- **SHOW TASKS/ACTIVE Displays a List of Tasks Active at Your Terminal.**
- **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.

Tasks initiated from terminals are generally identified by the first three letters of the command that initiated them. If the command was a RUN command, the task is named after the terminal that initiated it.

On RSX-11M-PLUS systems, the name of the terminal that initiated the task appears in parentheses next to the task name.

On large RSX-11M-PLUS systems, tasks run from terminals with numbers greater than 100 have names that include a letter. The terminal number always appears in parentheses next to the task name. See the *RSX-11M/M-PLUS Command Language Manual* for more information on task naming.

As you can see in the example, the system permits you to have more than one task active at a time.

Now type SHOW TASKS/ACTIVE/ALL and enter it. The output will be in the same format you got earlier, but much longer, since it shows the tasks currently active on the whole system.

Many tasks can be active at the same time all over the system. The system can do this because different tasks use different parts of the system's resources at different times. One task may be using the line printer while another is using a mass-storage device. It is the job of the operating system to keep track of which tasks need which system resources and to make the resources available in an efficient manner. This sharing of resources is controlled by the *Executive*. In fact, RSX stands for "resource-sharing executive."

Stopping a Command

```
>DIRECTORY (RET)

Directory DB0:[200,1]
28-MAR-80 14:33

A.A;1                1.          12-FEB-80 13:16
AZ.CMD;5             1.          13-MAR-80 15:38
COPY.CMD;2           1.          16-FEB-80 12:27
EDT.CMD;5            1.          28-MAR-80 13:10
(CTRL/C)
DCL>ABORT DIRECTORY (RET)
HIYA.MAC;1           5.          27-MAR-79 10:42
13:14:13  Task "DIRT10" terminated
          aborted via directive or CLI
```

- **The ABORT Command Halts Execution of a Command or Task Resulting from It.**

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

Now, in response to the DCL 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.

Logging Out

```
>LOGOUT (RET)
Connect time:    3          minutes
CPU time used:   3          seconds
Task Total:      34
Have a Good Afternoon
22-OCT-81 14:48   TT64:  logged off KERMIT
>
```

- **The LOGOUT Command Logs You Off the System.**

When you are through using the system, you must log yourself off, using the LOGOUT command.

On most RSX-11M-PLUS systems, the LOGOUT command also reports on your use of the system. Your system may not give you this information, but if

it does, 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 CPU to execute instructions.

For most of the time that you were logged in, the system was waiting for input from you. This is another reason why it is possible for many users to 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, some other user's needs can be serviced.

In addition to logging you off, the LOGOUT command also cleans up behind you, aborting any active tasks and returning resources to the system. If you should ever feel that you have lost control of your terminal, CTRL/C and the LOGOUT command will restore tranquility. Then, if you wish, you can log back in and try again.

One last word about problems with the system: *crash*.

Sometimes nothing seems to work, not even CTRL/C or the LOGOUT command. On these occasions, the system may have crashed. It was probably not your fault. A crash is the system's response to an unstable condition, usually caused by a *privileged* user or privileged task. If the system should crash, it will probably be brought back in a few minutes. In any case, nothing can be done from your terminal until messages inform you that the system is up and running again.

Summary

With this chapter, you have learned the most common ways of eliminating confusion from your terminal — CTRL/Z, CTRL/C, the ABORT command, and the LOGOUT command.

You now know enough about terminal operations to begin using some of the more complex, and useful, facilities of the system.

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 assist you.

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

Chapter 2
How To Create Files

A *file* is a collection of data in a form significant to a user.

The definition covers a lot of territory. Files in RSX-11M/M-PLUS systems can be of many types. *Text files,* like FLY.TXT, are in a readable format. Other files, such as *task image files,* are in an unreadable format.

The file type — the three-letter identification at the end of the filespec — usually gives a clue as to the contents of a file.

Most programs that run on RSX-11M/M-PLUS systems start out as a particular kind of text file called a *source file.* Source files are written in a *supported* computer language. Every RSX-11M/M-PLUS system supports the use of one or more languages. MACRO-11 source files are usually of the .MAC file type; FORTRAN-IV source files usually have the file type .FTN.

This chapter teaches you how to create text files on an RSX-11M/M-PLUS system.

You'll 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 create and edit a text file. Follow the examples closely to see how everything works. Later, you can experiment with these and other commands to gain experience.

Review the previous chapter on terminal operations. If you are not logged in already, log in as you did before. Make sure your terminal is set to DCL. Type a CTRL/C to be certain.

Creating a File

```
>SET TERMINAL/LOWERCASE (RET)
>CREATE (RET)
File(s)? DOCTOR.FEL (RET)
I do not like you, Doctor Fell, (RET)
^Z
>TYPE DOCTOR.FEL (RET)
I do not like you, Doctor Fell,
>
```

- **The CREATE Command Creates Files.**

Type SET TERMINAL/LOWERCASE and enter it. This permits you to type the examples exactly as shown.

Type CREATE and enter it. Supply the file name DOCTOR and file type .FEL in response to the File(s)? prompt and enter it.

The prompt does not appear. Type the line of text shown in the example. The RETURN key works like a typewriter carriage return only. When you have finished typing the line, enter CTRL/Z on a line by itself. The implicit prompt returns.

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

The CTRL/Z indicated the end of input and thus took you "out" of the CREATE task and returned you to DCL, which is also called *monitor level.*

You have now created a file called DOCTOR.FEL. It is automatically numbered version 1 and placed in the default UFD of [200,1]. Thus, the file in the example has the filespec [200,1]DOCTOR.FEL;1.

Note that even though you have created a text file, the file type is not .TXT, but .FEL. RSX-11M/M-PLUS systems permit you to give whatever name and file type you like to your files. On the other hand, RSX-11M and RSX-11M-PLUS also provide default file types for various purposes. If you use these default file types, you will not have to supply file types for many system tasks. More on this later.

The CREATE command is handy for making notes or writing short, ad-hoc programs or tests, but the facilities for editing files made with this command are primitive, consisting only of the DELETE key, CTRL/R, and CTRL/U.

For fancier *functionality,* RSX-11M/M-PLUS systems provide *editors,* which are system tasks designed to make text preparation easier.

The EDT Editor

```
>EDIT/EDT (RET)
File? >DOCTOR.FEL (RET)
     1      I do not like you, Doctor Fell.
*(RET)
[EOB]
*
```

- **The EDIT Command Invokes an Editor.**
- **The EDIT/EDT Command Invokes EDT, the DEC Editor**
- **The Prompt for EDT Line Mode is an Asterisk (*).**
- **EDT Automatically Numbers Lines.**
- **EDT Locates Lines by Numbers.**

Type EDIT/EDT and enter it. You should get a three-letter prompt and right-angle bracket:

```
EDT>
```

Many other system utilities prompt in this way. When you see a three-letter prompt, you know that any commands you enter will be directed to that utility, rather than to DCL.

EDIT/EDT is a new form of command for you. There is more than one editor supported on RSX-11M/M-PLUS systems. The EDIT command can invoke any of these editors through the use of *qualifiers* to the EDIT command. *Command qualifiers* are typed with the slash (/) and the qualifier immediately following the command, with no intervening spaces. You will recall that when you do not use the DCL prompts, the intervening space marks a new command field. Thus, when you use a command qualifier, you must use the slash and you must not leave a space between the command and the qualifier.

EDT, the DEC Editor, is used on a number of DIGITAL operating systems. Other editors available on RSX-11M/M-PLUS systems include EDI, the Line Text Editor, and SLP, the Source Language Input Program, a special editor used mainly for program maintenance. Your system may include other editors not supplied by DIGITAL.

Each editor has its advantages and disadvantages. EDT is especially versatile if you have a video terminal, but it works on hard-copy terminals as well. This section begins with instructions for using EDT on any terminal. EDT's *change mode,* which works best on a video terminal, is introduced at the end of the chapter.

In response to the EDT prompt, type DOCTOR.FEL (the filespec for the file you created earlier) and enter it. You should get an asterisk (*). This is the

EDT prompt. It signifies that you are "in" EDT and that EDT is ready to accept your commands. This is called EDT's *line mode.*

Press the RETURN key. EDT will print the first line of the file. It has the *line number* 1. EDT automatically numbers lines. Line numbers are used to locate and manipulate the text in your file.

Now press the RETURN key again. You should get the symbol [EOB] and another prompt. EOB means End-of-Buffer. In EDT, a *buffer* is a work space used in editing files. In this case, the buffer contains one line of text. The [EOB] means there are no more text lines. The buffer can, however, contain an unlimited number of text lines. For this practice session, you will be doing all your work in one buffer, called MAIN. EDT permits you to create further buffers if you need them.

NOTE

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

Entering Text into Files

```
*TYPE 1 RET
1                   I do not like you, Doctor Fell,
*T 1 RET
1                   I do not like you, Doctor Fell,
* RET
[EOB]
*INSERT RET
                    The reason why I cannot tell, RET
                    CTRL/Z
*T 1 RET
    1               I do not like you, Doctor Fell,
* RET
    2               The reason why I cannot tell,
* RET
[EOB]
*
```

- **The TYPE Command Prints Specified Lines on Your Terminal.**
- **T is the Abbreviation for the TYPE Command.**
- **The INSERT Command is Used to Add New Text to Files.**
- **CTRL/Z On a Line by Itself terminates the INSERT Command.**

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

When you created DOCTOR. FEL, you supplied no line numbers, but when you brought the file into EDT, line numbers were automatically assigned. These line numbers are used only by EDT. They are not a part of your file. When you return to DCL from EDT, you leave the line numbers behind.

Type T 1 and enter it. Line 1 is printed again.

Press the RETURN key. EDT prints [EOB] and returns a prompt.

Now type INSERT in response to the prompt and enter it. This command tells EDT that you want to insert text. There may be a brief delay before the cursor or print head moves down one line indicating that you can now insert text. No prompt appears. This is similar to what happened with the CREATE command. With the CREATE command, the cursor was at the left margin on your terminal, while with EDT, the cursor is indented.

Now type the additional line of text shown in the example. The RETURN key works like a typewriter carriage return only. When you have finished typing the line, enter a CTRL/Z on a line by itself. The EDT prompt should appear. Again, this is similar to ending input for the CREATE task.

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

Now press the RETURN key. EDT prints line 2 automatically and returns the prompt. You inserted line 2 just ahead of the end of the buffer.

Displaying Lines of Text in the File

```
*T WHOLE (RET)
    1             I do not like you, Doctor Fell,
    2             The reason why I cannot tell,
[EOB]
*T BEGIN (RET)
    1             I do not like you, Doctor Fell,
*T END (RET)
[EOB]
*T LAST (RET)
    2             The reason why I cannot tell,
```

- **Line Mode Can Display All or Part of a Buffer.**
- **The Lines to be Displayed are Selected by Range Specifications.**
- **Range Specifications Can be Line Numbers or Descriptive Words.**

Type TYPE WHOLE and enter it. The entire buffer should be printed on your terminal, with line numbers.

Now type T BEGIN and enter it. EDT should return to the beginning of the buffer and print the first line. 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 T END and enter it. The line printer moves to the end of the buffer and displays [EOB].

Now type in T LAST and enter it. The line pointer moves "up" one line and displays the line.

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

```
*HELP 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 or range specifications:

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

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

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

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

Additional information available:
ALL             AND             BEGIN           BEFORE          BUFFER
DOT             END             FOR             LAST            MINUS
NUMBER          ORIGINAL        PLUS            REST            SELECT
STRING          THRU            WHOLE
*HELP RANGE MINUS
The minus sign in ranges selects a single line which 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 which 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.
```

- **The HELP Command Provides Help Without Leaving EDT**

Type HELP RANGE and enter it. After a pause, EDT displays text explaining the different ways of expressing a range. What you have been doing is combining range expressions 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.

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 such EDT concepts as RANGE, WHOLE, and so forth.

The help text is usually quite complete. 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.

Simpler Ways of Displaying Lines of Text

```
*T 1 (RET)
     1              I do not like you, Doctor Fell,
*1 (RET)
     1              I do not like you, Doctor Fell,
* (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,
*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 you, Doctor Fell,
*T WH (RET)
     1              I do not like you, Doctor Fell,
     2              The reason why I cannot tell,
[EOB]
*WH (RET)
Unrecognized command
*
```

- **The TYPE Command Moves the Line Pointer to the Beginning of a Range.**
- **You Do Not Have to Type the TYPE Command if the Range Expression Begins with a Line Number.**
- **There are Many Ways of Expressing Ranges.**

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

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

Press the RETURN key by itself. Now 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.

This time, type T 1 THRU 2. Both lines are printed.

Type 1 THRU 2 and enter it. This is the equivalent of the previous command. EDT does not require an explicit TYPE command when the range begins with a line number.

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.

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

Finally, type WH and enter it. You should get an error message. Range expressions that do not start with a line number must be preceded by a command. Since you entered an illegal command, nothing happens except the error message. Your text is unaffected.

Renumbering Text Lines

```
*1 RET
    1             I do not like you, Doctor Fell,
*I RET
                  But this I know and know full well, RET
                  CTRL/Z
*T WH RET
    0,1           But this I know and know full well,
    1             I do not like you, Doctor Fell,
    2             The reason why I cannot tell,
[EOB]
*RESEQUENCE RET
3 lines resequenced
*T WH
    1             But this I know and know full well,
    2             I do not like you, Doctor Fell,
    3             The reason why I cannot tell,
[EOB]
*
```

- **I is the Abbreviation for the INSERT Command.**
- **The INSERT Command Inserts Text Ahead of the Line Pointer.**
- **The RESEQUENCE Command Renumbers Lines.**

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

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

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.

You will remember that when you inserted line 2, the line pointer was pointing to the end of the buffer. The new text at that time was inserted ahead of the end of the buffer.

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

Type RESEQUENCE and enter it. Now type T WH and enter it. The lines have been renumbered in increments of 1.

Moving and Copying Text Within the File

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

- **The MOVE Command Moves Text.**
- **RES is the Abbreviation of the RESEQUENCE Command.**
- **M is the Abbreviation for the MOVE Command.**
- **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 (the EDT abbreviation for the RESEQUENCE command) and enter it.

Print the whole buffer by typing T WH and entering it. The last line you inserted is now in its proper position in the rhyme.

Now type M 2:3 TO 1 and enter it. (M is the EDT abbreviation for the MOVE command.) The command means "Move lines 2 through 3 to just ahead of line 1." EDT informs you that you have moved two lines. Remember that with EDT's line-numbering rules, the range 2:3 could be many more than two lines.

Resequence again.

Type T WH and enter it.

Now type COPY 3 to 1 and enter it. EDT informs you that the line has been copied. Resequence again and print the whole buffer using the T WH command.

The rhyme is now nearly complete.

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

Replacing a Text String

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

- **The SUBSTITUTE Command Replaces Text.**
- **S is the Abbreviation for the SUBSTITUTE Command.**
- **You Can Make Substitutions Throughout a Range.**

Type 4 and enter it to move the line pointer to line 4 and print it. Remember, when typing a numerical range, you can simply enter the range without explicitly using the TYPE or T command.

Line 4 is almost correct, but it ends with a comma instead of a period.

Type SUBSTITUTE/Fell,/Fell. and enter it.

Print the whole buffer. The rhyme should now be complete and correct.

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.*

Since the SUBSTITUTE command makes its substitution on the first matching string it encounters on the current line, you did not command EDT to search for a comma and replace it with a period. If you had, the first comma in the line would have been replaced, not the last. The inclusion of the entire last word makes sure you replace the correct comma.

For this first substitution, EDT operated only on the line the line pointer was pointing to. In other words, the SUBSTITUTE command operated on a one-line range. You can also make substitutions throughout a range.

Type S/ell/umble/WHOLE and enter it. EDT searches the line for an instance of the string "ell" and makes the substitution. The new line is printed and EDT continues searching for the string "ell" throughout the buffer, making the substitution and printing each new line. When it has completed the operation, EDT informs you of the number of substitutions made.

Deleting Lines from Text

```
*1 (RET)
     1              I do not like you, Doctor Fumble,
*S/umble/ell/1 AND 2 (RET)
     1              I do not like you, Doctor Fell,
     2              The reason why I cannot tell,
2 substitutions
*S/umble/ell/REST (RET)
     3              But this I know and know full well,
     4              I do not like you, Doctor Fell.
2 substitutions
*I (RET)
                    All around the mulberry bush (RET)
                    (CTRL/Z)
*T WH
     1              I do not like you, Doctor Fell,
     1.1            All around the mulberry bush,
     2              The reason why I cannot tell,
     3              But this I know and know full well,
     4              I do not like you, Doctor Fell.
*DELETE 1.1 (RET)
1 line deleted
     2              The reason why I cannot tell,
*
```

- **The DELETE Command Removes Lines from a Buffer.**

Move the line pointer to line 1 and print it.

This time enter the SUBSTITUTE command with the range 1 AND 2 and notice the effect.

Now enter the SUBSTITUTE command with the range REST.

The rhyme has been restored to its original form. Now type I and enter it. Insert the additional line as shown and then print the buffer. Even though your substitutions carried you through line 4, the line pointer was still pointing to line 2 and therefore the insert went ahead of line 2 and was given the number 1.1.

Now type DELETE 1.1 and enter it. EDT informs you of the deletion and prints the next line on your terminal.

Searching for Strings of Text

```
*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,
*F BE (RET)
*"" (RET)
     1              I do not like you, Doctor Fell,
*F BE (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,
*+2 (RET)
     3              But this I know and know full well,
*-1 (RET)
     2              The reason why I cannot tell,
*
```

- **The FIND Command Moves the Line Pointer.**
- **You Can Search for a String by Quoting It.**
- **A String Search Moves the Line Pointer Past the String.**
- **You Can Search Again for the Same String by Typing Just the Quotes (" ").**
- **F is the Abbreviation for the FIND Command.**
- **You Can Search for a Multiple Line Range by Quoting from the First and Last Lines**
- **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," 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 is perhaps confusing at this point, but as you use EDT, you will find it less so.

Now type F BE and enter it. F is the abbreviation for the FIND command and BE is the abbreviation for the BEGIN command. The EDT prompt returns, but you see nothing else on your terminal. The line pointer has been moved to the beginning of the buffer again.

Type two quotation marks with nothing between them and enter it. The first line is reprinted. EDT remembers the last string that you searched for and searches for it again without your typing it.

Return to the beginning of the buffer by typing F BE and entering it.

Now type "I do" THRU "Fell." and enter it. The entire rhyme is printed on your terminal. The line pointer has not moved, however, as you can confirm by typing a dot (.) and entering it.

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 commands, such as BEGIN + 1 or LAST -2 or "reason" + 1.

Leaving the Editor

```
*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]
*EXIT (RET)
DB0:[200,1]DOCTOR.FEL;2 4 lines
>TYPE DOCTOR.FEL (RET)
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.
>EDIT/EDT DOCTOR.FEL (RET)
    1           I do not like you, Docto~ Fell,
*T WH
    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]
*D 2 THRU 4
3 lines deleted
*T WH (RET)
    1           I do not like you, Doctor Fell,
[EOB]
*QUIT (RET)
>TYPE DOCTOR.FEL
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 EXIT Command Takes You Out of EDT and Makes a New File.**
- **D is the Abbreviation for the DELETE Command.**
- **The QUIT Command Takes You Out of EDT But Does Not Make a New File.**

Type T WH to print the whole buffer on your terminal. The rhyme is complete.

Now type EXIT and enter it. You will now be leaving EDT. The full name of your file is printed, along with its length. As you see, this is version 2 of DOCTOR.FEL. EDT is informing you that it has created a new output file based on the editing that you did on version 1, the version you created using the CREATE command. By issuing the EXIT command, you directed EDT to create a new version of the file.

The implicit DCL prompt (>) returns, signifying that EDT is no longer active at your terminal. Now type and enter the command TYPE DOCTOR.FEL. Notice that this is the DCL TYPE command rather than the EDT TYPE command. The complete file appears on your terminal.

Return to EDT using the EDIT/EDT command and naming DOCTOR.FEL as the file to be edited. Use T WH to make sure the file is complete.

Type D 2 THRU 4 and enter it. D is the abbreviation for the DELETE command. EDT informs you that three lines have been deleted. Use T WH again to be sure the lines have been deleted. The buffer now contains only one line.

Now type QUIT and enter it. The implicit prompt returns immediately. EDT prints no messages on your terminal. This is because the QUIT command directs the editor not to create a new version of the file.

Type TYPE DOCTOR.FEL and enter it. The full four-line rhyme is printed on your terminal. Deleting the three lines had no effect because you left EDT using the QUIT command rather than the EXIT command. If you had left EDT using the EXIT command, then the one-line version of the file would have been printed on your terminal. The QUIT command is useful if you make a serious mistake in editing. In such a case, you can simply issue the QUIT command and start editing all over again using the same input file as before.

Here is what has happened. When you set out to edit an existing file with EDT, EDT makes a copy of that file (called the input file) for you to work on. If, for any reason, you do not want to keep the results of an editing session, you can issue the QUIT command. EDT then throws away whatever work you have done. The original input file, which was copied for editing, still exists. If you want to keep the results of your editing, you can issue the EXIT command. EDT then creates a new file (called the output file) containing the results of your work. The original input file, which was copied for editing, still exists, but now you have a new output file, consisting of the altered copy of the original file.

EDT has a number of ways of accepting input files and creating output files. The example shows the simplest means of doing so.

Creating Files with EDT

```
>EDIT/EDT NEW.FIL (RET)
Input file does not exist
[EOB]
*I (RET)
                This is the maiden all forlorn who milked the cow with the crump
led horn that kicked the dog that worried the cat that killed the rat that ate
the malt that lay in the house that Jack built. (RET)
        (CTRL/Z)
*RES (RET)
1 line resequenced
*T WH (RET)
                This is the maiden all forlorn who milked the cow with the crump
led horn that kicked the dog that worried the cat that killed the rat that ate
the malt that lay in the house that Jack built.
*EXIT (RET)
DB0:[200,1]NEW.FIL;1 1 line
>TYPE NEW.FIL (RET)
This is the maiden all forlorn who milked the cow with the crumpled horn that
kicked the dog that worried the cat that killed the rat that ate the malt that lay
 in the house that Jack built.
>
```

- **EDT Can Create Files**
- **A Line in EDT Extends from Return to Return**

Now enter EDT giving a new file name and type, one that does not appear in the directory for [200,1].

EDT displays a message stating that the input file you named does not exist. This message means that EDT is creating a file for you. Normally, you will create files using EDT rather than the CREATE command.

EDT also shows [EOB] indicating that the line pointer is pointing to the end of the buffer. EDT always starts out with the line pointer pointing at the first line in the buffer. In this case, the first line of the buffer is the end of the buffer. If you had given the name and type of an existing file, the first line of that file would have been displayed.

Now, issue the INSERT command and insert a line wider than your terminal screen. *Do not use the RETURN key.* EDT considers everything you enter with the RETURN key to be one line. EDT types on a new line for legibility. (Your example may not look exactly the same. Terminal models differ slightly in their handling of this situation.)

Return to line mode using the RETURN key and CTRL/Z. Resequence the lines and print the whole buffer. Notice that only one line is resequenced.

EDT line mode deals with lines wider than your terminal can handle by *wrapping* the lines. The line has only one line number because it is only one line, but it takes more than one screen line to print it.

An EDT line can be up to 255 characters long. If you try to insert a longer line, it will be *truncated.*

Now leave EDT using the EXIT command. Notice the EDT message that says the file is one line long.

When the implicit prompt returns, print the file on your terminal using the DCL TYPE command. The line is still wrapped, but not in the same way. Again, your example may not look exactly the same.

Summary

This has been your introduction to editing using EDT. You have learned to use the most common editing functions:

- Inserting and deleting lines
- Getting help
- Moving around in the text file
- Making local and *global* substitutions
- Using range designations
- Numbering and renumbering lines
- Moving and copying text
- Creating new files and new versions of files.

Most editing software provides some means of performing these functions, but EDT has additional capabilities that have not been touched on here.

The *EDT Editor Manual* has further introductory sessions for some of the advanced features of EDT. These features include:

1. Multiple buffers that permit you to work on smaller blocks of text while building a large file in the main buffer. If, for instance, you have text that must be inserted repeatedly in a file – a subroutine in a program perhaps, you can store that text in an alternate buffer and use the COPY command whenever you need that text.
2. A journaling facility that protects you against losing your files, should the system crash while you are editing, or if you accidentally edit more of your file than you meant.
3. A change mode that permits you to edit not only by line, but by character, word, sentence, or page. Change mode works on both hard-copy and video terminals, but it is particularly suited to video terminals. When editing in change mode on a video terminal, you see a screenful of text at a time and each change you make in your text is immediately shown on the screen. If the video terminal has a numerical keypad, you can use it to enter editing commands.
4. A DEFINE command that enables you to combine a series of EDT line or change mode commands into custom commands. These commands can be automatically defined for you each time you use the editor through special EDT initialization files.

You should explore the EDT help files and the *EDT Editor Manual* for further guidance on using the many capabilities of EDT. Your installation may have special versions of EDT that use custom commands to perform common functions.

Chapter 3
How to Manage Your Files

Almost everything that the operating system does either starts out as a file or ends up as one. DCL provides you with many tools for managing files.

You have done some file management already with the DIRECTORY and TYPE commands. You have also created files using the editor, EDT, and the CREATE command.

In this chapter, you will rename files, copy them, delete them, type them on your terminal, and print them on your system's line printer. You will also learn how to specify a large number of files without a large amount of typing through the use of *wildcards.*

The Directory

```
>DIRECTORY (RET)

Directory DB0:[200,1]
8-JUN-81 10:21

AZ.CMD;4                    2.        23-JAN-80 18:48
LOGIN.CMD;1                 1.        14-DEC-80 10:37
FLY.TXT;1                   1.        09-SEP-80 15:15
FLY.TXT;2                   2.        08-DEC-80 15:05
A.A;1                       1.        25-SEP-80 09:25
     .
     .
     .
NEW.FIL;1                   1.        29-FEB-80 00:00
CLEAN.CMD;5                 1.        13-JUL-80 15:03
FLY.TXT;3                   1.        15-MAR-80 08:05

Total of 234./250. blocks in 29. files

>
```

- **The DIRECTORY Command Displays Information About Stored Files.**

Type DIRECTORY and enter it. (You must have left EDT.)

The directory displayed on your terminal will probably not be the same as that shown here, but it should contain many of the same files.

The directory shown is in the standard format. The heading gives you the name of the mass-storage device where your files are held. In this example, the device is DB0:. Your directory may show a different device, but the name will be in the same form: two letters, a number, and a colon. The heading also identifies the UFD from which files are being displayed. In the example the UFD is [200,1].

In the directory listing, each file is identified by name, type, and version number and the date and time the file was created. The number between the filespec and the date is the number of *blocks* of disk space the file occupies. Every file is at least one block in size. The decimal point on the block count informs you that the number is a *decimal number,* a base-10 number. Version numbers, UFDs, and UICs are *octal numbers,* or base-8, and do not have decimal points. Although DCL does not require you to make these distinctions in commands, you should be aware that numbers on the system are sometimes octal and sometimes decimal.

At the bottom of the directory is a count of all the blocks used by files in the directory and of the number of files in the directory. The two block counts in the bottom line compare the number of blocks actually used and the number of blocks allocated for use. The latter number is usually higher because the system allocates blocks in groups, but uses them one at a time.

Using Wildcards to Specify Groups of Files

```
>DIRECTORY *.TXT (RET)


Directory DB2:[200,1]
08-JUN-81 15:17

FLY.TXT;3                1.        11-AUG-80 11:24
HELLO.TXT;1              2.        11-AUG-80 11:24
TEXT.TXT;1               7.        11-AUG-80 11:24
WHATSHERE.TXT;1          7.        11-AUG-80 11:24

Total of 23./25. blocks in 4. files

>DIRECTORY FLY.TXT;* (RET)


Directory DB2:[200,1]
08-JUN-81 15:17

FLY.TXT;1                1.        09-SEP-80 15:15
FLY.TXT;2                1.        08-DEC-80 15:05
FLY.TXT;3                1.        11-AUG-80 11:24

Total of 3./3. blocks in 3. files
```

- **The Asterisk (*) is a Wildcard. It Stands for "Match Zero or All Characters in this Position."**

Type DIRECTORY *.TXT and enter it.

You should get a directory listing only the most recent versions of all files with the file type .TXT. The asterisk (*) is a wildcard.

In this case, the asterisk directed the system to list all files in the UFD with the type .TXT regardless of file name. Notice also that since the DIRECTORY command did not specify any version number, only the most recent version of each file with the type .TXT is listed.

Now type DIRECTORY FLY.TXT;* and enter it.

This time you should get a list of the versions of FLY.TXT in the USER account. The wildcard means all versions of the file. If you had typed DIRECTORY FLY.TXT;2, the system would have displayed information on that file only.

```
>DIRECTORY F*.* RET


Directory DB2:[200,1]
30-DEC-80 15:18

FLY.TXT;3                1.        11-AUG-80 11:24
FUN.BAS;1                1.        30-DEC-80 15:11

Total of 2./6. blocks in 2. files


>DIRECTORY *F.* RET


Directory DB2:[200,1]
30-DEC-80 15:20

ESCF.TSK;1               4.     C  11-AUG-80 11:24

Total of 4./4. blocks in 1. FILE


>DIRECTORY *F*.* RET


Directory DB2:[200:1]
30-DEC-80 15:22

ESCF.TSK;1               4.     C  11-AUG-80 11:24
FLY.TXT;3                1.        11-AUG-80 11:24
FUN.BAS;1                1.        30-DEC-80 15:11

Total of 6./10. blocks in 3. files

>
```

- **The Wildcard (*) Can Be Combined with Letters and Numbers.**

Next type DIRECTORY F*.*. This time, all files in the USER account with file names starting with F are listed.

Try the other variations shown in the example.

DIRECTORY *F.* lists all files in the USER account with file names ending in F. DIRECTORY *F*.* displays a directory of all files in the USER account with file names that include an F in any position.

Of course, if there are no files that match the file name specified with the wildcard, you will get the following message:

```
DIR -- No such file(s)


>DIRECTORY %.* RET


Directory DB2:[200,1]
30-DEC-80 15:24

A.A;1                          1.        11-AUG-80 11:24
U.CMD;1                        1.        26-AUG-80 12:08
W.CMD;2                        1.        26-AUG-80 12:17
T.CMD;5                        1.        09-SEP-80 16:09

Total of 4./16. blocks in 4. files

>DIRECTORY *.% RET


Directory DB2:[200,1]
30-DEC-80 15:26

A.A;1                          1.        11-AUG-80 11:24

Total of 1./1. blocks in 1. file
```

- **The Per Cent Sign (%) is also a Wildcard. It Stands For "Match Exactly One Character in this Position."**

Type DIRECTORY %.*. The system lists files in the USER directory having a single-character file name and any file type.

Now type DIRECTORY *.%. The system lists files in the USER directory having a single-character file type and any file name.

You can use the wildcards with many DCL commands: COPY, TYPE, DELETE, PURGE, RENAME, and others. With a regular system of file names and types, the wildcards enable you to specify a large number of files without typing every filespec.

You can also use the asterisk (but not the percent sign) as a wildcard in the UFD portion of the filespec. For instance, try the command

```
>DIRECTORY [*,*]*.TXT RET
```

This command produces a list of the latest versions of all files in all .TXT directories on your mass-storage device. There may be a lot of them. Use CTRL/O to skip over some of the output. If this still seems to take too much time, use CTRL/C plus an ABORT DIRECTORY command to abort the listing.

You can display a list of all files in the directories on the default mass-storage device with the command DIRECTORY [*,*]. Try this command if you want a graphic illustration of what wildcards can lead to.

You cannot use any wildcard for device names. There are many kinds of devices on the system and not all of them are mass-storage devices. Permitting wildcards for device names would be either dangerous or wasteful or both.

Wildcards are a powerful tool for system users, but like other powerful tools, they must be used with care. Notice the way wildcards are used in the rest of this chapter.

Specifying Directory Formats

```
>DIRECTORY/BRIEF (RET)

Directory DB0:[200,1]

AZ.CMD;4
LOGIN.CMD;1
FLY.TXT;2
FLY.TXT;1
A.A;1
FLY.TXT;4
FLY.TXT;3
   .
   .
   .
TEXT.TXT;1
NEW.FIL;1
CLEAN.CMD;5


>DIRECTORY/FULL (RET)

Directory DB0:[200,1]
8-FEB-81 15:59

AZ.CMD;4              (7650,34)          1./1.              23-JAN-80 18:48
    [200,1]  [RWED,RWED,RWED,R]
LOGIN.CMD;1           (20423,13)         1./1.              14-DEC-80 10:37
    [200,1]  [RWED,RWED,RWED,R]
FLY.TXT;1             (20525,11)         1./15.             09-SEP-80 15:15
    [200,1]  [RWED,RWED,RWED,R]         22-JAN-79 12:32(2.)
FLY.TXT;2             (7636,45)          2./2.              08-DEC-80 15:05
    [200,1]  [RWED,RWED,RWED,R]
                                          .
                                          .
                                          .
NEW.FIL;1             (3521,443)         1./5.              29-FEB-80 00:00
    [200,1]  [RWED,RWED,RWED,R]
CLEAN.CMD;5           (10124,51)         1./1.              13-JUL-80 15:03
    [200,1]  [RWED,RWED,RWED,R]

Total of 234./250. blocks in 29. files
```

- **There are Three Display Formats for Directories: Default, Brief, and Full.**
- **Qualifiers to the DIRECTORY Command Specify the Display Format.**

Type DIRECTORY/BRIEF and enter it.

This command displays the directory in its shortest format. If you want to confirm or check a filespec quickly, you can use brief format. It prints on your

terminal about one-third faster than the standard format you saw in the previous examples. The brief format displays only limited information.

Now type DIRECTORY/FULL and enter it.

This command displays all the information available on the files in the directory. It takes about half again as long to print on your terminal as the standard format. Use CTRL/O if you don't want to see it all.

The number in parentheses after the filespec is the *file-ID number,* which gives the location of the *file header* on the disk. The remaining information, shown on the second line of each directory entry in the example, gives the UIC the file was created under and the *protection code* for the file.

The protection code details who may do what to the file. The meaning of the code is explained later. Most files take the default protection code, but in special instances, such as protecting the demonstration files in [200,1] from mistakes of new users, special protection codes may be needed.

The example given here may not match exactly the display you get from your system. File-ID numbers will be different and other details may vary, but the meaning of the display is the same.

```
>DIRECTORY/SUMMARY RET

Storage used/allocated for Directory DB0:[200,1]
8-FEB-81 16:01


Total of 4067./4195. blocks in 138. files


>DIRECTORY/FREE DB0: RET

DB0: has 6550. blocks free, 334120. blocks used out of 340670.
Largest contiguous space = 3103. blocks
13272. file headers are free, 13121. headers used out of
25993.
```

- **The /SUMMARY Qualifier to DIRECTORY Displays Information on the Space Occupied by a Directory.**
- **The /FREE Qualifier to DIRECTORY Displays Information on the Free Space on a Mass-Storage Device.**

Type DIRECTORY/SUMMARY and enter it.

After a brief delay, the amount of space used by and allocated for the UFD is displayed. If you want to know how much space another directory occupies, include the UFD in the command, like so:

```
>DIRECTORY/SUMMARY [303,5] RET
```

Now type DIRECTORY/FREE DB0: and enter it. The device name can be that of any mass-storage device displayed by the SHOW DEVICES command. After a delay, information about free space on the device named is displayed.

Printing Files

```
>TYPE (RET)
File(s)? TEXT.TXT (RET)
A HOLLOW VOICE SAYS "PLUGH."
>PRINT (RET)
File(s)? TEXT.TXT (RET)
>
```

- **The TYPE Command Prints Files on Your Terminal.**
- **The PRINT Command Prints Files on the Line Printer.**

Type TYPE and enter it. Type TEXT.TXT in response to the prompt, as shown in the example.

The file should be printed on your terminal.

Now type and enter PRINT, and type TEXT.TXT in response to the prompt.

The implicit monitor prompt returns, but nothing else happens.

Somewhere in your installation, probably close to the computer itself, there is a line printer. The file you named should be printed on the line printer. Find the line printer and then find the printed file. It will have the user name USER and the job name TEXT printed on the front.

The TYPE command is useful for looking at files on your terminal. If you need a hard copy or need to print a large file without tying up your terminal, you can use the PRINT command.

NOTE

Some installations restrict the use of the line printer. Find out whether you should test the PRINT command or not.

Copying Files

```
>COPY (RET)
From? LB0:[1,2]LOGIN.TXT (RET)
To? *.* (RET)
>DIRECTORY/BRIEF LOGIN.TXT;* (RET)

Directory DB0:[200,1]

LOGIN.TXT;1


>
```

- **The COPY Command Copies Files.**

Type COPY and enter it. (This is a DCL command, not an EDT command. The names are the same but a COPY command inside EDT is different from a COPY command at DCL monitor level.)

Type LB0:[1,2]LOGIN.TXT in response to the first prompt and enter it.

Type *.* in response to the second prompt and enter it. In this case, the wildcards mean that you do not want to change the name or the type of the file you are copying. Since you did not specify a device or a UFD, the COPY command defaulted to your device and UFD.

Now use the DIRECTORY command to check whether the copy was made. As you see, the wildcards automatically gave the same name and type to the copy as the original.

Type TYPE LOGIN.TXT and enter it. The file that is printed on your terminal when you log in should be printed on your terminal now, even though you are not logging in. You have copied a file from one location to another.

The file still exists in the original location, but now you have a copy in your own location as well.

Notice the device name in the response to the FROM? prompt. It is LB0:. LB0: is the name of a *pseudo device*. A pseudo device name is a pseudonym.

Any physical device can have more than one pseudo device assigned to it. This is done for convenience. Your installation may have any one of a number of models of mass-storage disks, but some device will have the pseudo-device name LB0: in addition to its physical name and possible other pseudo-device names. In this case, you may not know the name of the physical device holding the file [1,2]LOGIN.TXT, but you can be sure that the device had LB0: as one of its names because that is the way the system is set up.

At this point the most important pseudo device to you is TI:. TI: means the terminal you are logged in on, whatever its number.

The SHOW DEVICES command displays the names of physical devices and pseudo devices.

Renaming Files

```
>RENAME (RET)
Old file name? LOGIN.TXT (RET)
New file name? NEWNAME.* (RET)
```

- **The RENAME Command Renames Files.**

Type RENAME and enter it.

Type LOGIN.TXT in response to the Old file name? prompt and enter it.

Now type NEWNAME.* in response to the New file name? prompt.

The original file, the one that is printed when you log in, is still back in UFD [1,2] on LB0:. You have renamed your copy of the file.

Deleting Files

```
>COPY (RET)
From? NEWNAME.TXT (RET)
To? *.* (RET)
>COPY NEWNAME.TXT *.* (RET)
>COPY NEWNAME.TXT (RET)
To? NEWNAME.TXT (RET)
>DIRECTORY NEWNAME.TXT;* (RET)


Directory DB0:[200,1]
01-MAR-80 15:21

NEWNAME.TXT;1              1.        29-FEB-80 12:01
NEWNAME.TXT;2              1.        29-FEB-80 12:06
NEWNAME.TXT;3              1.        29-FEB-80 12:07
NEWNAME.TXT;4              1.        29-FEB-80 12:08

Total of 4./15. blocks in 4. files


>PURGE (RET)
File(s)? NEWNAME.TXT (RET)
>DIRECTORY NEWNAME.TXT;* (RET)


Directory DB0:[200,1]
01-MAR-80 15:22

NEWNAME.TXT;4              1.        29-FEB-80 12:07

Total of 1./5. blocks in 1 file


>DELETE (RET)
File(s)? NEWNAME.TXT;4 (RET)
>TYPE NEWNAME.TXT (RET)
TYP -- No such file(s)
SY0:[200,1]NEWNAME.TXT
```

- **The PURGE Command Eliminates All But the Latest Version of a File.**
- **The DELETE Command Can Eliminate One or More Versions of a File.**

As you may have noticed, it is quite easy to accumulate a number of versions of a file. For instance, every time you use the EDT EXIT command, you make a new version. In the next chapter, you will find yourself making many files

that are only temporarily useful. The PURGE and DELETE commands eliminate unwanted files. Here's how:

1. Use the COPY command to make new versions of NEWNAME.TXT. (Notice the different ways to issue the same command.) You will use the PURGE and DELETE commands to eliminate these new versions.
2. Check your directory to see how many versions of NEWNAME.TXT are in the UFD.
3. Now type PURGE and enter it.
4. Type NEWNAME.TXT in response to the prompt. Notice that you do not supply a version number (or semicolon) with this command.
5. When the prompt returns, check the directory to see how many versions of NEWNAME.TXT are left. You should now have only the most recent version of NEWNAME.TXT.
6. Now type DELETE and enter it.
7. Type NEWNAME.TXT;4 (or whatever version appeared in your last DIRECTORY listing) in response to the File(s)? prompt and enter it. This time you supply the semicolon and version number.
8. Now try to type NEWNAME.TXT on your terminal. You should get an error message.

The PURGE and DELETE commands are quite similar. However, the PURGE command will never eliminate all versions of a file, while the DELETE command will eliminate whichever versions you direct.

The DELETE command requires you to specify which version of a file you wish to eliminate. If you wish to eliminate all versions of a file, you must use the wildcard as the version number as follows:

```
>DELETE NEWNAME.TXT;*
```

The following DELETE command will delete every file in your directory except those that are specially protected:

```
>DELETE *.*;* RET
```

The wildcards can be dangerous when used with these commands. It is good practice to look at a directory for exactly the filespecs you plan to purge or delete before issuing the PURGE or DELETE command. If you are planning to issue the command

```
>DELETE *.FEL;* RET
```

you should issue the command

```
>DIRECTORY *.FEL;* RET
```

first to see just what you will be deleting.

The DELETE command requires you to specify which version or versions of files you wish to delete. If you wish to delete all versions of a file, you can use a wildcard, as shown:

```
>DELETE DOCTOR.FEL;*
```

If you do not specify a version number or wildcard in your DELETE command, DCL prompts you file by file for those you wish to delete, as shown:

```
>DELETE TEST.FIL
Delete file  DB2:[303,5]TEST.FIL;1    [Y/N/G/Q]? Y
Delete file  DB2:[303,5]TEST.FIL;2    [Y/N/G/Q]? N
Delete file  DB2:[303,5]TEST.FIL;3    [Y/N/G/Q]? G

The following files have been deleted:
DB2:[303,5]TEST.FIL;3
DB2:[303,5]TEST.FIL;4
DB2;[303,5]TEST.FIL;5
>
```

The answer Y means delete the file. The answer N means do not delete the file. The answer G means go ahead and delete all remaining files that otherwise match. The answer Q means quit deleting files.

CAUTION

> In general, you cannot purge or delete anyone's files but your own. However, you can purge or delete files from UFDs with the same group number as your own. The group number is the first number in the UFD or UIC.

Remember that a full filespec with device name, UFD, file name, file type, and version number can only refer to a single file, no matter how many files there may be on your system.

Naming Files

Files can have any name from zero to nine letters and numerals. They can have any type from zero to three letters and numerals.

If you liked, all your files could have the same name and type and differ only in version numbers, but that would be confusing. In general, your file names should be easy to remember and understand.

Default File Types

There are no restrictions on file types. You may find it convenient to give a group of related files the same file type so that you can maintain a subdirectory to be listed using the DIRECTORY command (as you did with the command DIRECTORY *.TXT.)

The system, however, uses a number of default file types for system tasks and functions. If you use these default file types, you won't have to include the file type in certain commands.

For instance, the file type .TSK is used for tasks that will run on the system. Thus, the default file type for the RUN command is .TSK. The following commands are identical:

```
>RUN ADVENT (RET)

>RUN ADVENT.TSK (RET)
```

If the task file was named ADVENT.BOB for some reason, you could issue the command:

```
>RUN ADVENT.BOB (RET)
```

and it would run, but

```
>RUN ADVENT (RET)
```

would not run (assuming you did not have another file named ADVENT with the type .TSK), because .BOB is not the default file type for the RUN command.

The default file types are particularly useful when preparing and running programs. This will be discussed in detail in the next chapter. The system documentation will tell you the default file types for different kinds of files and different system tasks.

Each computer language supported on RSX-11M/M-PLUS systems has its own default file type for source files.

You will find a table of the most important file types under *filespec* in the Glossary.

Summary

Files are important in RSX-11M/M-PLUS because of the way the system operates. You will learn more about this in the following chapters, but the general idea is that files are easy to move around on a *disk-based system.*

Disks are the most important mass-storage devices on RSX-11M/M-PLUS systems. The system itself is stored on a disk called LB0:, which is a pseudo-device name used to identify whatever disk the system is stored on.

The commands discussed in this section — DIRECTORY, RENAME, COPY, TYPE, PRINT, DELETE, PURGE — are used for file management. If you use the system, you need these commands, both for housekeeping in your own UFD and for smoothing your use of the system.

In particular, the PURGE and DELETE commands are useful for conserving disk space. You should not keep old copies of files that you do not need. If you have large files that you use rarely, you should not store them in your regular directory, but rather on a tape or disk that can be removed from the system.

You'll find more information on these file-management commands in the *RSX-11M/M-PLUS Command Language Manual.* The manual also explains how to use a private disk. See the description of the ALLOCATE command.

Chapter 4
How to Do Work on the System

Everything that is done on or by an RSX-11M/M-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. DCL... gives your SHOW TIME command a task name of its own, such as SHOT10, named after the command and the terminal from which it was entered (TT10:).
4. SHOT10 translates your SHOW TIME command into the equivalent MCR command, which is TIM, and passes that command to a task named ...MCR.
5. MCR goes to the Executive to find out what time it is, translates the time into a 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 of 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 transparent 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 should wish to see the commands in the utility's format, use the SET DEBUG command. See the HELP file for more information.)

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

Running Tasks Directly

```
>RUN $PIP (RET)
PIP> /LI (RET)


Directory DB0:[200,1]
28-MAR-80 14:33

A.A;1                          1.        12-FEB-80 13:16
AZ.CMD;1                       1.        13-MAR-80 15:38
COPY.CMD;2                     1.        16-FEB-80 12:27
EDT.CMD;5                      1.        28-MAR-80 13:10
LOGIN.CMD;4                    1.        20-MAR-80 13:20
HIYA.MAC;1                     5.        27-MAR-80 10:42
     .
     .
     .


PIP> (CTRL/Z)
>
```

- **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>) and then you can 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 a DCL DIRECTORY command. This is 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, signifying end-of-input, and return to DCL.

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

The RUN command itself is a task.

Loosely speaking, a task is a "computer program." Strictly speaking, a task is the fundamental executable unit in RSX-11M/M-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 the way that 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 enter the source program as a text file, using an editor or the CREATE command to make this source file.
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

```
>TYPE   HIYA.MAC (RET)
         .TITLE  HIYA
         .LIST   TTM
         .NLIST  BEX
         .ENABL  LC

; MACRO LIBRARY CALLS

         .MCALL  EXIT$S,QIOW$,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
          .
          .
          .
```

- **Source Files Are in Readable Format.**

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*.

Print the file on your terminal using the TYPE command. You may want to use CTRL/S and CTRL/Q.

The source program is a text file. You can read it.

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* MSG1: 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/M-PLUS systems. It was chosen for this demonstration because all RSX-11M/M-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,* which is an acronym for American Standard Code for Information Interchange. ASCII is a

universal 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, however, 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

```
>DIRECTORY HIYA.*;* RET


Directory DB0:[200,1]
20-MAR-80 10:07

HIYA.MAC;1                6.        19-MAR-80 08:58

Total of 6./6. blocks in 1. file
>
>MACRO RET
File(s)? HIYA RET
>
>DIRECTORY HIYA.*;* RET


Directory DB0:[200,1]
20-MAR-80 10:08

HIYA.MAC;1                6.        19-MAR-80 08:58
HIYA.OBJ;1                2.        20-MAR-80 10:08

Total of 8./8. blocks in 2. files
```

- **The MACRO Command Invokes the MACRO–11 Assembler.**
- **The MACRO Command Defaults to the 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.

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 did not give a file type or a version number.

The MACRO command invokes the MACRO–11 Assembler. The version number defaults to the most recent version, 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 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 type .OBJ, identifying it as an object file. This file type is the default for *output files* produced by the MACRO-11 Assembler.

Notice that the files are not of the same size. The object file is smaller because it has been translated from the less-efficient human-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.

```
>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 =
>
```

- **Object Files Are Machine-Readable.**

Now print the object file on your terminal using the TYPE command. This will work on either a video or hard-copy terminal, but it will use a lot of paper on the hard-copy 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.

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

Now print it again, but this time stand by with CTRL/S or the NO SCROLL 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 humans, they are left as is.

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

Here is what has 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 the 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,* that converts a binary number to a signed decimal ASCII representation. $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

```
>LINK (RET)
File(s)? HIYA (RET)
>DIRECTORY HIYA.*;* (RET)

Directory DB0:[200,1]
20-MAR-80 14:06

HIYA.TSK;1                5.     C  20-MAR-80 14:04
HIYA.OBJ;1                2.        20-MAR-80 10:08
HIYA.MAC;1                6.        19-MAR-80 08:58

Total of 13./13. blocks in 3. files
```

- **The LINK Command Invokes the Task Builder.**
- **The LINK Command Defaults to the File Type .OBJ.**
- **The Task Builder Transforms an Object File into a Task Image File.**

Type the command LINK HIYA, after the prompts shown.

Even though there are now two 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 properly identifying 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.

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

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

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 made has been contiguous. A noncontiguous file may be scattered all over the disk with a *file header* containing a map of the blocks used in the file. Task image files must be contiguous, meaning they are not scattered but together in one location. Not all contiguous files are task images, but all task images must be 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 made contiguous for the same reason, to save time on I/O.)

```
>TYPE HIYA.TSK (RET)

@O~SYSYSYSYTICLzI@Tx~zVxTI@Could I have your name please?RSX-11M-PLUS CALL-
ING @zAw  @APw

                                        %ff

     .
     .
     .
N c
   v
        W%
>
```

- **Task Image Files Are Not Readable.**
- **Task Image Files Are Different from Object Modules.**

Print the file on your terminal using the TYPE command. You may prefer to wait for a chance to try this on a video terminal if you have a hard-copy 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 readable, however, if you can catch them.

Here is what has 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. No task run under a multiuser system like RSX-11M/M-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

```
>RUN (RET)
Task? HIYA (RET)

Could I Have your name please?
Bo Diddley (RET)
RSX-11M-PLUS calling Bo Diddley
>
```

- **The RUN Command Installs, Runs, and Removes Tasks.**
- **The RUN Command Defaults to the File Type .TSK.**

Now issue a RUN command.

When you receive the prompt What?, type the name HIYA. You do not need to specify the file type .TSK, because the RUN command defaults to .TSK.

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 it
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 a 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 for the task is available in mem-

ory), 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's 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 rather than the RUN command.

Using Subroutines

```
>EDIT/EDT/OUTPUT:NEWHI.MAC HIYA.MAC
*'EXIT$S' (RET)
8                .MCALL EXIT$S,QIOW$S,DIR$,GTSK$
*'EXIT$S' (RET)
50               EXIT$S                  ; LEAVE
*I (RET)
(TAB)CALL(TAB) STARS(TAB);CALLS STARS SUBROUTINE (RET)
(CTRL/Z)
*EXIT (RET)
DB0:[200,1]NEWHI.MAC;1 150 lines

>
```

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

This is a new way of invoking EDT for you. Previously when you invoked EDT, you specified no 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 will give the output file a new name, one that does not appear in the UFD, that will thus be version 1. The file NEWHI.MAC will be created automatically 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 filespec (by explicit action).

When EDT returns with its prompt, search for the *second* instance of the string *EXIT$S*. 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 this 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 line:

```
(TAB)CALL(TAB) STARS(TAB); CALLS STARS SUBROUTINE (RET)
```

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.

Get a directory of .MAC files in the UFD. 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 you just inserted a call for.

```
>MACRO NEWHI (RET)
>MACRO STARS (RET)
>LINK NEWHI, STARS (RET)
>(RET)
>RUN NEWHI (RET)

Could I have your name please?
Rachmaninoff (RET)
RSX-11M-PLUS calling Rachmaninoff
**********************************
>
```

- **An Object Module Can Contain a Subroutine.**
- **Subroutines 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 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.

Issue the LINK command, naming both object modules: NEWHI, STARS. Separate the module names with a comma.

When the Task Builder is through, run NEWHI.

As you see, the STARS subroutine has altered the performance of the task.

When you built HIYA.TSK before, 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 library to try to resolve them.

Try linking NEWHI.OBJ without STARS.OBJ and 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

Thus far, this demonstration has concentrated on the MACRO-11 Assembly Language because the MACRO-11 Assembler is *bundled* with every RSX-11M/M-PLUS system. Most systems will also 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 assembles as 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 compiles as 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 the way the hardware works, but generally speaking a FORTRAN program is transportable from one computer to another and 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, while input files — source text files — are transportable, output files from the compiler — 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 is 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/M-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 Operating System and 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 for the source language. Thus, the object file is a common goal for any assembler or compiler operating on RSX-11M/M-PLUS systems, a target at which they all aim.

The high-level languages are supported, but they are not bundled. This means 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 analagous to the MACRO command for the most popular languages. There is a FORTRAN command and two different FORTRAN compilers 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-11 and BASIC-PLUS-2, the two forms of the Beginners All-purpose Symbolic Instruction Code supported on RSX-11M/M-PLUS systems, are slightly different.

A BASIC command is supplied, but BASIC-11 is a miniature operating system in itself. Programs in BASIC-11 are never compiled. The BASIC-11 system interprets programs line by line. BASIC-11 is a task that accepts, interprets, and executes programs written in BASIC-11. In effect, your program is data for the BASIC-11 task.

BASIC-PLUS-2 programs can be compiled, but a special procedure is used. BASIC-PLUS-2 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 EXIT$S 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/M-PLUS systems include the capability of calling assembly-language subroutines. These subroutines can often increase the speed of efficiency of your program.

A further demonstration of running tasks follows.

How Tasks Are Named

```
>RUN $PIP (RET)
PIP> (CTRL/C)
DCL>SHOW TASKS/ACTIVE (RET)
SHOT10   (TT10:)
DCL...   (TT10:)
TT10     (TT10:)
(CTRL/Z)
>
```

- **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.

Enter CTRL/C instead of a command. The explicit DCL prompt shows that you have interrupted the execution of the PIP task and can issue a DCL command.

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

DCL returns a list of the tasks active at your terminal. As before, the SHOW task is identified by a name in the form SHOTnn.

- 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 the way 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.

(In RSX-11M-PLUS installations with more than 64 terminals, the terminal number in task names may include a letter. See the *RSX-11M/RSX-11M-PLUS Command Language Manual* for more information on task naming.)

```
>RUN $PIP (RET)
PIP> (CTRL/C)
DCL>RUN HIYA (RET)
RUN -- Task name already in use
>
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/Z)
>RUN/TASK_NAME:LOLA $PIP (RET)
LOL> /LI (RET)


Directory DB0:[200,1]
28-MAR-80 14:33
A.A;1                          1.        12-FEB-80 13:16
AZ.CMD;1                       1.        13-MAR-80 15:38
COPY.CMD;2                     1.        16-FEB-80 12:27
EDT.CMD;5                      1.        28-MAR-80 13:30
LOL>
```

- **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.

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.

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.

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

Aborting Tasks

```
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 MCR
                  and with pending IO requests
```

- **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 much the same way as 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 named directly after your terminal. Since you overrode that task name, as confirmed by SHOW TASKS/ACTIVE in the example, you must abort the task by specifying the name you gave it.

Type ABORT/TASK LOLA and enter it as shown.

Other References

This completes the terminal warm-up session.

Now you should look up the commands you have learned in the *RSX-11M/M-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 *Command Language Manual* you may want to browse 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 remainder 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. The next chapter includes a demonstration of the SHOW MEMORY command, which produces a live-action picture of what is happening on the system. The final chapter introduces some of the conveniences of RSX–11M/M–PLUS systems.

Part II
Learning the System

Chapter 5
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 the RSX-11M/M-PLUS Operating Systems. You can follow up on these generalizations to learn the particulars of the system you are using. Suggestions for further reading appear at the end of this chapter.

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.

Applications and Operating Systems

RSX-11M/M-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 applications tasks.

RSX-11M/M-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 such as was demonstrated in Part I.

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

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 communications 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 an 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 Time-Sharing Environment

The general-purpose time-sharing 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 them are thinking, looking up commands, or the like between keystrokes most of the time.

In the general-purpose time-sharing 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/M-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 that installation. Therefore, the best sources of information about the operating system are the system manager, in-house documentation, and other people who use it.

Hardware and Software

RSX-11M and RSX-11M-PLUS offer a wide range of services and utilities. The system supports many kinds of input and output devices. RSX-11M and RSX-11M-PLUS are designed as general-purpose systems 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 custom system. The system you are using has been generated to include your installation's mix of hardware and software.

Since 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 time-sharing 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.

Because the RSX-11M Operating System runs on all PDP-11s, from the largest to the smallest, there are more system-generation options on RSX-11M. If the omitted option is needed, it is generally possible to generate a new RSX-11M system including the option. See your system manager for further information.

The RSX-11M-PLUS Operating System runs only on processors with 22-bit *addressing*. Since these machines were designed more recently, they have more ample memories and other modern features. Therefore, RSX-11M-PLUS has fewer system-generation options and provides software

features that take advantage of the hardware features of the processors RSX-11M-PLUS runs on. 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.

For the most part, there is little difference for the terminal user between a standard RSX-11M Operating System and an RSX-11M-PLUS Operating System. In the system documentation, these differences are highlighted by shading.

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.

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:

1. Keep track of all resources.
2. Enforce policy on who gets what resources, when, and how much.
3. Allocate the resources according to system policies.
4. Reclaim the resources when they are no longer needed.

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

1. *Memory,* the system's workspace, where active tasks, their data, and the Executive itself are located.
2. The Central Processing Unit, or CPU, the part of the computer that executes instructions or computes.

3. Peripheral devices, the input and output devices, including mass-storage disks, line printers, terminals, and the like.
4. 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.

All four of these resources are explained further in this part of the manual. Before these explanations, however, you should understand privilege, priority, and file protection, three control mechanisms that are universally applied to system users.

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 user 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, and 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 time-sharing 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.

The ways in which the system uses priority to control access to system resources include *checkpointing* and *swapping,* which are explained in the discussion of memory.

Control Through File Protection

Finally, the system 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.

Resource: The 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 the 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 the *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 of 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. Unless it has been checkpointed, an active task is *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 been thus 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 priorities 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.

Resource: 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 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:

1. The completion of input or output. If a task is waiting for I/O or cannot continue its I/O because of the unavailability of the I/O device, then it has no further need for control of the CPU.
2. The execution of a task.
3. 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.
4. The execution of the *round-robin scheduler*. The round-robin scheduler is a form of time-sharing that overcomes the Executive's tendency to give the most CPU time to tasks that appear first in the System Task Directory (STD). The round-robin scheduler rotates entries in the STD 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.

The SHOW MEMORY Command

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

```
>SHOW MEMORY (RET)
```

This command displays most of the information in the pool. In particular, it displays the contents of memory and the task that currently controls the CPU.

If you issue the SHOW MEMORY command from a hard-copy 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 set to this memory display, which is called *RMD*. See Figure 5–1.

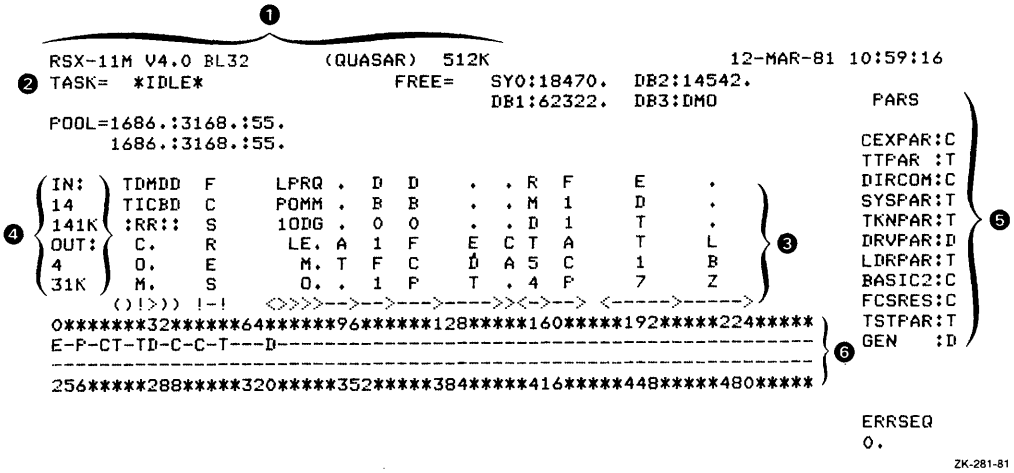

Figure 5–1: SHOW MEMORY Display

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

Look in the upper left-hand corner of the display. The top line ❶ gives the system name and version number and the size of the system's memory. Just under this line ❷ you will see a task name. This is the task that is currently controlling the CPU. Sometimes, this task name will be *IDLE*, signifying that no task is controlling the CPU. Most of the display is taken up by a list of task names ❸. These are the tasks currently resident in memory.

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, RMDEMO displays graphically the way that the Executive controls the contents of memory and use of the CPU.

The angle brackets and hyphens under the names of tasks resident in memory are roughly proportional to the size of the tasks.

The IN/OUT display ❹, under the currently executing task, gives a count of how many tasks are in memory and how many words of memory they occupy, a count of how many tasks are out of memory or checkpointed, and how many words of memory they occupied when they were memory-resident.

The names of all partitions in the system are displayed on the right-hand side of the display ❺, and other information about the partitions is given in the center of the display — the two lines of asterisks and numbers ❻.

There is a more detailed explanation of the meaning of this display in the *RSX-11M/M-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, you may have a display like that shown in Figure 5–1, a number of smaller tasks 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, use CTRL/Z to cancel the SHOW MEMORY display.

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

Resource: The 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 this manual, we have given attention to three devices found on all RSX-11M/M-PLUS systems. These are the terminal, the line 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 line printer is strictly an output device.

The terminal and the line 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/M-PLUS systems. Several DIGITAL terminals and printers among the record-oriented devices and a dozen or more file-structured disk and tape devices are supported. This support is primarily 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. Virtual terminals are used in *batch processing,* which is described in a later section.

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.

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

Resource: 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 plane, 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.

Since RSX-11M and RSX-11M-PLUS are disk-based systems, 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 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 INITIALIZE command.

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 5-2.) Each UFD is a file containing the names of the 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.

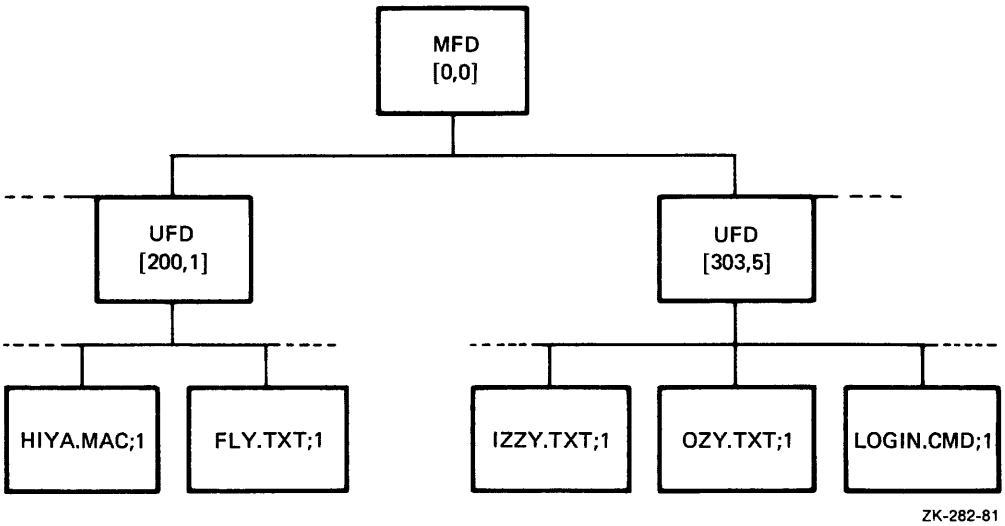

Figure 5-2: Structure of Files on a Volume

Tasks running on RSX-11M or 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.

Chapter 6
Some System Conveniences

Here is some information on some of the user conveniences included on most RSX-11M and RSX-11M-PLUS Operating Systems. These include the BROADCAST command, for sending messages to other terminals; the PRINT command, for printing files on your system's line printer; and indirect command files and batch processing, two methods of passing commands to the system automatically.

Batch processing is available on RSX-11M-PLUS systems only.

Broadcasting Messages

Type the following command on your terminal and enter it:

```
>BROADCAST TI: RET
Message? No matter where you are, there you are RET

08-FEB-81  10:33        FROM TT56:      TO TI0:
NO MATTER WHERE YOU ARE, THERE YOU ARE

>
```

You have just sent a broadcast message to yourself. You could have specified your own terminal number instead of TI: or some other terminal number to send a message to another system user. Broadcast messages generally break through whatever is appearing on the terminal at the time the message is sent. If the terminal is not available, you will be informed by a message.

On RSX-11M-PLUS systems, you can also send messages to other users by name.

The PRINT Command

Earlier in this manual, the simplest form of the PRINT command was introduced:

```
>PRINT TEXT.TXT RET
```

The PRINT command has a number of qualifiers that permit you considerable control over when, where, and how your files will be printed. One PRINT comand can name several files to be printed. Simply list the files, separated by commas, as follows:

```
>PRINT FLY.TXT;1,TEXT.TXT,FLY.TXT;3 (RET)
```

The files will be printed end to end, separated only by *form feeds*. As the line printer reaches the end of one file, it simply moves the paper up past the next perforation and starts printing the next file. If you wish to separate the files more distinctly, you can use the /FLAG_PAGE qualifier:

```
>PRINT/FLAG_PAGE FLY.TXT;1,TEXT.TXT,FLY.TXT;3 (RET)
```

In the first example, the listing of all the files is preceded by a job flag page, which identifies the job in banner style with the job name FLY, taken from the first file in the job. In the second example, the listing for each file is preceded by a file flag page, which identifies the file in banner style.

You can also request multiple copies of files with the /COPIES qualifier:

```
>PRINT/COPIES:2 TEXT.TXT (RET)
```

The /LENGTH qualifier to the PRINT command automatically formats files into pages of equal length.

For instance, standard line-printer paper is 66 lines long. If you print a file thas has no form feeds in it, it is likely that line 67 will be printed over the perforation in the paper. To avoid this, specify the page length (/LENGTH:60) with your PRINT command and the line printer will automatically jump to the top of a new page after printing 60 lines. If the printer encounters a form feed before 60 lines, it will jump to the top of a new page and start counting again.

If you wish to include form feeds in a text file, use CTRL/L on a line by itself when you are creating the file. The CTRL/L will appear in your text as <FF> if you use EDT. Other editors use different symbols. When a form feed is printed on a terminal, it appears as four line feeds. Combining CTRL/L and the /LENGTH qualifier is an easy way to keep your printed listings neat.

The PRINT command queues files for printing. You can display the contents of the print queues with the SHOW QUEUE command. See the *RSX-11M/M-PLUS Batch and Queue Operations Manual* for complete information on using the PRINT and SHOW QUEUE commands.

Automatic Command Entry

Often, 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. RSX-11M/M-PLUS includes an indirect command processor to pass commands automatically to the system.

In addition, RSX-11M-PLUS systems include batch processing.

With both indirect command processing and batch processing, you place the command or series of commands you wish executed in a file and pass the file to the system for further processing. Otherwise, the two are quite different.

Indirect Command Files

List the file [200,1]LOGIN.CMD on your terminal using the TYPE command:

```
>TYPE LOGIN.CMD (RET)
        .ENABLE QUIET
        .IF <CLI> EQ "MCR" SET /DCL=TI:
        TYPE HELLO.TXT
>
```

The file on your system may differ, but it should include these three lines. LOGIN.CMD is an indirect command file. Indirect command files are used to pass commands to the operating system or its tasks. Indirect command files can include commands or indirect directives (preceded by a period).

Indirect command files are processed by a task called the indirect command processor, which is shown as AT. in the list of installed tasks. The indirect directives are commands to this processor.

In this example, the directive .ENABLE QUIET suppresses the echoes from processing the commands that follow. The commands are processed, but you do not see them on your terminal.

The directive .IF <CLI> EQ "MCR" SET /DCL=TI: can be translated as "Test the CLI of this terminal and if it is MCR, issue the MCR command that sets it to a DCL terminal." In other words, this directive tests for a condition and acts on the results of that test.

If the special symbol <CLI> is set to MCR, then the terminal is also set to MCR. In that case, the indirect processor issues the MCR command to set the terminal to DCL. If the terminal is already set to DCL, the indirect processor goes on to the next line in the file. In other words, the directive assures that the terminal will be set to DCL so that it will accept and process the next command in the file, TYPE HELLO TEXT, which is a DCL command.

The command TYPE HELLO.TXT is processed and executed just as if you typed it in.

When you log in, the system first prints the file called LB0:[1,2]LOGIN.TXT on your terminal. Then the system checks your directory for the file called LOGIN.CMD. This file contains commands that you wish to have executed every time you log in. For example, LOGIN.CMD files commonly include the command SET TERMINAL/LOWER in case the terminal you are using has been set to uppercase.

As you see, the indirect command processor can be programmed. In fact, the indirect command processor can be programmed elaborately. Many common programming techniques, such as looping, counters, variables, labels, and arithmetic and logical operations, are available to you. In addition, a number of special symbols, like <CLI>, are available for use, with many directives that test system conditions.

Indirect command files are not limited to use at login time. The indirect command file processor is called AT., because the at-sign (@) is used to invoke an indirect command file.

Type and enter the following command:

```
>@LOGIN (RET)
```

The same routine that is executed automatically when you log in to the USER account is executed on your command. Note that the default file type for the indirect command processor is .CMD.

List the file SHOW.CMD on your terminal using the TYPE command:

```
>TYPE SHOW.CMD (RET)
        SHOW DEVICES
        SHOW USERS
        SHOW TIME
```

Now type and enter the command:

```
>@SHOW (RET)
```

All three SHOW commands included in the file are executed, one after the other. Because there was no .ENABLE QUIET directive included in the file, the commands are echoed as they are entered.

Here is another example of an indirect command file that is also a program. This example is an interactive program. List the file DELETE.CMD on your terminal, using the TYPE command:

```
>TYPE DELETE.CMD (RET)
        .ENABLE SUBSTITUTION
.BEGIN:
        .ASKS FIL Which file?
        TYPE 'FIL'
        .ASK DEL Delete it
        .IFT DEL DELETE 'FIL';*
        .GOTO BEGIN
>
```

You can use this file to clean up your directory.

Here is an example of the file in use:

```
>@DELETE (RET)
>* Which file? [S]: SAMPLE.TXT (RET)
>TYPE SAMPLE.TXT
This is a sample text file to be deleted.
>*Delete it? [Y/N]: Y (RET)
>DELETE SAMPLE.TXT;*
>* Which file? [S]: EXAMPLE.TXT (RET)
>TYPE EXAMPLE.TXT
This example file must not be deleted.
>* Delete it? [Y/N]: N (RET)
>* Which file? [S]: AMPLE.TXT (RET)
>TYPE AMPLE.TXT
One file should be ample. Delete this one.
>* Delete it? [Y/N]: Y (RET)
>* Which file? [S]:(CTRL Z)
>* <EOF>
>
```

The user had been asked to clear some space on the disk by eliminating unnecessary files from her directory. The user found three files that might be eliminated but was not sure what the files had in them. The indirect command file DELETE.CMD is designed for this situation. This file asks the user for the filespec of a file that might be unnecessary, types the file on the terminal, and then asks the user if the file should be deleted. Depending on whether the user responds Y or N, the file is either deleted or not. Then the indirect processor loops back to the label at the beginning of DELETE.CMD and asks for another filespec. This process continues until the user enters a CTRL/Z, signifying that the indirect command processor should exit. (Note that the TYPE command in DELETE.CMD types only the most recent version of the file, but the DELETE command deletes all versions if you answer Y.)

You can create a few files to be deleted and try this out for yourself.

In many common situations, even a simple indirect command file like DELETE.CMD can save you typing.

Indirect command files can pass commands to any task, not just to DCL and MCR. For instance, the BROADCAST command accepts an indirect command file in place of the terminal number and message. List the file SHAVE.CMD on your terminal, using the TYPE command:

```
>TYPE SHAVE.CMD (RET)
TI:DON'T LOSE YOUR HEAD
TI:TO GAIN A MINUTE
TI:YOU NEED YOUR HEAD
TI:YOUR BRAINS ARE IN IT
```

As you see, the file contains a list of terminals and messages to be sent to them. Now issue the BROADCAST command as shown:

```
>BROADCAST @SHAVE (RET)
```

The messages come to your terminal in order. As in the previous BROADCAST example, the TI: will in most cases be replaced by the terminal number to which you wish to send your message.

You can use indirect command files to control or program the operation of any system or applications task that accepts commands. For a full explanation of the indirect command processor, see the *RSX-11M/M-PLUS MCR Operations Manual.* As you will see, the indirect directives and special symbols constitute a complete programming language.

Batch Processing (RSX-11M-PLUS)

Batch processing is an alternative method of passing commands to the operating system by remote control. The text that follows illustrates the way batch processing works on RSX-11M-PLUS. Note, however, that some installations place limits on using the batch processors.

Batch jobs differ from indirect command files in that a batch job is a complete terminal session, while an indirect command file is only part of a terminal session. You must be logged in to a terminal to run an indirect command file,

but you can run a batch job long after you've logged out and gone home. A batch job runs on a special kind of terminal called a virtual terminal that is really software.

Another difference between batch processing and indirect command file 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.

List the file BATCH.BAT on your terminal, using the TYPE command:

```
>TYPE BATCH.BAT (RET)
$JOB HIYA [200/1]
$MACRO HIYA
$LINK HIYA
$RUN HIYA
$DATA
WILLIE NELSON
$EOD
$!COMMENT: HIYA ASSEMBLED, LINKED, AND RUN
$RUN HIYA
BASIL WOLVERTON
$!COMMENT: HIYA RUN AGAIN WITH DIFFERENT DATA
$EOJ
```

This is a complete *user batch job*. It does much of the same work that you did in Chapter 4.

In the following example, you pass the batch job to a batch processor with the SUBMIT command:

```
>SUBMIT/AFTER:(17:00) BATCH.BAT (RET)
```

The SUBMIT command given here places the batch job in a batch queue, from which it will be run after 17:00 (5 p.m.) on the day it was submitted. When the job runs, it produces the following log, which is printed on the line printer:

```
QMG Batch Job - BATCH          BPR  V02      17-Mar-81 13:48   Page 1
Processor BAP0


13:48:56        $JOB HIYA [200,1]

          ====================================
          User Job - HIYA         Terminal VT2:
                     UIC = [200,1]
          ====================================

          TERM
            .    RSX-11M-PLUS V01 BL7   MULTI-USER [3,54] SYSTEM
            .
            .   03-Mar-81
            .           Our other line printer has arrived, Installation will
            .           begin shortly, Happy days are here again!
            .
13:48:59        $MACRO HIYA
13:49:09        $LINK HIYA
13:49:16        $RUN HIYA
          TERM  Could I have your name please?
          DATA  WILLIE NELSON
          TERM  RSX-11M-PLUS calling WILLIE NELSON
            .   >
```

```
 13:49:17        $EOD
 13:49:17        $! COMMENT: HIYA ASSEMBLED, LINKED, AND RUN
 13:49:17        $RUN HIYA
          TERM   Could I have your name please?
          DATA   BASIL WOLVERTON
          TERM   RSX-11M-PLUS calling BASIL WOLVERTON
                 >
 13:49:18        $! COMMENT: HIYA RUN AGAIN WITH DIFFERENT DATA
 13:49:18        $EOJ
          TERM   CONNECT TIME:  1     MINS.
                 CPU TIME USED: 11    SECS.
                 TASK TOTAL:    28
```

This log includes a record of all the 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.

In batch jobs a dollar sign ($) precedes DCL commands. The dollar sign notifies the batch processor that a command follows.

The JOB command logs the batch job on to 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 MACRO, LINK, and RUN commands work as usual. When the task is run, it requests data — a name — from the virtual terminal. The DATA command identifies the following line as data. The data is passed and processed and the response of HIYA.TSK appears. The EOD command marks the end of the data.

Notice that 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.

Then, HIYA.TSK is run again. This time the data — another name — is handled differently. As stated, the dollar sign notifies the batch processor that the line contains a command. Therefore, a line without the dollar sign in the first position notifies the batch processor that the line contains data. As you see, both runs of HIYA completed successfully.

The DATA and EOD commands are not necessary in most cases. They were included in the example to alert you to the capability of including data in batch jobs.

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

A Final Word

This completes the *Introduction to RSX-11M and RSX-11M-PLUS*. You are now ready to use the most common functions of the operating system. As you grow accustomed to using the system, you should continue to explore the many facilities it provides.

Follow up on the further reading suggested earlier in this manual.

For more information on indirect command files, see the *RSX-11M/M-PLUS MCR Operations Manual.* This manual also contains information on the system from the operator's point of view. You can compare MCR with DCL.

The *RSX-11M/M-PLUS Batch and Queue Operations Manual* presents complete information on batch processing, the PRINT command, and the Queue Manager.

The *IAS/RSX-11 I/O Operations Reference Manual* covers FCS. For more information on RMS-11, see the RMS-11 documentation supplied with your system.

Finally, the system *Information Directory* describes each manual included in the system documentation. The *Mini-Reference* and *Master Index* are also useful.

Take it easy.

Part III
Glossary

account

Each system user, including parts of the system itself, is identified by an account number. Individual users are identified by an account name as well. Account numbers are made up of two octal numbers in themember. When you log in, you log in under a particular account name or number. This number informs the system where your files are and what kind of access to other files and system facilities you should be given. See *UIC* and *UFD.*

User accounts are either privileged or nonprivileged. Privileged users (those having group numbers equal to or lower than 10) have access to all system commands and to all parts of the system. Nonprivileged users are limited to everyday operations that do not threaten the integrity of the operating system.

acronym

Here are some of the acronyms most commonly used on RSX-11M/M-PLUS systems. Acronyms are often called *mnemonics.* Those acronyms in *italics* are defined elsewhere in this glossary.

Acronym	Meaning
ASCII	American Standard Code for Information Interchange
BAD	Bad Block Locator utility
BRU	Backup and Restore utility
CLI	Command Line Interpreter
CO:	Console Output pseudo device
CMP	File Compare utility
CRF	Cross-Reference Program for task maps
CRT	Cathode ray tube; video terminal
DCB	Device Control Block
DCL	DIGITAL Command Language
DEC	Digital Equipment Corporation
DMP	File Dump utility
DSC	Disk Save and Compress utility
DSR	Dynamic Storage Region (pool)
EDI	Line Text Editor
EDT	DEC Editor
FCB	File Control Block
FCS	File Control Services
FLX	File Transfer utility

Acronym	Meaning
FMT	Disk Formatter utility
I/O	Input/output
IOX	I/O Exerciser
LB:	The system library pseudo device
LBR	Librarian utility
LUN	Logical Unit Number
MAC	MACRO–11 Assembler
MCR	Monitor Console Routine
MFD	Master File Directory
PAT	Object Module Patch utility
PIP	Peripheral Interchange Program
PMD	Postmortem Dump
QMG	Queue Manager
RMS–11	Record Management Services
SLP	Source Language Input Program
STD	System Task Directory
SY:	User's default device
TCB	Task Control Block
TI:	Terminal pseudo device
TKB	Task Builder; the linker
UFD	User File Directory
UIC	User Identification Code
VFY	File Structure Verification utility
VT:	Virtual terminal
ZAP	ZAP utility, used to patch task images

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 systems run on 16-bit, 18-bit, and 22-bit processors. A 16-bit processor cannot address more than 32K bytes of memory; an 18-bit processor cannot address more than 256K bytes; a 22-bit processor can address up to 3.8 megabytes of memory.

RSX-11M-PLUS systems run only 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 a task that performs a specific job for the user. In general, the term refers to any task that is not part of the operating system or of a programming language.

Applications tasks are written for the installation, to monitor an industrial process, for example. Compare with *system task.*

ASCII

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

Thus, an ASCII file is a file which 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 and RSX-11M-PLUS.

Assembly language is used to generate binary machine code. See *binary machine code* for an example of the difference between the two. See also *MACRO-11 Assembly Language.*

back space key

System software does not use the BACK SPACE key. Results from its use will be unpredictable. Any line containing a BACK SPACE character (a nonprinting character) will be rejected or misinterpreted.

For correcting mistakes, use the DELETE key (sometimes called RUBOUT).

The BACK SPACE key is sometimes used by applications tasks.

batch chain

See *QMG batch job.* RSX-11M-PLUS only.

batch job

This term has two meanings. See *QMG batch job* and *user batch job.* RSX-11M-PLUS only.

batch processing

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

You do not have to be present when your batch job is run, nor do batch jobs require a physical terminal to run.

Compare with *indirect command file.* Batch processing is available on RSX-11M-PLUS only.

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.

block

A block is a unit of measurement for files. In almost all cases, a block is 256 words (512 bytes). A block is the smallest addressable unit of data that a random-access device can transfer in an input or output (I/O) operation.

The term block is also used to refer to an arbitrary number of contiguous bytes used to store logically related status, control, or other processing information, such as a User Control Block, File Header Block, or Task Control Block.

blocked task

A blocked task is a task that cannot execute. Blocked tasks do not compete for CPU time, or memory, but they are still considered active. Most commonly, a blocked task cannot execute because it is awaiting some input or some other information from the system, such as an event flag, indicating that some event has occurred. Tasks can also be blocked with the STOP/BLOCK (MCR BLK) command.

See also *task state.*

boot

Boot is short for bootstrap. In computer terminology, a bootstrap is a technique or device designed to bring itself into a desired state by means of its own action, such as a routine whose first few instructions are sufficient to bring the rest of itself into memory from an input device.

In RSX-11M/M-PLUS, bringing a fresh operating system into memory is called booting. The system must be booted after a crash. MCR includes a BOOT command.

buffer

The word buffer is a commonly used computer term referring to a temporary storage area used in performing I/O operations. In this manual, the term refers to the buffers created by EDT for your use in creating and altering files. EDT always starts out with a single buffer, named MAIN, for you to do your work in. You can create and name alternate buffers to try different formats, for instance, or to hold a file you wish to extract some information from for use in the MAIN buffer. See the *EDT Editor Manual* for more information on EDT and its buffers.

bundled

A bundled product is one that is always provided as part of the operating system. The MACRO-11 assembler is an example of a bundled product, one that is included in all systems.

The COBOL compiler is unbundled. It is not included in all systems. It must be purchased separately.

cathode ray tube

Video terminals receive output on a cathode ray tube. Also called a CRT. See *terminal.*

change mode

Change mode in EDT is the alternate to line mode. Line mode operates on text one line at a time. Change mode operates on text one character at a time. Change mode is particularly suited tos. See the discussions of keypad and nokeypad editing in the *EDT Editor Manual* for more information on change 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 checkpointed.

circumflex

The circumflex looks like this:

^

On most terminals it is typed by holding down the SHIFT key and pressing the 6, but it may be located elsewhere on your terminal. The circumflex is used by RSX-11M/M-PLUS to indicate in terminal output that a control character has been pressed. Thus, CTRL/Z is echoed on your terminal as follows:

^Z

The circumflex is used in the BASIC-11 and BASIC-PLUS-2 languages to indicate exponentiation.

You should not confuse the circumflex with the up-cursor key on the alternate keypad, which is used in EDT change mode.

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-11M and RSX-11M-PLUS provide two CLIs. For further information, see *DCL* and *MCR*.

code

The word code is a general term for the instructions in a computer program. The term is often used in contrast to data. The code is the process; the data is what is processed.

The term code is also used in the more conventional sense of a form of notation. See also *ASCII*.

command

A command is an instruction or request for the CLI (or a system task, such as an editor) to perform a particular action.

For instance, the command

```
>PRINT IZZY.TXT RET
```

directs the operating system to perform a series of operations. The operating system must find the file, queue it for the line printer, and print it.

The commands necessary to run an operating system are collectively called a command language. Two command languages are available on RSX-11M/M-PLUS systems, DCL and MCR. See the entries on these for more detail.

command dispatcher

The command dispatcher, MCR..., is a task that takes commands typed at a terminal and passes them to DCL or MCR or to some other CLI task that executes commands.

command line interpreter

See *CLI.*

command qualifier

A qualifier, preceded by a slash (/), that affects the operation of a DCL command is called a command qualifier. For instance, the DELETE command is used to delete files from directories, but the DELETE/QUEUE command is used to delete entries from the Queue Manager queue file of print jobs and batch jobs.

Compare with *filespec qualifier.* See also *qualifier.*

compiler

A compiler is a system task that translates a program written in a high-level language into an object module in binary machine code. Compilers are unbundled software. RSX-11M/M-PLUS supports a number of compilers, including the following high-level languages: FORTRAN-IV, FORTRAN-IV-PLUS and FORTRAN-77, BASIC-PLUS-2, and COBOL.

contiguous

A contiguous file consists of physically adjacent blocks on a mass storage device, such as a disk. Most files are scattered (noncontiguous) because this allows more efficient use of disk space. All task files (.TSK type) are made contiguous by the Task Builder. Often, data files are made contiguous because such files require less time for input/output operations, since the entire contents of a contiguous file can be brought into memory in a single read operation.

CPU

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

crash

A crash is the system's response to an unstable condition, particularly if the Executive is corrupted. Rather than continuing to operate and allowing the system to do itself damage, the system crashes. Many conditions can cause crashes. The Crash Dump Analyzer utility helps find the cause of a crash by formatting the contents of memory at the time of the crash. The dump must then be analyzed to determine the cause of the crash.

There is no concrete indication at your terminal that the system has crashed, but if the system has crashed, you will not be able to use your terminal.

See also *hang.*

CRT

CRT stands for Cathode Ray Tube, or video terminal. See also *terminal.*

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.

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. It provides a means of communication between the user and the operating system. DCL is designed for ease of use. Commands are English words, and if necessary elements are not typed in, DCL will prompt you for them. Compare with *MCR.*

decimal number

Numbers in the base-10 numbering system are called decimal numbers. The numerals 0 through 9 are used in this numbering system. This is the conventional numbering system.

In general, you need not worry about whether a number in a DCL command is octal or decimal. The documentation notes the few cases in which the numbering system used makes a difference. Displays that show decimal numbers generally indicate the fact by terminating the number with a decimal point. In the display from the DIRECTORY command, both octal and decimal numbers are shown.

See also *octal number.*

DECnet

DECnet is an unbundled family of hardware/software products that create distributed networks of DIGITAL computers.

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 sometimes have unexpected results. Defaults are always included in the documentation.

In RSX–11M/M–PLUS, a wide range of defaults is used, with the idea that the less that has to be specified in any given situation, the easier the system is to use and the smaller the chance of human error.

DEFINE command

The EDT DEFINE command enables you to customize EDT to suit your particular editing purposes, such as writing programs in a particular language or inserting text-formatting commands of some sort. The EDT DEFINE MACRO command enables you to combine a group of line-mode commands into a single line-mode command. The DEFINE KEY command enables you to combine a group of change-mode commands into a single command that you can execute with a single keystroke. See the *EDT Editor Manual* for further information.

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 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 elements of a character string, statement, or task.

For instance, in the filespec

```
TREK.TSK
```

the period (.) is the delimiter that enables the system to tell the difference between the file name TREK and the file type TSK.

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 (:), semicolon (;), slash (/), and comma (,). Spaces and tabs are also commonly used.

device

A device is any peripheral hardware connected to the processor and capable of receiving, storing, or transmitting data. Line printers and terminals are examples of record-oriented devices. Magnetic tapes and disks are examples of mass-storage devices. Terminal line interfaces and interprocessor links are examples of communications devices.

All devices have names in the form ddnn:, where dd is a two-letter mnemonic, nn is an octal number, and the colon (:) is a required terminator.

Devices are not necessarily hardware. See *pseudo device.*

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 as well as a unique number that identifies the file's actual location and points to a list of its file attributes. See *MFD* (Master File Directory) and *UFD* (User File Directory) for definitions of the two main types of directories.

The DCL DIRECTORY command displays directory information for specified files.

disk

The disk is the major form of mass-storage device on an RSX–11M/M–PLUS system. Disks are high-speed, random-access devices. There are several kinds of disks. Floppy disks are small, flexible disks. Hard disks are either fixed in place or removable. Removable disk types include a single hard disk enclosed in a protective case and a stacked set of disks enclosed in a protective case.

disk-based system

In disk-based systems, such as RSX–11M and RSX–11M–PLUS, the tasks and other functions that make up the operating system are stored on a disk and written into memory as they are needed by users.

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 screen or hard copy, the process is called echoing. Terminals are dual devices, sending input and receiving output. Typing on the terminal is sending input to the computer. Echoing is receiving output from the computer.

editor

An editor is a system task used for creating and altering text files. Three editors are supported on RSX-11M/M-PLUS. EDT, the DEC Editor, is introduced in this manual. The other editors are EDI, the Line-Oriented Editor, and SLP, the Source Language Input Program, a special programmer's editor. EDI is introduced in the *RSX-11M/M-PLUS Guide to Program Development.*

error message

Error messages are sent by the system when some action you have requested fails. Each error message identifies the particular part that detected the error.

The great majority of error messages result from typing mistakes 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, resource allocation, and supervision of input and output (I/O).

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.

FCS is one of two sets of file routines included on RSX-11M/M-PLUS systems. FCS permits one form of file organization and two forms of file access. See also *RMS-11.*

field

The term field usually refers to a portion of a command or a command element. The file name and file type are two fields of the filespec, for instance.

file

A file is a set of data elements arranged in a structure significant to the user. The file is one of the basic units of information on an RSX–11M/M–PLUS system. See also *volume.*

A file is any named, stored program or data, or both, to which the system has access. Access can be of two types: read-only, meaning the file is not to be altered, and read-write, meaning the contents of the file can be altered.

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 DIRECTORY command.

file-ID number

The file-ID number is a number in the form (mmmm,nnnn) that is displayed by the DCL DIRECTORY command and that is used by the file system to locate files on Files–11 disk volumes. In some commands, the file-ID number can be substituted for a filespec, but in general this number is used only by the system.

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.*

Files–11

Files–11 is the name of one of the file structures used on RSX–11M–PLUS. This same structure is also used on the RSX–11M Operating System. Files–11 volumes can pass from one of these systems to another with no file incompatibility. Volumes from other operating systems can be converted to Files–11 structure with FLX, the File Transfer program.

filespec

The filespec is the unique identification of a file that gives its physical location and generally an indication of its contents.

All files are specified in the following form:

```
ddnn:[g,m]filename.typ;version
```

For example:

```
DB0:[200,1]LOGIN.CMD;1
```

The device name takes the form ddnn:, where dd is a two-letter mnemonic and nn is an octal number. The colon (:) separates the device name from the UFD.

Next is the UFD, a pair of octal numbers enclosed in brackets ([ ]). The first number is the group number; the second is the member number.

File names can include 1 to 9 letters and numbers, but not other characters. The file name usually gives some indication of the contents of the file. The period (.) separates the file name from the file type.

File types can be from 0 to 3 letters or numbers. The file type usually gives some indication of the contents or purpose of the file. The semicolon (;) separates the file type from the version number.

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.
.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	FORTRAN 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.

The version number is an octal number that differentiates among various versions of files of the same name and type.

filespec qualifier

A filespec qualifier is a DCL qualifier, preceded by a slash (/), that is attached to a filespec to override some aspect of the command for that particular file.

Compare with *command qualifier.* See also *qualifier.*

form feed

A form feed is analogous to a line feed, but instead of moving down one line to resume printing, the line printer moves past the perforations in the paper to the top of a new form or page.

A form feed consists of a number of line feeds. The number differs for different forms and also for different output devices. You can insert a form feed in a text file by including a CTRL/L.

functionality

Functionality is a polysyllabic computer industry term for what the hardware or software can do.

global

In the RSX-11M/M-PLUS context, global means affecting the entire file, or the entire system, or the entire task, depending on the context. Global can mean changing all instances of a string in a file.

Here are some other examples. Any user can assign a new name to a device, but the assignment will not affect any other user. A privileged user can assign a new name to a device globally, meaning that all users are affected. Within tasks, a global symbol is one whose meaning is the same throughout the task, while a local symbol has meaning only within its own section of the task.

Global is a computer industry term with many meanings.

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, dormant, or locked.

Hung terminals may result from a busy system, a crash, or from checkpointing of the task to which you are sending input. Another possible cause is an error in a user-written task.

hard-copy terminal

Terminals that print output on paper are called hard-copy terminals. See also *terminal.*

hardware

Hardware is the physical computer equipment, including such mechanical devices as the line printer, the terminals, the mass-storage devices, and so forth.

Compare with *software.*

help file

A help file is a text file in a format suitable for use with the HELP command. Help files can include simply organized information and can provide up to nine levels of search. Help files describing DCL, MCR, and some of the system utilities are included in most installations. Some installations have help files for their applications tasks. Users can also write their own help files for their own UFDs.

high-level language

High-level languages, such as BASIC-11, BASIC-PLUS-2, FORTRAN-IV, FORTRAN-77, FORTRAN-IV-PLUS, and COBOL, are transportable programming languages. Programs in these languages are not tied to a particular kind of computer. They are called high-level languages because programs written in these languages usually provide a higher level of information about what the program will do.

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.

```
❶ >MACRO HIYA (RET)
❷ LINK HIYA (RET)
  TKB -- *FATAL*-FILE HIYA.OBJ;1 HAS ILLEGAL FORMAT


❸ >
❹ >LINK HIYA (RET)
❺ >
```

❶ The MACRO command was issued to assemble the file HIYA.MAC.

❷ 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.

❸ A prompt was issued following the failed LINK command.

❹ The MACRO command completed, causing another prompt to be issued. The LINK command issued in response to this prompt is completed successfully.

❺ A prompt was issued indicating completion of the LINK command.

indirect command file

Indirect command files provide a means of automatically passing commands to the operating system. In addition to simply passing commands, indirect command file directives permit you to use such programming techniques as loops, counters, labels, and symbol substitution to set up elaborate command sequences that can be altered through user interaction.

input file

Many system utilities and other tasks take existing files, alter them, and produce new files. For example, the MACRO–11 assembler takes a source file and produces an object file. In this case the source file is the input file and the object file is the output file.

One common mistake made in using the system is confusing the input and the output files. DCL usually prompts for these files, but most system utilities require you to identify your input and output files by position in a command line. You should be sure of the syntax for the command you are using.

See *output file.*

install

An installed task is one that is named in the System Task Directory (STD), a list of Task Control Blocks (TCBs) which contain information about each task.

Taking a task out of the STD is called removing it.

Users automatically install and remove their tasks through the RUN command. Privileged users can also install and remove tasks explicitly.

A task cannot execute unless it is installed.

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

In an interactive system, the user and the operating system communicate directly via a terminal. The operating system immediately acknowledges and acts upon requests entered by the user at a terminal. RSX-11M and RSX-11M-PLUS are interactive systems.

journaling

EDT automatically records your editing session as it goes along. If the system crashes for any reason while you are editing, the record is preserved. You can then restore your file to where it was before the crash using special commands to EDT. This feature is called journaling. 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.

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 with the system, but there may also be user libraries prepared for your installation.

Libraries can also be any other collection, such as text messages.

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, the DEC Editor, has two main modes of operation: line mode and change mode. Line mode operates on a line or group of lines and is well-suited to manipulating large blocks of text.

See also *change mode.*

line number

EDT and some other editors automatically assign numbers to the lines in a text file. Editors follow different systems of numbering. The numbers are useful in finding and manipulating the contents of the file.

line pointer

EDT and some other editors use 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 a line at a time. It is used for printing large amounts of output that would otherwise tie up a slower device. Almost every system has a device designated as the line printer. In some cases, the "line printer" will actually be a high-speed terminal.

On most systems, the line printer is under control of the Queue Manager, which assures orderly output to the line printer, one file at a time.

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.

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. Logging in requires either the MCR HELLO command or the DCL LOGIN command, plus a User Identification Code (UIC) or name, and a password.

logout

Logging out informs the operating system that you have finished using a particular terminal. Logging out requires the LOGOUT command. Logging out aborts all nonprivileged tasks active from your terminal and end your private access to any mass-storage devices.

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/M-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 got 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/M-PLUS systems are written in MACRO-11. 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, but which can be called by a single name.

MACRO-11 includes a number of functions designed to make programming easier, including directives to divide programs into sections, conditional assembly directives, a comprehensive system macro library, and user-defined macros and 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.

mass-storage device

A mass-storage device is an input/output device where data and other types of files are stored while they are not being used. Typical mass-storage devices include disks, magnetic tapes, floppy disks, and DECtapes.

Each mass-storage device uses a particular magnetic medium to hold its data. If the data is organized in Files–11 format, that data is called a volume. See *file-structured device*, and *volume*. See also the *RSX–11M/M–PLUS Command Language Manual*.

Master File Directory

See *MFD*.

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*.

medium

The medium is the physical disk or tape that carries magnetically encoded information.

The plural of medium is media. Compare with *volume*.

memory

Memory is a series of physical locations into which data or instructions can be placed in the form of binary words. Each location in memory can be addressed and its contents can be altered.

When a task is run, it is installed in memory. From memory, the task has access to the *CPU*.

Memory should not be confused with *mass-storage devices*.

MFD

Each Files–11 volume includes a Master File Directory (MFD). The MFD is a file containing pointers to all UFDs on the volume. UFD is a file containing pointers to all files in the UFD. See the *RSX–11M/M–PLUS Command Language Manual*.

mnemonic

Mnemonic is a hard-to-remember word that means aid to memory. It is pronounced ne-MON-ic. PIP is a mnemonic for Peripheral Interchange Program.

Most mnemonics are acronyms. See *acronym*.

Monitor Console Routine

See *MCR.*

monitor level

Command Line Interpreters (CLIs) are sometimes called terminal monitors. They monitor the activity that is taking place on your terminal that concerns the operating system. 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.*

multiprogramming

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

multiuser

A multiuser system, such as RSX-11M or RSX-11M-PLUS, permits a number of users to work on their terminals with little or no interference among users.

Most systems include facilities to protect you from having your work affected by the work of another user. These multiuser protection facilities include the LOGIN and LOGOUT commands, the file protection system, and the ability to make a device your private device through the ALLOCATE command.

object file

See *object module.*

object module

An object module — a file with the type .OBJ — is a program that has passed through the assembler or compiler and is ready to go to the Task Builder. It has been translated from source languages into an object module that can be linked by the Task Builder to produce an executable task image. A task image may require one or more object modules.

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.

In RSX-11M/M-PLUS, file version numbers, UICs and UFDs, and device numbers (including terminal numbers) are octal numbers. In most cases, DCL does not require you to distinguish octal from decimal numbers. Where the distinction is important, it will be noted in the documentation.

operator

Every installation has an operator, some person responsible for maintaining the system. In small systems, the job may be combined with that of the system manager or informally divided among several people, but regardless of the arrangements, someone must take care of the work of changing ribbons, rebooting the system, keeping records, and so forth. Usually the operator is privileged and knowledgeable about your installation.

operating system

An operating system is a set of tasks that collectively automate the management of computer resources to provide efficient computer operation.

An operating system is used for user communication with the computer, for program development, and for scheduling the use of the central processing unit and its peripherals most efficiently.

An operating system includes three basic elements:

1. The Executive, which controls the system and user tasks in the operating system
2. The file system
3. The utility tasks, such as editors, file-handling routines, and other specialized facilities

In addition to the operating system, an installation usually includes one or more programming languages and a number of applications tasks.

output file

Many system utilities and other tasks take existing files, alter them, and produce new files. For example, the Task Builder takes object modules and transforms them into a task image. In this case, the object modules are input files and the task image is the output file.

See *input file.*

parse

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

A PRINT command without a filespec, or with illegal characters in the filespec, 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 keyword associated with a particular user. When logging in, you must supply the correct password before the system will allow you access. In most cases, your password will not appear on your terminal.

PDP–11 computer

The PDP–11s are a family of computers manufactured by DIGITAL. PDP means Programmable Data Processor. The RSX–11M Operating System is designed for use on all PDP–11s. The RSX–11M–PLUS Operating System is designed for use on PDP–11s with 22-bit addressing.

peripheral devices

Any unit, distinct from the CPU and memory, that can provide the system with input or accept output from it, is called a peripheral device or peripheral. Terminals, line printers, and disks are 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.

For RSX–11M–PLUS systems, see also *secondary pool.*

print head

The print head is the moving mechanism on a hard-copy terminal that prints characters on the paper. On many hard-copy 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.

privileged

In general, privileged commands or tasks are allowed to perform operations normally considered the domain of the monitor or Executive or which can affect system operations as a whole.

There are privileged tasks, privileged users, and privileged terminals.

A privileged task can refer to areas outside its partition, namely to the Executive and to the I/O page. Such tasks can move data outside their own dedicated areas and can thereby interfere with the proper operation of the system.

Nonprivileged users can run installed privileged tasks.

Privileged users have group numbers lower than or equal to 10.

A terminal with a privileged user logged in is a privileged terminal. Privileged users can set privileged terminals nonprivileged.

program

A program is a series of actions aimed at a particular result, a process. Programming languages are a means of describing procedures so that they can be performed by a computer.

See also *task*.

prompt

A prompt is a sign that the system is ready to accept input from you.

The TYPE command prompts

```
File?
```

if you do not name a file to be typed.

DCL prompts with a right-angle bracket (>), called the implicit prompt, or three letters plus the bracket (DCL>), called the explicit prompt.

EDT prompts with an asterisk (*) in EDT command mode.

protection code

Each file has a protection code that specifies what access different categories of system users may have to the file and what they may do to the file when they access it.

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 (both group and member number) as that the file was created under
3. GROUP — Users with the same group number as that the file was created under
4. WORLD — Other users

There are also four kinds of access to files:

1. READ ACCESS — A user, or a user's tasks, may read, copy, print, or type the file.
2. WRITE ACCESS — A user, or a user's tasks, may add new data to the file by writing to it.
3. EXTEND ACCESS — A user, or a user's tasks, may change the amount of disk space allocated to the file.
4. DELETE ACCESS — A user, or a user's tasks, may delete the file.

The system default protection code is expressed as follows:

(SYSTEM:RWED,OWNER:RWED,GROUP:RWED,WORLD:R)

You can display the protection code for files with the DIRECTORY/FULL command. You can set the protection on your own files with the SET PROTECTION command.

pseudo device

A pseudo device is an entity treated as an I/O device by the user or system, although it is not any particular physical device. It is a forwarding address through which actual physical devices can always be reached.

This convention makes it possible to refer to a device without knowing its physical name and number. Thus, the pseudo device TI: always refers to the terminal you are using, no matter what the number or whether it is local or remote, hard-copy or video.

See the *RSX-11M/M-PLUS Command Language Manual* for more information on pseudo devices.

QMG batch job

The SUBMIT command defines a QMG batch job. A QMG batch job is made up of one or more user batch jobs. The jobs in the QMG batch job are processed in the order they are submitted, on the same batch processor, and without interruption.

Batch processing is available on RSX-11M-PLUS systems only.

qualifier

A qualifier in DCL is always preceded by a slash (/). The qualifier alters the action of a command. Often, qualifiers override defaults. For instance, the command

```
>PRINT IZZY.TXT RET
```

specifies no page length, meaning that the file could be printed over the perforations in line printer paper. Adding the /LENGTH qualifier to the command can prevent the file from being printed over the perforations, as in this example:

```
>PRINT/LENGTH:60 IZZY.TXT RET
```

This example illustrates a command qualifier, which alters the command itself. DCL also uses file qualifiers, which alter the effect of a command for a file specified with the command. For instance, the command

```
>PRINT/COPIES:2 OZY.TXT, IZZY.TXT, FIZZY.TXT RET
```

causes two copies of each file to be printed at the line printer: two copies of OZY.TXT, followed by two copies of IZZY.TXT, followed by two copies of FIZZY.TXT. A file qualifier can override the command qualifier for that particular file. Thus, the command

```
>PRINT/COPIES:2 OZY.TXT, IZZY.TXT/COPIES:1, FIZZY.TXT RET
```

results in two copies of OZY.TXT, followed by one copy of IZZY.TXT, followed by two copies of FIZZY.TXT.

queue

A queue is a waiting line, a list of items to be processed according to system or user priorities. There are many queues in an RSX-11M/M-PLUS system.

Queue Manager

The Queue Manager is a system task that controls queues of jobs directed to batch processors, line printers, or other output devices.

random access

This term refers to memory or mass-storage devices where all information is equally accessible. With random access, the next location from which data is to be obtained is not dependent on the location of the last data obtained.

All records appear to be adjacent on a random-access device. As far as the user is concerned, there is no beginning, middle, or end to the data.

See also *sequential access.*

read

When a task is accepting data, it is said to be reading. This is a standard term in the computer industry. When you issue a TYPE command, the system must read the designated file from the disk and write it to the terminal. See also *write.*

real time

RSX-11M and RSX-11M-PLUS are real-time systems.

Real-time jobs require response to physical events as they occur.

Real-time computation is performed while a related or controlled process is occurring so that the results of the computation can be used in the process.

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.*

relocatable addresses

When a routine is moved from one memory location to another, its address references must be changed so that the routine will execute at its new location. In the terms of this manual, relocatable addresses are the provisional memory addresses assigned by the assembler or compiler for object modules. These addresses are assigned as if the resulting task would have the computer to itself. The Task Builder changes these relocatable addresses into addresses that the system can use when the object modules are linked.

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

Any data or instructions located in the main memory are said to be resident in memory.

resolve

When the Task Builder defines symbol references or changes relocatable addresses for a running task, the Task Builder is said to have resolved these references to the symbols or to the relocatable addresses.

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–11 is the more sophisticated of two sets of routines supplied on RSX–11M/M–PLUS systems. The routines are used to open and close files, read from files, write to files, and extend and delete files.

RMS–11 supports three forms of file organization and three forms of file access. See *FCS.*

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.

scope

Video terminals are often called scopes. See also *terminal.*

scroll

When more than a screenful of output is sent to a video terminal, the output 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.

secondary pool

On RSX–11M–PLUS systems only, some data for Executive functions is moved from the pool to a secondary pool. This data includes the Task Control Blocks (TCBs) for prototype tasks, such as ...PIP; some data used by devices and the file system; and accounting information.

sequential access

Sequential access to data means that records or files are read one after another in the order in which they appear in the file or volume.

A deck of punched cards or a magnetic tape is a sequential-access medium. If you are half way through the tape and wish to read some record that is a 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.

Compare with *random access.*

significant event

A significant event is declared whenever there is a change in system status. Whenever there is a significant event, the Executive reviews the eligibility of tasks to execute, because the change that caused the significant event to be declared may mean that a priority task that was blocked is no longer blocked.

For instance, a significant event is declared when a task completes its execution or when a task cannot continue I/O because of the unavailability of an output device. The round-robin scheduler causes a significant event to occur regularly.

slave terminal

A terminal that sends and receives I/O from a task and not directly from the operating system is called a slave terminal and is said to be "set slaved". No system commands can be entered from a slaved terminal. Slaved terminals communicate only with the tasks that control them.

software

Software is the collection of tasks, procedures, rules, and documentation associated with the operation of a particular computer system. The operating system is software. EDT is editing software. The MACRO–11 Assembler is software.

Compare with *hardware.*

source file

A source file is a text file containing material suitable for translation into an object module by an assembler or compiler. Such files cannot be run or task built. The Task Builder turns object modules into task image files.

source language

Most programming languages are source languages. These languages describe the procedure you wish the computer to follow. They are the source of the task that is actually run on the computer.

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 SHOW TASKS/INSTALLED command.

string

A string is a sequence of characters. When you use an editor to search for a word or phrase, you are searching for a string. The sequence of characters that forms a command is often called a command string.

subcommand

In EDT, the DEC Editor, commands entered in character mode are called subcommands to distinguish them from commands entered in command mode.

subroutine

A subroutine is a routine that can be used as part of another routine. For instance, you might write a routine to print the time in large numbers on your terminal. You could then call that routine as a subroutine in some task that required printing the time in large numbers.

supported

Briefly, support refers to the obligations that DIGITAL has to its customers.

RSX-11M/M-PLUS systems allow for many variations of hardware and software. While a particular item of hardware or software may not be present on every system, that hardware or software is considered supported if it is possible to include it without a special effort. For instance, not all systems include the RX01 floppy disk hardware, but all systems include the DX: driver for this hardware. Therefore, the RX01 is supported. The Software Product Description (SPD) for your system identifies what is supported.

Support also refers to the assistance your installation gets from DIGITAL in setting up and operating the system. Support of this sort is defined at the time the system is purchased.

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 is the form that a command must follow. Misspelled words are the most common syntax errors.

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 converting binary numbers to decimal, saving the contents of registers, formatting input and output, and managing memory.

System library routines can also be called by user tasks. See the *IAS/RSX-11 System Library Routines Reference Manual* for further information.

system task

A task that performs *system-level* functions is called a *system* task. Thus, a system task is any task that is part of the basic operating system, such as an editor or other system utility. See also *applications task.*

System Task Directory

See *STD.*

task

The task is the fundamental, executable programming unit. It may include one or more routines taken from a library or routines written for a particular purpose.

Many DCL or MCR commands that you enter are tasks. Any utility you invoke is also a task.

Task Builder

Essentially, the Task Builder fixes the values of external or relocatable symbols in the object module, thus transforming the object module, or several such modules, into an executable task image. This is called linking. The Task Builder can be invoked using the DCL LINK command.

The Task Builder also allocates the physical and virtual address space needed for a task.

Task Control Block

See *TCB.*

task image file

The contiguous file containing a runnable image of a task is called a task image file. This file is built from one or more object modules by the Task Builder.

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

TCB

TCB stands for *Task Control Block.* Each installed task has a TCB in the Dynamic Storage Region (pool). The TCB contains all the information needed to run the task. The TCB is created when the task is installed and eliminated when the task is removed. The System Task Directory (STD) consists of TCBs.

terminal

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

- Hard-copy terminals keep a permanent record of output on paper.
- Video or CRT (cathode-ray-tube) terminals have a video screen for receiving output.

text file

Text files are those files written in ASCII code that can be read by both humans and by software.

See also *source file.*

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 and RSX-11M-PLUS are fundamentally real-time systems, the round-robin scheduler provides a form of time sharing called time slicing.

See also *round-robin scheduler* and *real time.*

translator

DCL is basically a translator. When you type a command to a DCL terminal, DCL first checks to make sure the command is in proper syntax and then translates the DCL command into the appropriate MCR or utility command.

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.

The term "transparent spooling" refers to the capability of RSX-11M-PLUS systems with the Queue Manager to print output at a line printer by using the name of the line printer in place of an output file specification.

tube

A video terminal is sometimes called a tube. See also *terminal.*

UFD

Files are contained in User File Directories (UFDs). The UFD is a file listing the files in a Directory. The UFD is a two-number code in the form [g,m] that is in every file specification, either explicitly or by default, and that locates the file.

In most cases, the UFD will be the same as the User Identification Code (UIC), under which the user logs in. Nonprivileged users can go from UFD to UFD, but they cannot change their UIC. These terms are often used interchangeably, but strictly speaking the UIC identifies the user and the UFD identifies a collection of files.

See also *UIC*.

UIC

Each RSX-11M/M-PLUS user has a two-number identification code enclosed in brackets that is 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.

See also *UFD*.

unbundled

Software that is not supplied as part of the basic RSX-11M-PLUS Operating System is called "unbundled" software. The high-level languages, such as FORTRAN or COBOL, are unbundled software, and must be purchased under a separate license.

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 also *QMG batch job*.

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

Batch processing is available in RSX-11M-PLUS systems only.

User File Directory

See *UFD*.

User Identification Code

See *UIC*.

user task

See *applications task*.

utility

A utility is a general-purpose task included in an operating system to perform common functions, such as editing or file handling.

video terminal

A video terminal is a terminal with a video screen for accepting output. See *terminal.*

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. Virtual terminals are supported on RSX-11M-PLUS only.

volume

The volume is the largest logical unit of the file structure. A volume contains files. The volume may either be Files-11 format or not. Only Files-11 volumes can be accessed by the system.

The term volume is often used as a synonym for *device*, because you name the volume a file is located on by supplying a device name in the filespec. The device name you supply is the name of the device on which the volume you wish to access has been mounted.

wildcard

A wildcard is an asterisk (*) or per cent sign (%) used to replace parts of a file specification included in a command. With a wildcard, one file specification can specify more than one file.

The asterisk means "match zero or all characters in this position."

The per cent sign means "match exactly one character in this position."

See the *RSX-11M/M-PLUS Command Language Manual* for a full description of how the wildcards work.

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 term in the computer industry. When you issue a PRINT command, the file is read from wherever it is stored and written to the line printer.

See also *read.*

Index

Introduction to
RSX-11M and RSX-11M-PLUS
Order No. AA-L763A-TC

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

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.

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) _____

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

BSSG PUBLICATIONS ZK1-3/J35
DIGITAL EQUIPMENT CORPORATION
110 SPIT BROOK ROAD
NASHUA, NEW HAMPSHIRE 03061

Do Not Tear - Fold Here

Cut Along Dotted Line